PhUSE US Connect 2018

Paper DH09

Managing Programs Development Life-Cycles in SAS LSAF

Jean-Marc Ferran, Qualiance, Copenhagen, Denmark

ABSTRACT

Managing efficiently programs development life-cycles is often a challenge in clinical study reporting. Companies have been using a mix of naming conventions, spreadsheets and processes utilizing subfolders to identify programs status during the iterative code development and validation over the course of a study. Life-cycle management using manual processes is error-prone and can result in audit findings.

SAS LSAF (Life Science Analytics Framework - previously SAS Drug Development) provides modern capabilities around versioning and eSignature. These capabilities in combination can help study teams to implement simple and efficient life-cycles management on programs and other artefacts. Dedicated APIs also enable users to consume related attribute values and develop reports and metrics to monitor progresses of their studies.

This presentation will elaborate on LSAF features supporting life-cycle management, their implementation and present real-life examples.

INTRODUCTION

This paper intends to highlight concepts for a simple Life Cycle Management process around programs to help companies move away from spreadsheets while not creating an overhead for their programmers.

The concepts are based on SAS LSAF, a web- and cloud-based Statistical Computing Environment and examples are based on SAS LSAF version 4.7. LSAF formerly known as SAS Drug Development has been on the market for over 15 years and in its last suite provides an interesting set of features to control code development. Users can copy and develop code in a Workspace that only them can edit and publish and share their code in a Repository using check-in/check-out principle.

The 2 features that this paper focus on are versioning and eSignature. While certain companies use an “horizontal” way of doing versioning where each study has 3 mirrored hierarchies (e.g. Development, Validation and Production) where programs and artefacts are copied across, this paper assumes one hierarchy only for each study while artefacts and programs in particular can have several versions and statuses.

The use of spreadsheets and program header for tracking progress often leads at some point in the study to programs’ statuses being out-of-sync between the 2. Both versioning and eSignature capabilities are supported by APIs in LSAF and it is possible to programmatically consume such information and compute such statuses rather than relying on a programmer to keep different artefacts and documents in sync.

Having an accurate view on program statuses and access to granular Life Cycle information enables to then create Dashboards and analyze further programs development time and iterations and identify potential bottlenecks across studies or projects.

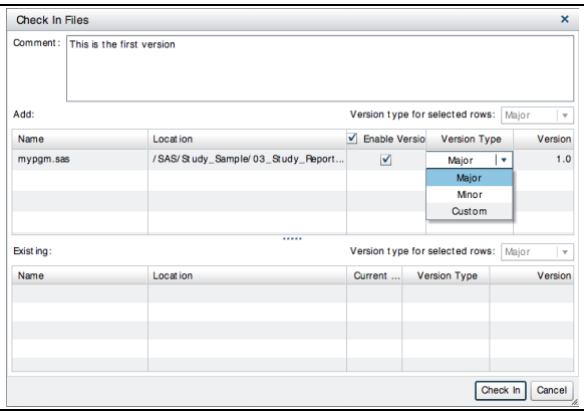
This paper also explores how dependencies between artefacts and their validation statuses could be handled using the concept of LSAF jobs and manifests.

VERSIONING AND E-SIGNATURE IN LSAF

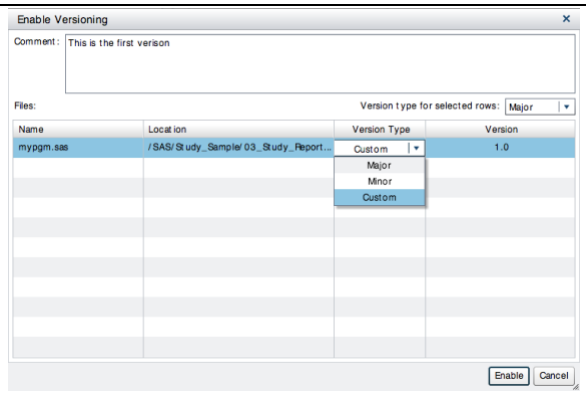
This section details how to use versioning and e-signature in LSAF as well as their features and how to consume then such information using LSAF APIs.

