Paper AD09

Breaking Boundaries: Working Across Different Workspaces in LSAF

Kai Wanke, OCS Life Sciences, ‘s Hertogenbosch, The Netherlands

ABSTRACT

In LSAF it is common practice to create a parent job that runs multiple child jobs consecutively. The parent and child jobs will create and act in their own, separate workspaces. While often it is sufficient to act within these boundaries, there can be cases when it is desirable that one job accesses the workspace of another one. E.g. when a child job needs to access a status report continuously updated by its parent. This paper presents an approach that allows a child job access to the parent jobs’ workspace by overwriting existing parameters in the child job. A job parameter will be modified on-the-go to contain the parent jobs’ transient workspace location which can then be used during the job execution. This approach can be further modified to create shared workspaces to facilitate the direct interaction and cooperation between parent and child jobs.

INTRODUCTION

The Life Science Analytics Framework (LSAF) by SAS© is a tool aimed at the optimization and automation of the processing of (clinical trial) data. Its functionalities include data storage and version control, the application of CDISC standards, the generation of automatic process flows and the execution of SAS programs.
Typically, the user will make use of two different types of file directories in LSAF:

The first is the repository, which primarily acts as a permanent file storage location. Files stored here cannot be accessed or modified by regular programs, protecting them from accidental changes. LSAF allows version controlling of files so that the changes that were made are being recorded and can be traced back. Moreover, users with administrative rights can limit the access of other users to files in the repository and determine which kind of changes are permissible, making the repository a very robust storage area.

The second functional part of LSAF is the workspace, a more dynamic environment created for every user individually. The workspace is a ‘sandbox’ environment in which user can write and execute SAS programs and modify files that were taken from the repository. Files can be carried over from the workspace to the repository to either create new permanent files or update existing versions.

Data processing in the repository is usually performed via LSAF job files. Jobs are XML files that describe various parameters used for the execution of SAS-programs to perform different functions. Jobs describe which programs are going to be run, which files in the repository may be altered and which folder(s) in the repository can contain job output. In addition, they permit the use of parameters that are called by its associated SAS programs to dynamically alter its functions[1].

The processes described in a job file do not occur in the repository. Instead, the job creates a temporary storage location called the transient workspace. This type of workspace contains copies of files from the repository that were listed as the job input. Then, it executes its SAS programs based on the submitted parameters to perform its designated tasks. In the absence of SAS errors, a completed job will copy its modified output back into the repository, either adding new files or overwriting old ones in the process. The transient workspace and its files are typically not directly accessible to the user and will be deleted at the end of the successful job execution.

Working with clinical trial data often requires several different processing steps and LSAF allows the combination of multiple jobs in a process flow to handle these situations. Usually, every job performs a defined task with specific programs and files from the repository to shape the data into its desired form. Typically, the jobs are being executed in consecutive sequence where every job is going to create its own transient workspace, runs its program and places the results back into the repository from which the following job will take it over into its own transient workspace. This sequential execution of the jobs prevents jobs from using more than one transient workspace at a time.

[1]Job parameters are functionally identical to global SAS macro variables.

Some process flows use one specific ('parent') job to control and execute the various ('child') jobs which makes it possible to coordinate their execution and to automate the process flow. An interesting side effect of this process flow is that the transient workspace of the parent job is going to be maintained during the execution of the child jobs. This transient workspace will only be removed when the entire process flow is finished and the parent job ends. This means that in this scenario, two transient workspaces exist at the same time but in distinct, non-overlapping locations. Moreover, the two transient workspaces are created at different time points with different parameters, reflected in the respective content of the repository that has been moved into them.

The different transient workspaces are usually separated completely and it is not possible for e.g. one child job to access the transient workspace of the parent job. In the vast majority of cases, using another job's transient workspace is not necessary. However, we found that there are specific conditions where it can be beneficial to access certain files generated by the parent job which have not been moved to the repository prior to the execution of a child job. In this paper, a method is described to allow a child job to access files from its parent jobs' transient workspace during the run of a complex process flow and present scenarios where this helps to further automate this process flow.

