Paper SM08

Automation of Process Flow Instances in LSAF

Caro Sluijter, OCS Life Sciences, ‘s-Hertogenbosch, the Netherlands

ABSTRACT

One of the notable features of SAS® Life Science Analytics Framework (LSAF) is the workflow capability. This functionality allows you to automate processes by using process flows; to allocate tasks and to keep track of the progress of tasks and activities. A process flow is particularly useful when a process consists of several standardised tasks that need to be executed routinely and chronologically within or across studies. Process flows can thus eliminate repetitive tasks.

However, setting up process-flow instances can in itself be a repetitive task, because for all process flow elements the details need to be selected manually. As process flows are meant to reduce or eliminate repetitiveness, we also wanted to remove repetitiveness in creating the Process Flow Instance.

In this paper we will demonstrate how LSAF API macros can be used to set up a Process Flow Instance in an automated fashion requiring as little as one or two parameters.

LSAF, PROCESS FLOWS AND PROCESS FLOWS IN LSAF

It is important that you understand how process flows work in general and in LSAF before we start with elaborating on the solution to eliminate repetitiveness of Process Flow Instance creation.

The SAS Life Science Analytics Framework (LSAF) is a single solution, integrated system. It provides a centralised, standardised data repository and SAS programming and execution environment in a single solution. One of the key features of LSAF is the workflow capability, the capability to create process flows.

A process flow provides a visual overview or workflow diagram of all the tasks and relationships involved in a process. Using process flows reduces the manual involvement and it automates, streamlines and tracks processes which benefits the management of your clinical research processes. A process flow is particularly useful when a process consists of several standardised tasks that need to be executed repetitively within or across studies.

Workflow capability in LSAF allows you to create process flows to allocate tasks and to keep track of the progress of task and activities. A process flow can contain several different elements, such as an element to notify someone or a group for either a message or a task, an element to activate or run a program or a job, or an element to pick up a signal, for instance that data is uploaded on a specific location.

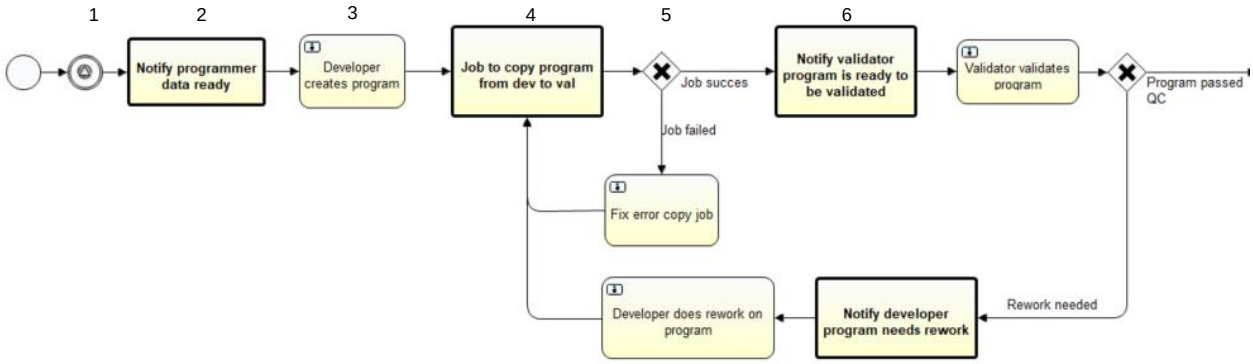

Figure 1: Example process flow: Lifecycle Program

Figure 1 shows an example of a part of a process flow that portrays the lifecycle of a program. The process flow starts with waiting for a signal that data is uploaded (1). When that occurs, a message will be sent (2) to the programmer group that data is ready to be processed. Next, a user task is defined (3) for a developer to create a program. Once this task is completed, a job is activated (4) to copy the developed program from the development environment to the validation environment. A check occurs whether the jobs runs correctly (5) and if this is the case, the validator group is notified (6) that a program is ready to be validated.

It is important to be familiar with the terms Process Flow *Definition* and Process Flow *Instance*. A Process Flow Definition is an empty shell of a process flow that is not yet linked to any files or users in the system. It stems from a BPMN (Business Process Model and Notation) file that is created outside LSAF, for example using Eclipse. From the Process Flow Definitions one or more Process Flow Instances are created. Process Flow Instances are the process flows as they are used by users within the system.

Using process flows in LSAF can be broken down into 3 main steps.