USAGE IN LSAF

LSAF provides a versioning capability that user can use on any type of objects. In particular, the versioning in LSAF enables users to save minor and major versions of their code as well as adding comments. It is also possible to create custom versions and skip next minor or major version. The table below illustrates how this can be done from the Workspace or Repository.

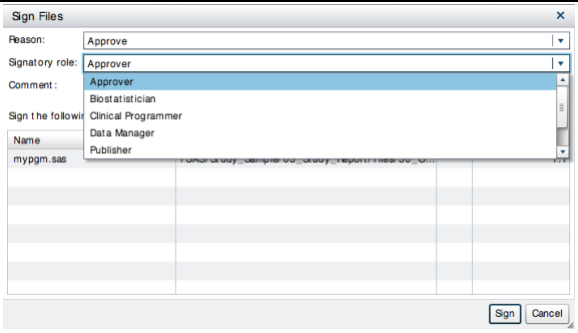


Turning versioning at file check-in from Workspace

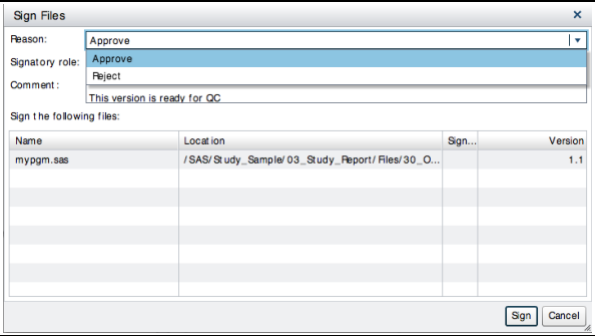


Enabling versioning in Repository

Once an object is under versioning, it is possible to e-sign the latest version with different roles and reasons. Roles are defaulted to “Approver”, “Reviewer”, “Clinical Programmer”, “Statistician”, “Data Manager” and “Publisher” and Reasons to “Approve” and “Reject” and these values can be customized to meet internal business processes’ needs. E-Sign only requires then user to enter their password.

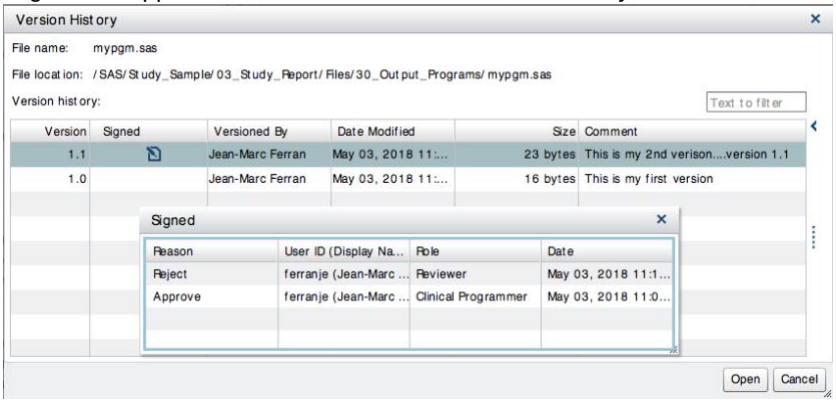


Select Signatory Role



Select Signatory Reason and Add Comment

All versions with the signatures applied are then available in the version history:



Objects versions and their properties can then be open, reviewed or reused. eSignature information is available for every version that is signed as indicated by the icon in the “Signed” column of the version history screen.

The advantage of the versioning and the eSignature features in LSAF is that they are simple and only few clicks away for the user to turn on, apply, access and review, which does not create a significant overhead compared to working with a simple fileshare.

CONSUMPTION VIA LSAF MACRO APIS

One of the most advanced feature of LSAF is the suite of Java and Macro APIs that are available ([1]) to enable users to optimize processes and automate tasks ([2]). LSAF APIs provide an extensive coverage of both operational tasks (e.g. file and folder manipulation) and more advanced features (e.g. create a job) using Java or SAS Macro programming languages. The LSAF APIs cover in particular retrieval of information about versions and e-signature.