PROBLEM DESCRIPTION

The approach presented in this paper was developed during the creation of an automatic data processing framework for one of our clients. This framework has the general purpose of using a parent job to execute a range of child jobs with different individual processing functions. The aim of these jobs was to automatically obtain, process, validate and convert clinical trial data obtained by different vendors into standardized, CDISC compliant datasets.

This process is monitored by the creation of different report SAS-datasets: First, there is the dataflow report, a general report created by the parent job that summarizes the run time and status of individual child jobs. Second, every child job creates a detailed report that describes which files it used, what it did and if it encountered any issues during the process. The framework contains a notification job to collect these datasets and to combine them in one final report dataset which is then used to write an email to be send to the user.

This notification job is run as the final child job in the process flow, which creates an complex situation: The detailed report that was created by preceding child jobs have been copied back into the repository prior to the execution of the notification job, meaning that it was available to be copied into its transient workspace. The dataflow report, however, has not been moved back into the repository since the parent job will continue to run until the final child job has been completed. Because of this, the notification job cannot access the report and is unable to finalize the message for the user (Figure 1):

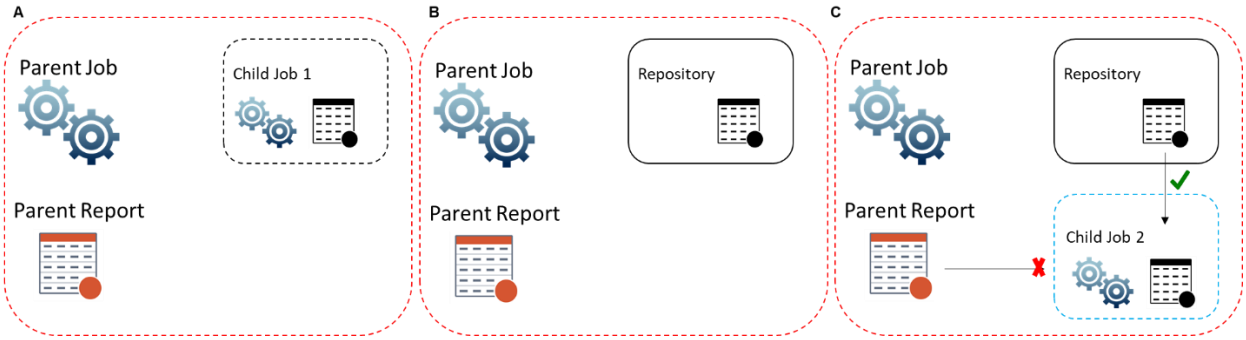

Figure 1: Schematic representation of the interaction between transient workspaces and repository and a complex process flow involving one parent and two child jobs.

Figure 1A: At the start, the parent job is going to create its own transient workspace (red square) from which is initializes the first child job. This job will create its own, separate transient workspace (black square) in which it starts to process data obtained from the repository. The parent creates a report dataset in its own transient workspace which is located in a spatially distinct location than child job 1.

Figure 1B: At the end of its execution, child job B moves its processed data back into the repository. The execution of the parent job continues so that its transient workspace still exists and its report is not moved into the repository.

Figure 1C: Child job 2 is initialized and creates its transient workspace (blue square). The child job is able to access the previously updated repository meaning that it can make use of the dataset processed by child job 1. The parent report, however, is still located in the transient workspace of the parent job and cannot be reached by child job 2.

In summary, there is no straightforward way to either 'push' a dataset from the transient workspace into the repository or to 'pull' it from the repository to the transient workspace while the dataflow is still running.

This problem could be avoided by simply waiting until the parent job finishes its execution and to start the notification job as an independent process. However, there are several reasons why this was not the preferred option in this project:
1) Having two separate jobs would defeat the original idea of having one singular, automatic process flow that performs every action without additional input.

2) Scheduling the jobs would be possible but requires the user to time their execution. The execution time of the initial process flow may vary from time to time (e.g. because the amount of delivered trial data can be different) so that the timing of the two jobs can be difficult to set. The final report may be sent too early (before every process is completed) or too late for a user to react.