- Step 1: Deploying the Process Flow Definition onto LSAF to act as a shell;
- Step 2: Using that Process Flow Definition to create a study specific Process Flow Instance.

Both steps can be broken down into smaller manual steps. Step 1 has to be executed only once per Process Flow Definition.

- Step 1.a: Creating the process flow BPMN file outside LSAF;
- Step 1.b: Uploading the BPMN file containing the process flow into LSAF;
- Step 1.c: Deploying and activating the process flow in LSAF.

Step 2 has to be executed every time you want to create a new Process Flow Instance. This step can be broken up like this:

- Step 2.a: Creating a Process Flow Instance;
- Step 2.b: Populating the Process Flow Instance elements, by assigning the correct path, users, jobs etcetera;
- Step 2.c: Activating the Process Flow Instance.

Especially step 2.b is repetitive and cumbersome. And as one of the goals of a process flow is to eliminate the repetitiveness in our day-to-day work, we also wanted to come up with a solution to eliminate the repetitiveness in the creation of Process Flow Instances. The next sections of this paper will present a solution to automate the creation of Process Flow Instances.

REMOVING REPETITIVENESS BY USING API MACROS

Every manual task in LSAF can be executed using LSAF specific Application Programming Interface (API) macros. This means that we can create an LSAF program that executes API macros to populate the Process Flow Instance elements. There are two categories of APIs that are related to the process flows: APIs regarding the process flow itself and APIs regarding the process flow setup.

ProcessFlow	lsaf_activateprocessflow
ProcessFlow	lsaf_createprocessflow
ProcessFlow	lsaf_createprocessflowmanifest
ProcessFlow	lsaf_deleteprocessflow
ProcessFlow	lsaf_getmyprocessflows
ProcessFlow	lsaf_getprocessflowdata
ProcessFlow	lsaf_getprocessflowproperties
ProcessFlow	lsaf_getprocessflows
ProcessFlow	lsaf_processflowexists
ProcessFlow	lsaf_suspendprocessflow
ProcessFlow	lsaf_updateprocessflowdata
ProcessFlow	lsaf_updateprocessflowproperties

ProcessFlowSetup	lsaf_getpfsetupelements
ProcessFlowSetup	lsaf_getpfsetupjobinfo
ProcessFlowSetup	lsaf_getpfsetupjobparameters
ProcessFlowSetup	lsaf_getpfsetupnotifinfo
ProcessFlowSetup	lsaf_getpfsetupnotifrecips
ProcessFlowSetup	lsaf_getpfsetupsignallocs
ProcessFlowSetup	lsaf_getpfsetuptimers
ProcessFlowSetup	lsaf_getpfsetupusercandidates
ProcessFlowSetup	lsaf_getpfsetupuserinfo
ProcessFlowSetup	lsaf_updatepfsetupJobInfo
ProcessFlowSetup	lsaf_updatepfsetupjobparameters
ProcessFlowSetup	lsaf_updatepfsetupnotifinfo
ProcessFlowSetup	lsaf_updatepfsetupnotifrecips
ProcessFlowSetup	lsaf_updatepfsetupsignallocs
ProcessFlowSetup	lsaf_updatepfsetuptimers
ProcessFlowSetup	lsaf_updatepfsetupusercandidates
ProcessFlowSetup	lsaf_updatepfsetupuserinfo

Figure 2: LSAF API Macros related to Process Flows

The solution comes in the form of a SAS program in LSAF. This SAS program uses the API macros to create new Process Flow Instances from existing Process Flow Definitions. This SAS program is turned into a job with a minimal number of parameters required to create the Process Flow Instance.

GLOBAL VARIABLES

We start our program with a section of global macro variables to make our life easier later in the program.
First we created some macro variables that split up the specific study macro path within LSAF into four different values. Second, we created macro variables to assign a date and time, to later use in the name of the Process Flow Instance. Third, we created macro variables to assign the Process Flow Instance name, specify groups that we assign later on to specific user tasks or notifications and assign the location of the job in the repository needed for this process flow. The process flow name consists of multiple elements to be sure that a Process Flow Instance has a unique name, otherwise SAS will produce an error that you cannot create a Process Flow Instance with an already existing name.

```
/*************************************************************************
  Create global variable(s)
*************************************************************************/
/*Split up macro path into the different sections*/
%let path_real  = %sysfunc(tranwrd(&path,&_sasws_,""));
%let compound   = %scan(&path_real,2,"/");
%let indication = %scan(&path_real,3,"/");
%let study      = %scan(&path_real,4,"/");

%let date       = %sysfunc(inputn(&sysdate9., date9.),yymmdd10.);
%let time       = %sysfunc(tranwrd(%sysfunc(time(),time6.0),:,_)); /*Time, ':' not allowed in sas name*/

/*Process flow instance name*/
%let pfname = pf_lifecyle_&compound_&indication_&study_&date_&time

/*Specify group that will be assigned to tasks and/or get notifications*/
%let groupid1 = dev_&study.; /*Developers of the study*/
%let groupid2 = val_&study.; /*Validators of the study*/

/*Location of the copy job*/
%let jobpath1 = /Hands-On_Demos/OCS/OCS LS Process flows/OCS LS Process Flows Lifecycle/dev/programs/copytool_stop2_execute_copy_
```