LSAF Macro APIs %LSAF_GETCHILDREN enables to get information about objects across a hierarchy and %LSAF_GETVERSION enables to get information about all versions of a given object. Combining both enables to get an overview of all versions of all objects of a certain type (e.g. SAS programs only) in a SAS dataset.

name	versionLabel	isCheckedOut	isLocked	isSigned	createdBy	dateCreated	lastModifiedBy	lastModified
program1.sas	1.0	0	0	1	dev_jfe	11MAY16:08:29:28	dev_jfe	Wed May 11 08:29:28 GMT 2016
program2.sas	1.0	0	0	1	dev_jfe	11MAY16:08:29:29	dev_jfe	Wed May 11 08:29:29 GMT 2016
program3.sas	2.0	0	0	1	dev_jfe	11MAY16:08:29:29	ext_training_018	Wed May 11 14:15:02 GMT 2016
program3.sas	1.0	0	0	1	dev_jfe	11MAY16:08:29:29	dev_jfe	Wed May 11 08:29:29 GMT 2016
program4.sas	1.0	0	0	1	dev_jfe	11MAY16:08:29:29	dev_jfe	Wed May 11 08:29:29 GMT 2016
program5.sas	2.0	0	0	0	ext_training_016	11MAY16:13:27:34	ext_training_018	Wed May 11 13:43:37 GMT 2016
program5.sas	1.0	0	0	1	ext_training_016	11MAY16:13:27:34	ext_training_016	Wed May 11 13:27:34 GMT 2016

SAS datasets including all version information of programs using %LSAF_GETCHIDREN and %LSAF_GETVERSION

Once all versions are identified, LSAF Macro API %LSAF_GETSIGNATURES can be used for objects identified where isSigned=1 in order to retrieve all eSignatures of all eSigned versions.

name ▲	versionLabel	userId	reason	role	comment	dateSigned
program1.sas	1.0	ext_training_016	Approve	Reviewer		11MAY16:13:23:43
program1.sas	1.0	dev_jfe	Approve	Statistical Programmer		11MAY16:08:30:39
program2.sas	1.0	dev_jfe	Approve	Statistical Programmer		11MAY16:08:30:38
program2.sas	1.0	ext_training_017	Approve	Approver		11MAY16:13:39:59
program3.sas	2.0	ext_training_018	Approve	Clinical Programmer		11MAY16:14:15:36
program3.sas	1.0	dev_jfe	Approve	Statistical Programmer		11MAY16:08:30:38
program3.sas	1.0	ext_training_016	Reject	Reviewer		11MAY16:13:26:01
program4.sas	1.0	dev_jfe	Approve	Statistical Programmer		11MAY16:08:30:38
program5.sas	1.0	ext_training_016	Approve	Statistical Programmer		11MAY16:13:29:35
program5.sas	1.0	ext_training_017	Approve	Reviewer		11MAY16:13:40:58

SAS datasets including all version information of several programs

The attributes provided by %LSAF_GETSIGNATURES enables to identify users who signed, the reason, the role, the comment and the date of signature among others. In the context of developing a development life cycle for programs, such attributes can be used to compute the status of a program using simple rules.

COMPUTING PROGRAM STATUS

This section assumes the use of a Programming Plan (aka the "ProgPlan") and a Validation Tracker in the form of spreadsheets or other that can be consumed and read in LSAF. Using the versioning and e-Signature standard capability of LSAF (i.e. reason is "Approve" or "Reject") and assuming that a Programmer (Any roles not in "Approver" or "Reviewer") and a Validator (Roles "Approver" and "Reviewer") are required in the Life Cycle of a program. There could be more than one Reviewer where different reviewers have different focus such as code and output reviews, which could be tackled by extending the list of Roles or Reasons or handle objects with their dependencies (See next section).

A program is considered validated here if the last version is both eSigned by a Programmer and Validator (generally different users), preferably in this chronological order.

PROGRAM STATUS

The table below describes 7 different statuses to consider based on:

- Whether the program is planned to be developed – i.e. is listed in ProgPlan or has been created on the go
- Whether the program is under versioning – i.e. this would indicate that the code has reached some level of maturity
- Whether the program is checked-out – i.e. a programmer is working on it
- Whether the program is signed by Programmers and Reviewers – i.e. the program is ready for validation, failed validation or validated.