3) The framework was intended to be as easy to use as possible and the aforementioned issues would reduce its user-friendliness. To overcome this, we aimed to find a method that allows a child job to access the transient workspace of its parent job during its execution which will be presented in the following.

OVERVIEW OF THE APPROACH

LSAF automatically creates various macro variables to point to different locations relative to a running job (Figure 2). The most important one for the presented method is "&_SASWS_" which refers to the transient workspace of the currently running job. A limitation of this macro variable is, that this default behaviour cannot be changed and that no other transient workspace can be selected.
To overcome this limitation, this approach will use an LSAF API is used to modify an empty child job parameter to add the value of &_SASWS_ as resolved by the parent job. By doing this, a new macro variable is created in the child job which inherits the location of the parent job transient workspace. Calling this macro variable will then make it possible to override the default settings in the child job in order to access the parent jobs' transient workspace.

```
/* SAS Program location */
&_SASFILEPATH_;

/SAS/0000/Files/LSAF_Test/Example_Job/Programs/child_job.sas
/* User transient workspace */
&_SASUSRWS_;

/lsafshared/SASWorkspaces/.transient/child_job.job-840cc342-4fda-4dc7-8410-b03904f44582/Users/KW1
/* Job transient workspace */
&_SASWS_;

/lsafshared/SASWorkspaces/.transient/child_job.job-840cc342-4fda-4dc7-8410-b03904f44582
```

Figure 2: Common automatic macro variables used in LSAF to refer to the relative location of a running job or (user) workspace.

INHERIT THE PARENT JOB TRANSIENT WORKSPACE LOCATION IN THE CHILD JOB

CREATE A JOB PARAMETER CONTAINING THE TRANSIENT WORKSPACE LOCATION OF THE PARENT JOB

To contain the location of the parent jobs' transient workspace, an empty parameter will be created in the child job file (Figure 3). LSAF APIs can modify existing parameters but cannot create new ones so that an empty template is necessary. In this case, the job parameter will be called "_SASPJWS_" and created as a character parameter with no default value. The parent job file contains the parameter "jobpath" which is used to define the repository locations of the jobs used here. This example also makes use of a parameter called "Output", present in both jobs. This parameter points to the same location in the LSAF repository and will be used as output location of the modified report dataset.