CREATE PROCESS FLOW INSTANCE

In order to populate a Process Flow Instance, you first need to create an empty Process Flow Instance where the name and context of the Process Flow Instance are defined. This can be done with the API %lsaf_createprocessflow:

```
%lsaf_createprocessflow(lsaf_path           = &path_real.
                       ,lsaf_processflow    = &pfname.
                       ,lsaf_processflowdef = DevelopmentProgramLifecycle
                       );
```

The parameter *lsaf_path* assigns the context (i.e. the folder) in which the Process Flow Instance should operate. The parameter of *lsaf_processflow* is used to name the Process Flow Instance. And the parameter *lsaf_processflowdef* is used to link the correct Process Flow Definition to the Process Flow Instance.

POPULATE THE PROCESS FLOW INSTANCE

There is a number of different elements a process flow can have. Each element requires its own configuration. All elements created for our example process flow have been identified with an *elementid*, this *elementid* is assigned to the element by the developer of the process flow BPMN file.

In the following sections the configuration of each process flow element is explained.

USERTASK

Each UserTask can be assigned to either a group of users or a single user. To get this information into the process flow we need to populate the information in the dataset WORK.GETPFSETUPUSERCANDIDATES or WORK.LSAFGETPFSETUPUSERINFO, respectively.

```
/*Create dataset with attributes for UserTask Candidates*/
data work.getpfsetupusercandidates;
  length processflowpath
         processflowname
         grpSrcCtxt
         elementid
         type
         principalid  $100;

  /*Determine general attributes for UserTasks*/
  processflowpath = "&path_real.";
  processflowname = "&pfname.";
  grpSrcCtxt      = "&path_real.";

  /*UserTask = Create Program*/
  elementid       = "createprogram";
  type            = "GROUP";
  principalid     = "&groupid1.";
  output;

  /*UserTask = Execute Rework*/
  elementid       = "executerework";
  type            = "GROUP";
  principalid     = "&groupid1.";
  output;

  /*UserTask = Validate program*/
  elementid       = "validateprogram";
  type            = "GROUP";
  principalid     = "&groupid2.";
  output;

run;
```

```
/*Create dataset with attributes UserTask single user*/
data work.lsafgetpfsetupuserinfo;
  length processflowpath
         processflowname
         elementid
         name
         value $100;

  /*Determine general attributes of User information*/
  processflowpath = "&path_real.";
  processflowname = "&pfname.";

  /*Assign person to UserTask*/
  elementID = "fixerrorcopyjob1";
  name      = "assigneeid";
  value     = "ocs_caslui";
  output;
run;
```

There are a number of items to configure for each UserTask when assigning a task to a group:

- *processflowpath*, *processflowname*, and *grpSrcCtxt* should be the same for each UserTask. As these elements describe the Process Flow Instance that will be updated.
- *elementid* is the name of the element you add information for. You predefine your *elementid* values in the BPMN file.
- *type* is the value on which type of group of users are assigned to the UserTask such as GROUP or USER;
- *principalid* contains a value that corresponds to the name of the candidate. The value is either a group name or a user ID. In our example for instance all validators or all programmers in a study to assign the task to.

When assigning a task to a user, the configuration is slightly different:

- processflowpath and processflowname are the same;
- *elementid* is the name of the element;
- *name* to identify the type of user information you want to add; and
- *value* which, in our example, is the username of the user you want to assign the UserTask to.

