

Tracking and Status Join Adolescence

Laura Phelan, Merck Serono International S.A, Geneva, Switzerland
Magnus Mengelbier, Merck Serono International S.A, Geneva, Switzerland

The methods for tracking and status reporting of clinical trial reporting are well versed topics within the clinical reporting process. The proliferation of manual tracking tools are common as well as difficult to manage as the number of projects and team members grow. Add the dimension of global project teams and the administration can become intensive and exceedingly difficult. We follow a typical tracking and status process from first steps to considerations for larger Life Sciences systems.

INTRODUCTION

Every reporting phase of a clinical trial uses a large number of programs, data sets, listings, tables and graphics to communicate the statistical results to support the study objectives. With so many program and output files to manage, there is a clear need to ensure the correct processes and conventions for generating output for the study reports have been adhered to, and that this requirement can be fulfilled in a user-friendly and efficient way.

The initial tracking and status reports were manual administrative tasks to coordinate resources and deliverables. Larger studies created a more demanding and extensive administration requirement.

We discuss a semi-automated tracking system, which has been developed to record all the relevant metadata related to the deliverables included in a report. The semi-automated tools are considered an aide to assist in the manual administrative tasks that remain.

We also consider how this tracking process and tools may be enhanced to use the meta-data provided by SAS Drug Development. The SAS Drug Development solution is delivered with tools of varying capability and complexity, which applied in the process may enable a more automated tracking and status.

The challenge is there to combine the original idea born of the current semi-automated system and the gamut of enhanced features available through the more mature and sophisticated capabilities of SAS Drug Development.

THE CURRENT TOOLS

This current tracking system is a two-step process: tracking and status reporting. The study specific tools should provide an overview in sufficient detail to document that programming efforts were completed successfully according to process requirements and applicable conventions.

Step one involves each program that creates an output while saving any associated and permanent files. Permanent files can include a wide range of outputs, for example a LOG, LST, CGM graphic, Excel table, PDF file to whatever the program outputs may be. Included at the end of each program is a call to the tracking macro, conveniently named %track(). This macro stores all of the information about the output and its associated program, and any other relevant outputs including the program log, in a user-named SAS data set stored in a central tracking location within the study area. If a program creates multiple outputs, the %track() macro is called once for each individual output.

The user-named SAS data set is a convention to avoid errors resulting from simultaneous access to a single tracking data set as SAS/Share is not licensed. Similar functionality compared to the study tracking and status tools could be provided as standard validated suite and a central database for all projects, but this would

considerably restrict the flexibility and adaptability to project and study specific nuances in the programming process.

The %track() macro works with Windows Application Programming Interface (API) functions to retrieve file information to be included in the tracking data set. The information includes file creation, access and last modification dates.

The second step involves combining the information from all user tracking data sets linked to the study, and a little consolidation and derivation to show, among other details, that:

- all programs have been correctly executed without errors;
- the study programs and deliverables have been executed and generated, respectively, in a standard chronological order;
- each output has a unique independent identifiable validation

A simple review of the documentation will also enable the programming team to assert that all planned deliverables are tracked, and managed according to the expected process and convention.

THE CURRENT TOOLS – PART ONE

The following example of code demonstrates how the %track() macro can be used when an output listing is created. Note the explicit save of the output and log with the DM, which indicates that the %track() macro can be used for both a batch execution and an interactive SAS session, should you be forced to do so.

```
%let _path_ = X:\sas\phuse;
%let file = example;

%inc "&_path_\macros\track.sas";

libname mylib "&_path_";

title "Demographic Characteristics";

proc sql flow=30;
    select * from mylib.dem;
quit;

DM output "output; FILE '&_path_&file..lst' REPLACE" wedit;

%track( validation_type=Double programming,
        output=&_path_&file..lst,
        author=Laura Phelan,
        purpose=Example of a Listing);

DM log "log; FILE '&_path_&file..log' REPLACE" wedit;
```

The program, log and LST output file path and names are defined with macro variables in the example, which is a common internal convention but not a requirement.