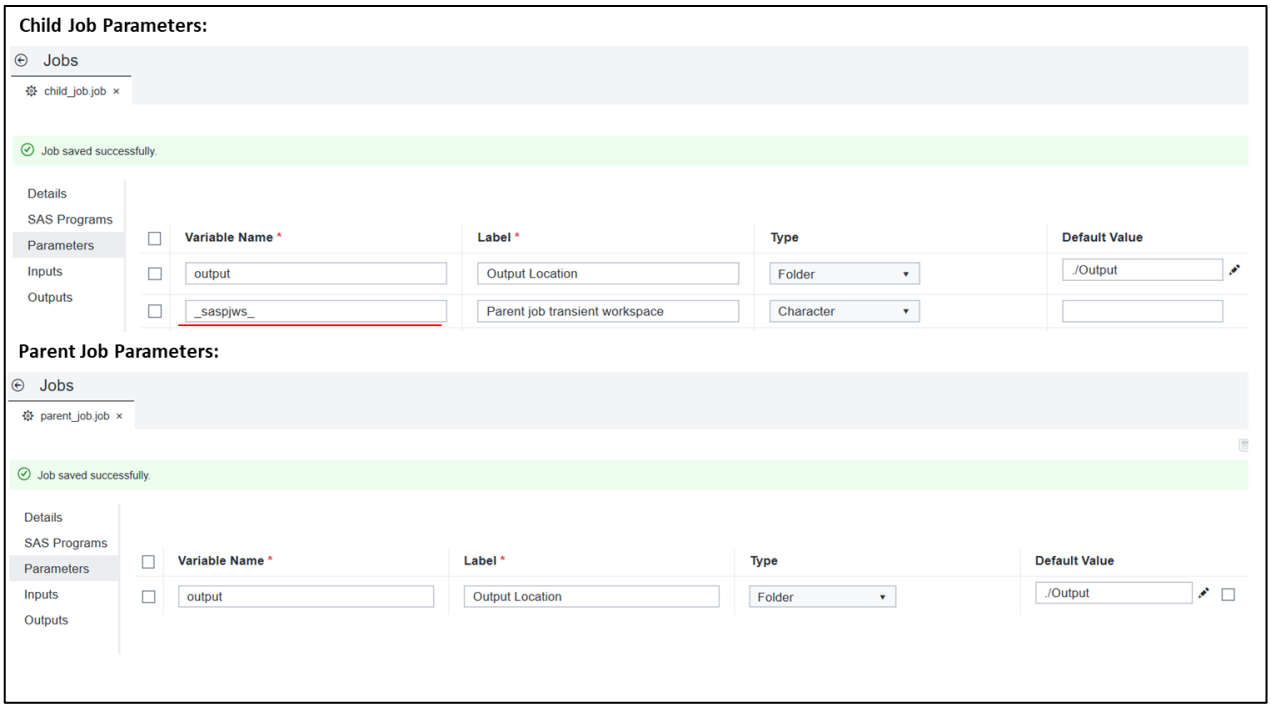

Figure 3: Job parameters used in the example child and parent job. Note that the "_SASPJWS_" parameter (underlined in red) was included as an empty character parameter.

The value of the _SASPJWS_ parameter will be modified in the parent job. The API *GetJobParameters* will be used to create a SAS dataset which contains a list of parameters available in the selected job file.
This dataset can be modified via a simple data step which it used to override the empty value of "_SASPJWS_" with the current value of "&_SASWS_."; i.e. the location of the transient workspace of the currently running (parent) job.

Next, the API *SubmitJob* is used to execute the child job. The call to this API contains an optional parameter to select a SAS dataset with modified job parameters to override its default settings.
Next to this, the program contains a *libname* statement together with a generic data step that is used to simulate a basic report dataset located in the transient workspace of the parent job.

Last, the program contains *%put* statements to print the current value of _SASWS_ to the log file of the parent job. It will be used to confirm the correct location was submitted to the child job.

EXAMPLE PARENT JOB CODE

```
/* Assign report library */
libname output "&output.";

/* Create initial report */
data output.report;
  length message $ 50 transient_workspace $ 300;
  message = "Start of parent job";
  transient_workspace = "&_SASWS_.";
  output;
run;

/* Define the path to the child job (repository) */
%let jobpath=&jobpath.;
%let jobpathrepository=%substr(&jobpath.,%length(&_sasws_.)+1);
```

```
/* Access child job parameters */
  %LSAF_getJobParameters(LSAF_path=&jobpathrepository./child_job.job
                        ,LSAF_version=
                        ,sas_dsname=originalJobParameters);


  /* Populate the empty _SASWSPJW_ parameter */
  data work.JobParameters;
    set work.originalJobParameters (rename=(defaultValue = value));
    if name="_saspjws_" then Value="&_SASWS_.";
  run;

  /* Submit the modified job */
  %LSAF_submitjob(LSAF_path=&jobpathrepository./child_job.job
                 ,sas_dsname=work.JobParameters);

  /* Obtain submission status 10 seconds after job execution */
  %put %sysfunc(sleep(10,1));

  %LSAF_GETSUBMISSIONSTATUS(LSAF_JOBSUBMISSION_ID=&_lsafJobSubmissionId_);

  /* Print the transient workspace path in the log */
  %put [Parent Job] original parent transient workspace _SASWS_:;
  %put [Parent Job] &_SASWS_.;
```

ACCESS THE TRANSIENT WORKSPACE IN THE CHILD JOB

After successful completion of the previous steps, it is possible to call _SASPJWS_ as a macro variable in the child program. First, several *%put* statements are used to print the values for &_SASWS_ and &_SASPJWS_ to the log in order to compare the paths stored in both variables.