After the datasets are updated with the information, two API macros are used to update the Process Flow Instance with our configuration:

```
/*Update datasets with datastep to assign groups of candidates to
  the UserTasks*/
%lsaf_updatepfsetupusercandidates(sas_dsname = work.getpfsetupusercandidates);

%lsaf_updatepfsetupuserinfo(sas_dsname=work.lsafgetpfsetupuserinfo);
```

JOB INFORMATION AND JOB PARAMETER INFORMATION

Just as we populated the UserTask elements, we also need to populate the job elements. In our example the process flow contains one job which needs one input parameter. Population of a job element consists of two parts: assigning the job information element (in the dataset WORK.GETPFSETUPJOBINFO) and the job input parameters (in the dataset WORK.GETPFSETUPJOBPARAMETERS):

```
data work.getpfsetupjobinfo;
  length processflowpath
         processflowname
         elementid
         name $100
         value $200 ;

  /*Determin general attributes of jobs*/
  processflowpath = "&path_real.";
  processflowname = "&pfname.";

  /*Set location copydevtoval job*/
  name            = "jobPath" ;
  value           = "&jobpath1.";
  elementID       = "copydevtoval";
  output;

  /*Set location copyvaltoprod*/
  name            = "jobPath" ;
  value           = "&jobpath1.";
  elementID       = "copyvaltoprod";
  output;

run;

/*Update dataset with new information on job location*/
%lsaf_updatepfsetupjobinfo(sas_dsname = work.getpfsetupjobinfo);
```

The items to configure the correct job to the element are:

- *processflowpath* and *processflowname* are the same;
- *elementid* is the name of the element;
- *name* to identify the type of job information you want to add;
- *value*, in our example, as the path to the job in the repository.

The Process Flow Instance is updated with the API macro %lsaf_updatesetupjobinfo.

Second, we assign the job parameter values per job with the following code:

```
data work.getpfsetupjobparameters;
  length processflowpath
         processflowname
         elementid
         jobpath
         name
         type
         value $100;

  /*Determin general attributes of job parameters*/
  processflowpath   = "&path_real.";
  processflowname   = "&pfname.";
  fileVersion       = "";

  /*Add parameter for job copydevtoval*/
  elementid         = "copydevtoval";
  jobpath           = "&jobpath1.";
  name              = "location_parameters";
  type              = "Folder";
  value             = "&path_real./dev/programs";
  includesubfolders = 0;
  output;

  /*Add parameter for job copyvaltoprod*/
  elementid         = "copyvaltoprod";
  jobpath           = "&jobpath1.";
  name              = "location_parameters";
  type              = "Folder";
  value             = "&path_real./val/programs";
  includesubfolders = 0;
  output;

run;

/*Update dataset with new information on job input parameters*/
%lsaf_updatepfsetupjobparameters(sas_dsname = work.getpfsetupjobparameters);
```

To configure the job parameters per element, we use some of the same variables but with different values:

- *processflowpath* and *processflowname* are the same;
- *elementid* is the name of the element;
- *name* identifies the job parameter;
- *value*, in our example, contains the value of the job parameter
- *type* the type of the override property, such as FOLDER, FILE, CHARACTER etcetera;
- *jobpath* corresponds to the location of the job in the repository.

The Process Flow Instance is updated with the API macro %lsaf_updatepfsetupjobparameters:

SIGNAL CATCHING EVENT

In this process flow a signal catching event is included: This element will look in a certain location for a specific signal, such as new data being uploaded. To activate this element, we need to assign the location where the signal can be found in the WORK.LSAFGETPFSETUPSIGNALLOCS dataset:

```
data work.lsafgetpfsetupsignallocs;
  length processflowpath
         processflowname
         elementid
         location $100;

  processflowpath = "&path_real.";
  processflowname = "&pfname.";
  elementid       = "catchdatauploaded";
  location        = "&path_real./dev/input";
  output;
run;

%lsaf_updatepfsetupsignallocs(sas_dsname=work.lsafgetpfsetupsignallocs);
```

The only new variable in this dataset is *location*. This variable contains the path to the location you want a signal to be captured. The Process Flow Instance is updated with the API macro %lsaf_updatepfsetupsignallocs.