#	Status	Status Description
0	Not Started	Program is in ProgPlan but not in Repository
1	Draft not versioned	Program is in Repository but not versioned
2	Draft versioned	Last version is not signed
3	Draft not versioned / Checked-out*	If the program is checked-out but not versioned
4	Draft versioned: Checked-out*	If the program is checked-out and versioned
5	Ready for validation	Last version is signed by Programmer as "Approve"
6	Failed validation	Last version is signed by Validator as "Reject"
7	Validated	Last version is also signed by Validator as "Approve"

*: APIs do not allow to recover eSign info from versions of of an item that is checked-out

Possible statuses of a Program using a ProgPlan and LSAF eSignature capability

The important aspect is that all the information required (is in ProgPlan?, is versioned?, is checked out?, is signed?, What signature attributes?) can be retrieved programmatically. Logic can be implemented in order to compute these 7 statuses.

One of the main benefit of this simple Life Cycle is that when a program has status "Validated" (Status 7) and a user purposely or by mistake update a program, a new version is automatically created with no eSignature information and Status of the program is then "Draft versioned" (Status 2).

The status can be documented in a report (See section Creation of Status Reports) or in a LSAF extended attribute that can be added by object type. If the later, such attribute can be populated programmatically using LSAF API Macro %LSAF_UPDATEPROPERTIES.

HANDLING OF EXCEPTIONS

The process of using eSignature is rather liberal in LSAF and is not dictated by an in-built workflow. As a result, some exceptions and warnings can be found and reported. Such exceptions or warnings are the result of misuse of the Life Cycle and associated processes.

In particular, having same programmer as validator is often not accepted although certain companies use a risk-based approach when in certain cases (e.g. low complexity and low impact), self-validation is possible. This situation can easily be spotted and reported as an exception. The situation where no programmer has signed and only a validator could also be reported as an "exception" or "warning" depending on how rigid the Life Cycle is around establishing that a program is ready for validation.