The %track() macro uses a set of parameters as configurable options. Among the parameters are:

Parameter	Description
validation_type	Method of output validation
output	Path and filename of the deliverable being tracked
pgm_file	Program file
log_file	Program execution log file
lst_file	LST output file

Both the primary and validation programs by design use the %track() macro, which facilitates associating a primary and QC output. The common reference is the output parameter as the current process stipulates that the output, and not the actual program, is the validated item.

There is one unique row per output maintained that reflects settings and other attributes when the output was last generated. The attributes captured include, but are not limited to, program name, titles, footnotes, file dates, operator identity and execution date.

```
proc print data=track.tr_lphelan;
run;
```

Tracking Dataset for LPhelan					
userid	purpose	output	title1	CREATEDS	run_datetime
LPHELAN	Example of a Listing	H:\SAS\PHUSE\EXAMPLE.LST	The SAS System	23JUL2007:17:31:16.00	23JUL07:17:31:19

THE CURRENT TOOLS – PART TWO

The second step is done with the status macro, again conveniently called %status(), which combines the user tracking data sets, mapped by output. Collating file names, dates and other useful information produces summaries, execution sequences and other indicators for each tracked output in the study. The status report has become a convenient tool that exposes possible critical deviations from our working process and conventions. This includes a minimum set by process and conventions.

- Primary output production date occurring after validation run date,
- Output date preceding latest program source code update, for both primary and QC programs independently,
- Errors and warnings in the log

More importantly, the status report also highlights any formal or contributing outputs in the study directory structure that has not clearly been tracked, but should be included.

Preliminary Results: Study 12345								
For PMF (Folder Contents): For Consistency Checking by the Lead Programmer and Signing by the Validation Programmer								
Part I: Primary vs Validation Program and Output Datetimes (Output Dates Include when Programmers Update their Track Datasets)								
----- Folder=DATADER\DATASETS -----								
File Name	Primary			Validation			Status	Signature
	Program /Author	Date /Time	User Producing Output	Program /Author	Date /Time	User Producing Output		
ADHAEM.SAS7BDAT	ADLABS.SAS JSMITH	11JUN2007 14:09:51	JSMITH	QC.LABS.SAS LPHELAN	11JUN2007 14:28:44	LPHELAN	OK	
ADHAEM_V2.SAS7BDAT	ADLABS_V2.SAS JSMITH	21JUN2007 10:22:52	JSMITH	QC.LABS_V2.SAS LPHELAN	21JUN2007 10:28:35	LPHELAN	A	
ADVITALS.SAS7BDAT	ADVITALS.SAS JSMITH	11JUN2007 12:31:42	LPHELAN	QC.VITALS.SAS JSMITH	11JUN2007 12:33:23	JSMITH	B	
CONQMED.SAS7BDAT	CONQMED.SAS JSMITH	11JUN2007 12:23:19	LPHELAN	QC.CONQMED.SAS JSMITH	11JUN2007 12:28:42	JSMITH	OK	

T:\TESTDRUG\12345\FINAL\DOCUMENTS\TRACKING\STATUS_V2.SAS 28JUN2007

Note: Run/Rerun implies the time the tracking dataset is updated (ie, when track macro is called, normally after output execution)

A=Prim. pgm changed Prim. not rerun/B=Prim. output updated but Val. not rerun/C=Prim. run before Val. Run/D=Val. pgm changed but Val. not rerun/E=Prim. pgm changed but output not updated/F=Prim. pgm changed but log not updated/G=No Prim. pgm/H=No Val. pgm/I=No Prim. log file/ J=No Val. log file/K=No Prim. Author/L=No Val. Author/OK=Consistent

EVOLUTION

The tracking and status will most commonly be included in changes and enhancements to the reporting process. This can include small steps as the tools and experience evolve, or a generation leap coinciding with the introduction of a new fully or semi-automated reporting system. The palpable difference between a task based manual reporting process and a more automated system will create its own business requirements.