Next, a library pointing to the location of the initial report dataset created by the parent program will be assigned. For this, the Output and _SASPJWS_ parameters are being used: By default, SAS adds the &_SASWS_ macro variable to all parameters pointing to a folder location in LSAF. *%let* and *%substr* statements are used to first extract the value from &_SASWS_ from the Output parameter to and then replace it with the one of &_SASPJWS_ which redirects this parameter to the location of the transient workspace in the parent job. This modified parameter is then used in a *libname* statement to assign a library pointing to the location of the report dataset. A simple data step is sufficient to access this dataset and to add another record to it.

The program contains a call to the LSAF API *LSAF_EXISTS* which is used to confirm that the dataset created by the parent job has not yet been moved to the repository.

EXAMPLE CHILD JOB CODE

```
  /* Assign library in parent job transient workspace */
  %let pj_output = %substr(&output.,%length(&_sasws_.)+1);
  %let pj_output = &_SASPJWS_.&pj_output.;

  libname output "&pj_output.";

  /* Confirm that the report does not yet exist in the repository */
  %lsaf_exists(lsaf_path=%substr(&output.,%length(&_sasws_.)+1)/report.sas7bdat);

  /* Access and Update the parent report */
  data output.report;
    set output.report;
    output;
    message = "Start of child job";
    transient_workspace = "&_SASWS_.";
    output;
  run;
```

```
/* Print transient workspace locations in the log-file */
%put [Child Job] Parent workspace:;
%put [Child Job] &_SASPJWS_.;
%put [Child Job] Child transient workspace:;
%put [Child Job] &_SASWS_.;
```

EXPECTED RESULTS

Using this code will allow the child job to make changes to the report dataset in the transient workspace of the parent job (Figure 4). The child job will have added a new line to the report before it has been moved to the repository. The log files confirm that both jobs executed successfully, that the output library was assigned to the same location and that the child job was able to access and modify the dataset inside the transient workspace. The *%put* statements show that _SASPJWS_ was able to point to the parent job transient workspace and that _SASWS_ shows the transient workspace of the child job. The call to the *lsafExists* API shows that the report dataset was not yet present in the repository and confirms the efficacy of this method.