#	Exceptions reported as an issue
0	Programmer and Validator are the same
1	Only Validator has signed

Possible exceptions to report as an issue

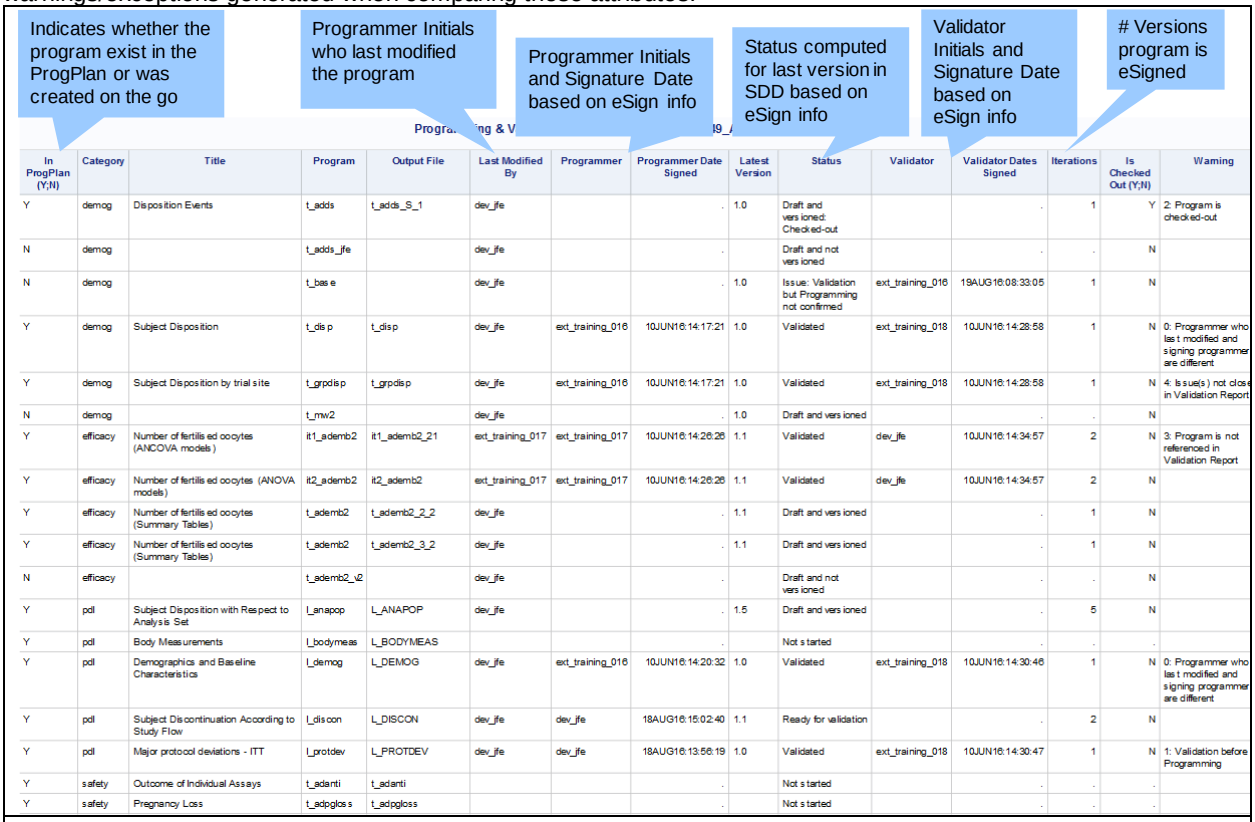
There are other situations that may be considered for review while impact on the quality of the code and reality of validation status may be low. Anyone can eSign a program they have write access to and the user eSigning the program as a Programmer could be different than the last user who modified the code (which is recorded in the program version's attributes). This case (Exception 0) could be due to illness or programmer winning at the lottery and never turning up again. A Validator can also eSign before a Programmer (Exception 1) in case Validator has been notified over email or verbally and just went on with the task while the Programmer forgot to eSign the last version. The last 2 exceptions (Statuses 4 and 5) are related to the use of Validation Tracker or Report where no documentation or issue status may be out of sync with the Program Status based on eSignature information. In this case, a review and update of the Validation Tracker may be necessary or would indicate that certain issues haven't been taken care of.

#	Exceptions reported as warnings
0	Programmer who last modified and Signing Programmer are different
1	Validator eSigns before the Programmer
2	Program is checked-out
3	Program has validated or failed validation status but program is not referenced in Validation Report
4	Program has validated status but program still has open issues in Validation Report

Possible warnings to report

CREATION OF STATUS REPORTS

The logic can be run across all programs of a given study and across domains (e.g. demographics, efficacy, safety, etc.) and statuses can be computed for each program and listed in a report with both relevant LSAF attributes and warnings/exceptions generated when comparing these attributes.