NOTIFICATION MESSAGES

In our example users are notified when data is ready to be programmed, a program is ready for validation and when rework is needed. Within the process flow you can customize these messages and determine who should receive a specific message. To assign the notification message the dataset WORK.LSAFPFSETUPNOTIFINFO is updated with the following code:

```
data work.lsafgetpfsetupnotifinfo;
  length processflowpath
         processflowname
         elementid
         name
         value $100 ;

  /*Determin general attributes of message*/
  processflowpath = "&path_real.";
  processflowname = "&pfname.";


  /*Set message to be send out if new files are being uploaded*/
  elementID       = "notifydataready";
  name            = "message";
  value           = "Study: &study; Data is uploaded and ready for progra
  output;

  elementID       = "notifyreadyforvalidation";
  name            = "message";
  value           = "Study: &study; Program ready for validation.";
  output;

  elementID       = "notifyprogaminproduction";
  name            = "message";
  value           = "Study: &study; Program moved to production";
  output;

run;

%lsaf_updatepfsetupnotifinfo(sas_dsname = work.lsafgetpfsetupnotifinfo);
```

The items to configure to assign a message to an element are;

- processflowpath and processflowname are the same;
- *elementid* is the name of the element;
- *value* contains the notification message.

The Process Flow Instance is updated with the API: %lsaf_updatepfsetupnotifinfo

To assign the recipients of the message we update the dataset WORK.LSAFGETPFSETUPNOTIFRECIPS:

```
data work.lsafgetpfsetupnoticyrecips;
  length processflowpath
         processflowname
         grpSrcCtxt
         elementid
         type
         principalid $100;

  /*Determin general attributes of Message Recipients*/
  processflowpath = "&path_real.";
  processflowname = "&pfname.";
  grpSrcCtxt      = "&path_real.";

  /*Set-up a group to receive the message*/
  elementid = "notifydataready";
  type      = "GROUP";
  principalid = "&groupid1";
  output;

  elementid = "notifyreadyforvalidation";
  type      = "GROUP";
  principalid = "&groupid2";
  output;

  /*Set-up a person to receive the message*/
  elementid = "notifyprograminproduction";
  type      = "USER";
  principalid = "ocs_caslui";
  output;

run;

%lsaf_updatepfsetupnotifrecips(sas_dsname=work.lsafgetpfsetupnoticyrecips);
```

The items to configure to assign a group or single recipient to the notification element are;

- processflowpath and processflowname are the same;
- *grpSrcCtxt* is the same for all notification elements and equal to the value to assign a user task to a group;
- *elementid* is the name of the element;
- *type* describes the type or recipient, such as GROUP or USER;
- *principalid* is the recipient of the message, which can be a User or a Group (as assigned at the start of the program)

The Process Flow Element is updated with the API macro %lsaf_updatepfsetupnotifrecips.

ACTIVATE THE PROCESS FLOW INSTANCE

This is where the magic happens. Once all configuration is done – and that is a one-time process for each Process Flow Definition – the last step to take is to activate the Process Flow Instance. This is achieved with the following API macro:

```
%lsaf_activateprocessflow(lsaf_path = &path_real.
                         ,lsaf_processflow = &pfname.
                         );
```

Once activated, this (mostly automatically created) process flow is indistinguishable from any other Process Flow Instance. Each time a new instance of this Process Flow Definition is required, all that is needed is to run this SAS job, define the location where it should run, and that's all!

CONCLUSION

By using this method, we created a solution to create, populate and activate a Process Flow Instance only needing one parameter consisting of the path to the study folder in the repository. This provides a considerable reduction of effort required to initiate new Process Flow Instances.

These are a few tips for when you want to automate the creation of your Process Flow Instances:

- make use of macro variables for variables that are needed in multiple datasets: ProcessFlowPath, ProcessFlowName and GroupID.
- use macro variables for paths that might change when a program is moved in the repository and determine these paths at the top of the program: job paths, output paths, input paths.

Programmers generally don't like performing repetitive tasks. After all, that's what programming is for. Using process flows in LSAF is an excellent way to reduce repetition in the daily work of programmers and data managers.

By using the approach described in this paper we eliminated even more repetitiveness by automating the creation of Process Flow Instances. By eliminating these tasks, we can move our focus away from the process and to the content and results of your studies.

REFERENCES

- https://ocs-lifesciences.com/what-is-lsaf/
- SAS Life Science Analytics Framework Macro API - https://support.sas.com/kb/60/264.html
- Manage TLFs Development in Life Science Analytics Framework 5.3 Using SAS and R code, https://www.pharmasug.org/proceedings/2021/SI/PharmaSUG-2021-SI-213.pdf

RECOMMENDED READING

Paper DH05: Utilising LSAF for data management at a Belgian biotech, by Louella Schoemacher.

CONTACT INFORMATION

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

Caro Sluijter
OCS Life Sciences
Email: sasquestions@ocs-consulting.com
Web: https://ocs-lifesciences.com/
Web: https://www.linkedin.com/in/caro-sluijter/

Brand and product names are trademarks of their respective companies.