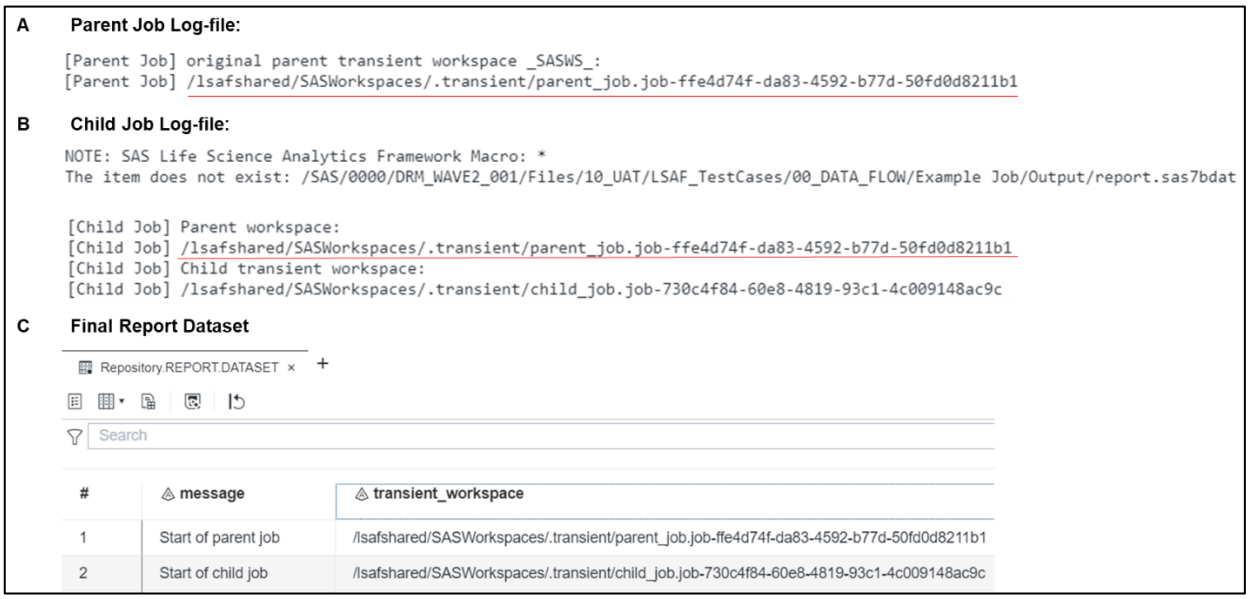

Figure 4: Summary of the expected output in this example. A) Log file generated by the parent job shows the location of its transient workspace (underlined in red). B) Log file generated by the child job confirms that the report dataset does not yet exist in the LSAF repository. The program is nevertheless able to access the path to the report in the transient workspace of the parent job (underlined in red). C) Final output dataset created here. Both jobs were able to access and modify this dataset before it was placed into the LSAF repository.

APPLICATION EXAMPLE: EXECUTE JOBS BASED ON PREVIOUS JOBS

The method presented in this paper can not only be applied to modify a dataset before it has been moved to the repository, but also to read it and to execute parts of following programs in its response. For example, the process flow that was described in the initial problem can be modified to include user-made jobs and programs. These programs usually do not create a report dataset and cannot be included into the final report as the provided default jobs. To deal with this, the notification job was modified so that it is able to read the process flow report to detect unexpected (i.e. user-made) jobs that were run. The notification job will then execute a specific macro that isolates SAS warnings and errors from the log-files of these jobs to include those in the final report.

The general idea behind this code is the same as in the previous example: The child job contains the parameter _SASPJWS_ which is filled by the parent job and allows it to access its transient workspace. The child job makes use of a “DATA _NULL_” data step to read the messages in this report which list which jobs have been executed before. These messages can then trigger the generation of macro variables by a *symputx* call function. In the end, this macro variable is then evaluated by *%IF - %THEN* statements to conditionally execute parts of code in the child job.

The example given here is a simple modification of the previous code, where the child job will detect if the report has been created by the parent job and will create a modified message in the log file and output report (Figure 5).

EXAMPLE CODE TO READ A FILE IN THE PARENT JOB TRANSIENT WORKSPACE

```
  /* Assign parent output library */
  %let pj_output = %substr(&output.,%index(&output.,/SAS/));
  %let pj_output = &_SASPJWS_.&pj_output.;

  libname output "&pj_output.";

  /* Only execute the program if the parent job ran */
  %let continue = N;
  data _null_;
    set output.report;
    if message = "Start of parent job" then call symput("continue","Y");
  run;

  /* Update the report if it was created by the parent job */
  %if &continue. = %str(Y) %then %do;
    %put Parent job execution detected.;
    %put Child job will continue.;

    /* Update the parent report */
    data output.report;
      set output.report;
      output;
      message = "Child job is starting after the parent job";
      transient_workspace = "&_SASWS_.";
      output;
    run;
  %end; %else %do;
    /* The child job will not create an output if the parent job did not modify the
report */
    %put Parent job execution not detected.;
    %put Child job will make no changes.;

 %end;
```