In ProgPlan (Y;N)	Category	Title	Program	Output File	Last Modified By	Programmer	Programmer Date Signed	Latest Version	Status	Validator	Validator Dates Signed	Iterations	Is Checked Out (Y;N)	Warning
Y	demog	Disposition Events	t_adds	t_adds_S_1	dev_jfe		.	1.0	Draft and versioned: Checked-out		.	1	Y	2: Program is checked-out
N	demog		t_adds_jfe		dev_jfe		.		Draft and not versioned		.	.	N	
N	demog		t_base		dev_jfe		.	1.0	Issue: Validation but Programming not confirmed	ext_training_016	19AUG16:08:33:05	1	N	
Y	demog	Subject Disposition	t_disp	t_disp	dev_jfe	ext_training_016	10JUN16:14:17:21	1.0	Validated	ext_training_018	10JUN16:14:28:58	1	N	0: Programmer who last modified and signing programmer are different
Y	demog	Subject Disposition by trial site	t_grpdisp	t_grpdisp	dev_jfe	ext_training_016	10JUN16:14:17:21	1.0	Validated	ext_training_018	10JUN16:14:28:58	1	N	4: Issue(s) not close in Validation Report
N	demog		t_mw2		dev_jfe		.	1.0	Draft and versioned		.	.	N	
Y	efficacy	Number of fertilised oocytes (ANCOVA models)	it1_ademb2	it1_ademb2_21	ext_training_017	ext_training_017	10JUN16:14:26:26	1.1	Validated	dev_jfe	10JUN16:14:34:57	2	N	3: Program is not referenced in Validation Report
Y	efficacy	Number of fertilised oocytes (ANOVA models)	it2_ademb2	it2_ademb2	ext_training_017	ext_training_017	10JUN16:14:26:26	1.1	Validated	dev_jfe	10JUN16:14:34:57	2	N	
Y	efficacy	Number of fertilised oocytes (Summary Tables)	t_ademb2	t_ademb2_2_2	dev_jfe		.	1.1	Draft and versioned		.	1	N	
Y	efficacy	Number of fertilised oocytes (Summary Tables)	t_ademb2	t_ademb2_3_2	dev_jfe		.	1.1	Draft and versioned		.	1	N	
N	efficacy		t_ademb2_v2		dev_jfe		.		Draft and not versioned		.	.	N	
Y	pdl	Subject Disposition with Respect to Analysis Set	l_anapop	L_ANAPOP	dev_jfe		.	1.5	Draft and versioned		.	5	N	
Y	pdl	Body Measurements	l_bodymeas	L_BODYMEAS			.		Not started		.	.	.	
Y	pdl	Demographics and Baseline Characteristics	l_demog	L_DEMOG	dev_jfe	ext_training_016	10JUN16:14:20:32	1.0	Validated	ext_training_018	10JUN16:14:30:46	1	N	0: Programmer who last modified and signing programmer are different
Y	pdl	Subject Discontinuation According to Study Flow	l_discon	L_DISCON	dev_jfe	dev_jfe	18AUG16:15:02:40	1.1	Ready for validation		.	2	N	
Y	pdl	Major protocol deviations - ITT	l_protdev	L_PROTDEV	dev_jfe	dev_jfe	18AUG16:13:56:19	1.0	Validated	ext_training_018	10JUN16:14:30:47	1	N	1: Validation before Programming
Y	safety	Outcome of Individual Assays	t_adanti	t_adanti			.		Not started		.	.	.	
Y	safety	Pregnancy Loss	t_adpgloss	t_adpgloss			.		Not started		.	.	.	