The introduction of a system may also increase the expectations on how, what and when tracking and status information is made available. A system such as SAS Drug Development provides by default more advanced functions than your local network drive, but this does not necessarily mean that the business or process requires each and every one.

CASE OF SAS DRUG DEVELOPMENT

The SAS Drug Development (SDD) system stores details of the files, folders, data and other elements that can be used in tracking and status. Some information is easily available to each program through the Parameter Data Set, other items can be reached through the Command Facility Macro suite or the more extensive SDD Java Application Programming Interface (API), all topics which we will consider further.

This new information can be added to our concept of the tracking and status macros, providing a more complete and detailed picture. As the information available can be extensive, a review of applicable elements is prudent.

THE PARAMETER DATA SET

The Parameter Data Set in SDD is associated with each submission of SAS code from the system Process Editor or the Job Scheduler. The data set is created by the *system*, as if by magic, prior to the SAS code being executed, such that all information is available and accessible to the executing program code from the start.

The data set contains a host of information such as program location, name, output destinations, configuration parameters, etc., which can be useful for a track and status macro. The data set also includes all other attributes and configuration parameters captured for the execution of program code. This is helpful to track any and all user specified settings and configurations.

The data set name and location is specified by the SDD global macro variable SDDPARMS. The data set contains one row per parameter and attribute, while the key variables are Parameter ID (ID), Value Type (VALTYPE), Value (VALUE) and Update (UPDATE). The Update variable is the indicator if the user has overridden or changed the default value in any way. Key default parameters and attributes can be date (execution date), user (SDD operator), filename (program file name), and version (program file version).

The data set also applies if only a selection of program code is submitted through the Process Editor, which can be identified using the program name, as this will follow conventions of a SDD temporary file name, for example _itt7_.sas. Consequently, if the entire program code is selected and submitted using the Process Editor, the submitted code is still identified as a selection, which may provide an incorrect or correct indication depending on your organization conventions and culture.

COMMAND FACILITY MACROS

The Command Facility macros are SAS macros that provide a simple route to query information on files, folders, data and other elements, e.g. SDD objects, stored within SDD using SAS macro syntax.

An over simplified use of the Command Facility macros will retrieve all properties and version information for the DATA data set, returned as the two data sets PROPERTIES and VERSIONS, respectively.

```

%swd_start(url = , user =, password= );

%swd_getallobjectproperties("/foo/bar/data.sas7bdat", data = work.properties );
%swd_versions( "/foo/bar/data.sas7bdat", data = work.versions );

%swd_stop;

```

The %swd_getallobjectproperties() macro will provide, among all properties, object global id, display name, content type, creation date, last modification date and other pertinent information.

The %swd_versions() macro will return version information as one record per version. Interestingly, the comment potentially provided when creating a new version is not available, which may hinder usability. For the extensive information on versions, the developer or end-user is deferred to the more complete, extensive and complex Java API.

The Command Facility macros can perform simple administrative tasks as well, such as check-in and check-out. Extending the above example, checking out, updating and checking in an output file, if the program is signed can be accomplished in a fairly simple manner.

```

%macro save( file = );

%swd_start(url = , user =, password= );

%swd_getallobjectproperties("/foo/bar/data.sas7bdat", data = work.properties );
%swd_versions( "/foo/bar/data.sas7bdat", data = work.versions );

data _null_;
  set work.properties ;
  where ( parameter = "isSigned" );
  call symput('signed_program', value);
run;

%if ( &signed_program eq 1 ) %then %do;

  %* -- code to save file ;

%end;

%swd_stop;

%mend;

```

Note that by default SDD does not hinder a new version to be created when a file is checked in unless write access is restricted or managed. In order to manage that restriction, a weak form of read/write access can be implemented using a custom output file property and the Command Facility macro %swd_getallobjectproperties() and %swd_setobjectproperty(). This will enable control of write access to output files through program code, while the user retains all regular permissions through the user interface.

The Command Facility Macros does not provide access to all possible information elements and controls, but the information available can be ample to manage a simple process flow.