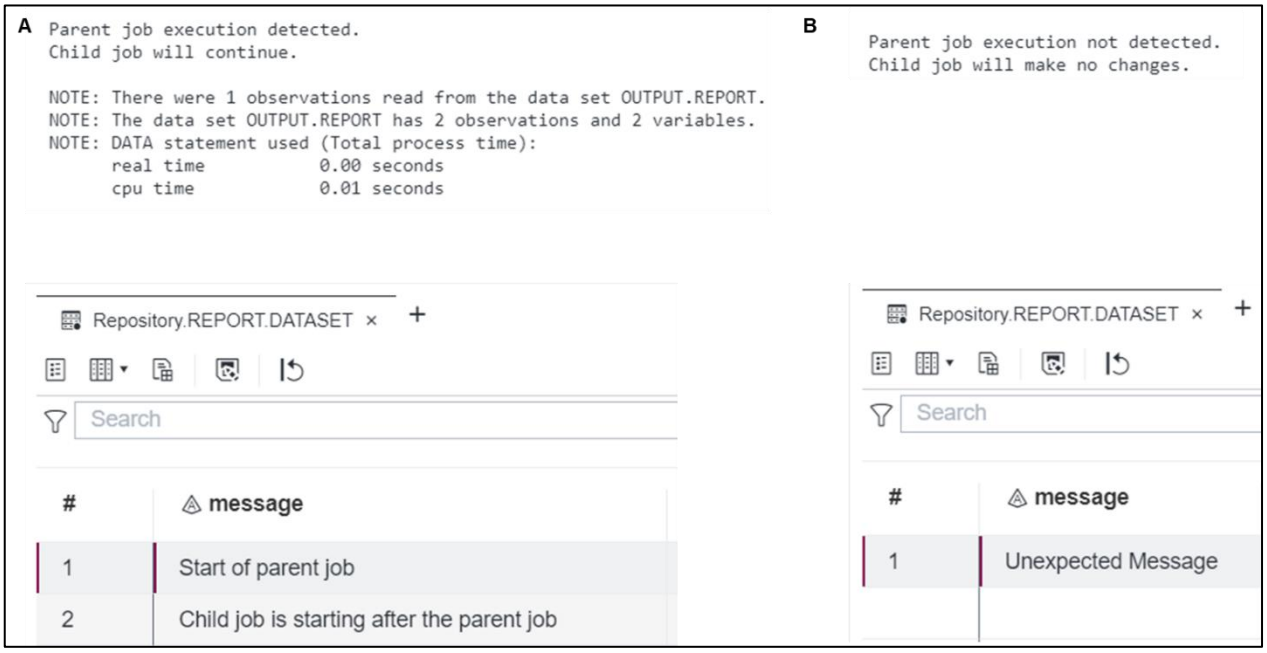

Figure 5: Log file and report dataset created by the modified child job code. A) The child job was executed after the parent job created the report dataset causing it to modify its log message and output dataset. B) The message in the parent job was changed triggering the alternative condition in the child job resulting in a different message and preventing it from changing the output dataset.

CONSIDERATIONS

Using different transient workspaces during an LSAF job is not standard operating procedure and there are several considerations that should be taken into account. The user should keep in mind that the separation of a process flow in different separate workspaces also presents a safety mechanism in LSAF: Jobs will only migrate files back into the repository when their execution was successful and without errors. This prevents the introduction of faulty datasets into the repository where they may override functional files. Next, the general workflow and interactions between repository are controlled and predictable which makes it easy to design jobs to deal with them. Adding too many interactions may cause confusion for the user and could create unexpected problems that are difficult to detect and debug.

Because of this, it is advised to use this approach with caution when dealing with actual study data. Mostly, it should be applied for 'neutral' datasets as the here presented reports that monitor the process flow. Moreover, it is recommended to generate these datasets in distinct LSAF repository locations that do not overlap with any form of data storage or processing.

Following these general guidelines, it is possible to combine different transient workspaces without endangering the integrity of processed clinical trial data.

CONCLUSION

LSAF presents a great platform for the storage and processing of clinical trial data. Its compartmentalisation between repository and (transient) workspace permits to process data in safe and controlled ways. Still, there are certain scenarios where the spatial separation between these environments can interfere with the automation of process flows and causing them to be less efficient or convenient to handle. This paper presented a method to increase the potential of child jobs to communicate with their parent. Using this method under the aforementioned considerations opens new possibilities in the setup, optimization and automation of complex process flows.

ACKNOWLEDGMENTS

The main method presented in this paper was developed with kind support by the SAS institute.
I would also like to thank my colleagues at OCS Life Sciences, especially Bas van Bakel and Radosław Pszczółkowski, for their valuable input and support during the preparation of this paper.

CONTACT INFORMATION

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

Author Name: Kai Wanke
Company: OCS Life Sciences
Address: Ruwekampweg 2G
City / Postcode: 's Hertogenbosch 5222 AT
Work Phone: +31735236000
Email: sasquestions@ocs-consulting.com
Web: https://ocs-lifesciences.com/