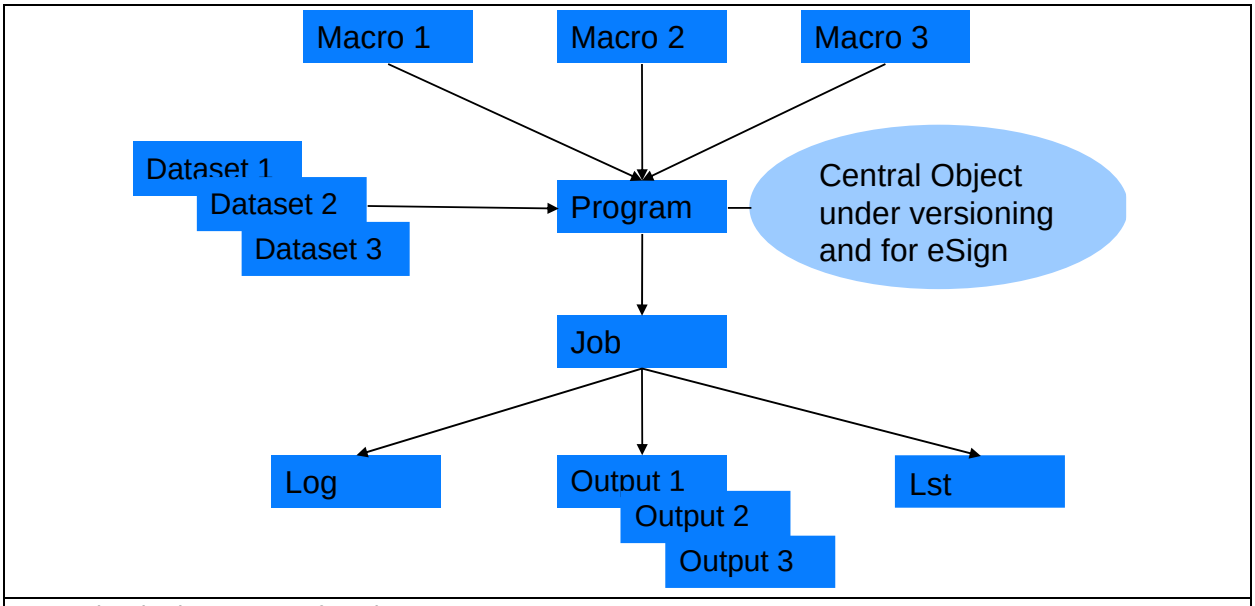
Status Report Example

Such report can be created for every study and associated dashboards with metrics informing the advancement of the study (eg. % of programs validated, % of programs in draft, etc.) can be created per project, therapeutic area or globally.

Such report can also provide other indicators such as the number of versions the program is eSigned. This metric is particularly interesting as it provides an indication to whether a program has been cumbersome to develop and validate. If many iterations, this may indicate that the specifications are not clear, a standard program or additional training may be required.

EXTENSION TO HANDLE DEPENDENCIES

The concepts developed so far do not touch upon the issues around validation status and dependencies with other artefacts that require validation too. The picture below illustrates the different artefacts that are linked to a program in LSAF. In particular there is the concept of Jobs that enable to trace fully the input and output of a given program or set of programs execution. LSAF Jobs must be created for any program(s) execution in Repository and produce a Manifest that list all inputs and outputs and the central programs themselves.



Dependencies between artefacts in LSAF

While the program is the central object that would be put under versioning and eSign, Datasets and Macros in particulars are also subject to validation and a program cannot be considered validated if the input are not validated. In some cases, the input artefacts can be implicitly validated through the validation of the central program. The same way, programs usually produce outputs (datasets or report) and the validation of a program is associated with the validation of its outputs.

The Life Cycle described in the previous sections can be extended through the use of Jobs to include as well as the validation status of input (and output) artefacts based on their eSignatures. Meaning that a Program cannot be considered “Validated” if e.g. an input dataset or macro does not have such status while program’s eSignatures analysis in isolation may say otherwise.

Analyzing jobs and their manifests using LSAF APIs would support dependencies analysis and provide this extra level of checks that is often assumed to be the responsibility of the programmer and validator to check when validating a program.

CONCLUSION

Simplicity of versioning and eSignature functionality (few clicks and limited options) that are the back-bone of implementing life-cycle management in LSAF makes it easy to adopt for users. Differences may occur between users in terms of when to turn on versioning on their code though but the requirement of turning on versioning when program is ready for validation is enforced by design as eSignature applies at a version level rather than object level.

The method described in this paper enables to avoid having a spreadsheet-based Programming Plan out-of-sync with changes applied in LSAF at a program level as well as tracking inadvertent code changes and unplanned programs that may require to be cleaned up before the study is called "Final" and locked. Besides up-to-date dashboards can be set-up and provide an accurate picture of study/project progress.

Using such method and collecting routinely information about the life-cycle of each program also enables to track programs that require longer/shorter life-cycles to be finalised where more/fewer iterations are required. Such information can inform Managers, Study and Project Leaders about potential bottle-necks, need for standardisation or training.

LSAF features enables to create simple life-cycle management tools focusing on single artefacts or more complex ones where dependencies can be captured through use of LSAF Jobs and evaluated in the computation of statuses. Different life-cycles can be used for e.g. exploratory work and more GCP-regulated clinical study work.

Other LSAF's capabilities such as Extended Attributes (to e.g. document Validation Level and Status) or Process Workflows can be considered for enhancing the concepts further.

REFERENCES

[1]: SAS Life Science Analytics Framework Macro API - http://support.sas.com/kb/60/264.html
[2]: SDD APIs: Extending the Capabilities of SAS Drug Development, Jean-Marc Ferran (Qualiance), 2014

ACKNOWLEDGMENTS

Thanks to Egbert Van Der Meulen for his support and feedback.

CONTACT INFORMATION

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

Jean-Marc Ferran
Qualiance ApS
Email: JMF@qualiance.dk
Web: http://www.qualiance.dk