JAVA APPLICATION PROGRAMMING INTERFACE

The Java Application Programming Interface (API) is a developer's tool set with greater capability than the Command Facility macros, specifically created for accessing and interacting with SDD from one or more external system. The Java API functions can also be used from within SDD, if installed, very similar to the Command

Facility macros with one caveat. The SAS macros that use the Java API do not exist as a standard and any macros developed to use the Java API are custom utilities specific to your system and organization. In addition, the Java API cannot out-of-the-box be used directly from within SAS code, but will also require custom Java code written to manage access, exceptions, errors and other custom process flows.

In general, the methods of using the Java API from a SAS macro are the similar to the Command Facility and both will most likely use the DATA step Java object interface. Command Facility macros are available examples on how to connect SAS program and Java code in SDD, and SAS in general.

ENHANCING THE FIRST TOOLS

Extending the first tools may require a substantial investment, but any return can be argued favorably. SDD maintains more information about the program files, logs, outputs, data sets, etc. and program execution as a side effect of the system design and functionality. This information can be extremely valuable in tracking progress and documenting the programming effort.

The tracking meta-data available is of consequence more extensive, both as obtainable information and an additional requirement due to the high dependency on SDD configuration parameters. The possibility to identify conditions pertaining to the execution of program code, e.g. selection of code, signed programs, etc., will enable a more formal control of how and when outputs are created.

The first step is to extend tracking to capture the configurations required for each program code execution in SDD. Add a more formal programmatic control of individual outputs and the tracking information can in more detail document the programming process and outputs. This is easily achieved by using the SDD Parameter Data Set, creative parsing of the program log and general file properties, as seen with the %swd_getallobjectproperties() macro.

The architecture and system design of SDD also imposes further restrictions on the tracking data sets previously discussed. The current implementation, e.g. not SDD, uses a user-specific data set with no concurrent access. SDD adds complexity by its system design, as there is a distinct possibility for concurrently executing programs to overwrite the others additions. Creating temporary tracking data sets unique to each output with a scheduled reoccurring consolidation can circumvent this.

The use of an external tracking database not only circumvents the concurrent data set access limitation, but also provides additional capacity and capability as well. The next evolutionary step can add functions pertaining to planning and output standards in an effort to enhance the programming environment further.

The track and status tools will gain a more central monitoring role, extended from the initial documenter. The new generation of the %track() macro will enable the collection of tracking information for each program execution, e.g. attempt to generate an output. The extensions to the %track() and %status() macros can also include monitoring rules for standard and invalid configurations, to enable resolution of any issues and conflicts as early as possible in the programming process and prior to final delivery.

This approach is readily accomplished with the use of the SDD Parameter Data Set, although some more detail control may require post-processing the log to assert correct execution. A stronger option will be to extend the %track() macro context to enforce each program execution such that a valid output deliverable is only generated if the program execution complies with a set of predefined formal conditions, e.g. the previously mentioned %save() macro.

CONCLUSION

The track and status tools are central in documenting the programming involved for Statistical and Integrated Clinical Reports. The tools have evolved from a strict manual administrative process to a semi-automated approach.

The current tools primary purpose as an aide in the manual administrative process works efficiently in small to medium studies, but do impose process performance and logistical issues for larger volumes of programming effort.

The added features and new meta data introduced with the SAS Drug Development system does add possibilities and an increase in expectations. As with any environment, the system introduces its own set of issues, but a considered tracking and status process will provide additional security.

The investment in a well-formed track and status tool set as part of SAS Drug Development is arguably beneficial to provide a clear monitoring and overview of the programming status, workflow and deliverables.

The SDD system does have basic functionality in the Parameter Data Set and the by default installed Command Facility macros. Further customization with the Java API may be required to fully support the business requirements for formal tracking and status.

The goal is not to be the Big Brother of study reporting, but provide simple tools for the clinical reporting process and administration requirements.

REFERENCES

- SAS 8.2 User Documentation
- SAS 9.1.3 User Documentation
- SAS Drug Development 3.3 Command Facility User Guide
- SAS Drug Development 3.3 Java Application Programming Interface