Paper 1148-2017

My SAS Grid Scheduler

Patrick Cuba, Cuba BI Consulting

ABSTRACT

No Batch Scheduler? No problem! This paper describes the use of a **SAS DI Studio** job that can be started by a time dependent scheduler like Windows Scheduler (or crontab in UNIX) to mimic a **batch scheduler on SAS Grid.**

INTRODUCTION

As business users of SAS Software sometimes you would like to run your code to a schedule or when data is available for you to run your analysis on. Having software like Platform LSF, Control-M or JAMS would free up a lot of your time to work on something else rather than having to manually run your SAS code to produce insights.

Once you have a scheduler installed then you need someone to manage it and this is where IT would take over. Your reports would then need to be analyzed and likely converted to production ready code. In some cases your code may need to be modified to conform to some sort of code governance and guidelines. Any issues with the code base and IT would be going back to the business user to see how they can fix whatever problem came up. What if you need to make *modifications* to your code? You will have to engage with IT to make changes and follow internal change management procedures with change windows to when you can actually implement those changes.

You could however have the *scheduling software* managed by the business department but then you would need someone in your department upskilled to manage the scheduling software. The solution proposed in this paper has *one SAS DI Job* scheduled through IT. This **single SAS job** will run all users' reports and also allow users the flexibility to change the code to meet their business requirements.

THE CLASSIC BATCH SCHEDULER

What do we mean by **Batch Scheduling**? It means we need to take a batch of programs and run them at a *specified time* and/or after a *certain event* (a trigger). The scheduled program (a job) would need to be monitored, produce logs and generate alerts to be sent showing the program's success or failure. A batch scheduler should also be able to run dependent programs in independent flows, where job 2 runs after job 1 only at the successful completion of job 1.

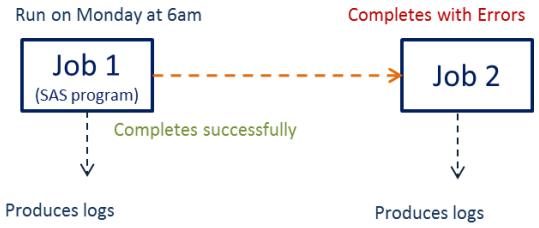


Figure 1 Classic Batch

INTRODUCING MY SAS GRID SCHEDULER

Let's cover off the components we need for our scheduler.

1. An interface to tell the scheduler:
    a. What SAS programs to run (and where to save our logs to);
    b. Dependencies between SAS programs;
    c. File event triggers;
    d. Notifications and whom to send to; and
    e. When to run the SAS program
2. SAS DI Studio with Grid/Connect licensed; and
3. Some custom SAS code logic to further mimic scheduling (more on this later).

LET'S EXAMINE EACH COMPONENT IN MORE DETAIL…

1. THE INTERFACE – *(Tell the Scheduler what to do)*

a. For the purpose of this scheduler the SAS program you want to run on the server should be saved as a .sas file. In SAS Enterprise Guide you only need to right click the program you want to run and click "Save As" to save it to a location that will be accessible by the scheduler. This may be a network path or it could be a location set up under File Navigation (SAS Management Console).

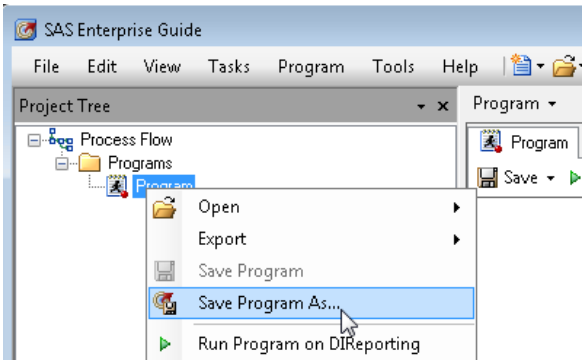


Figure 2: Saving a program in SAS EG

b. **Dependencies**… John works for Pricing and he needs to create a report monthly. John wants to schedule this report because he wants the report to be available before he gets to the office. Mary works for Products. And she wants to capture pricing metrics using the results John has calculated and she too would like to see her reports ready by the time she gets to work. She runs her Product reports monthly too and on the same day as John. But only after John's data is available.

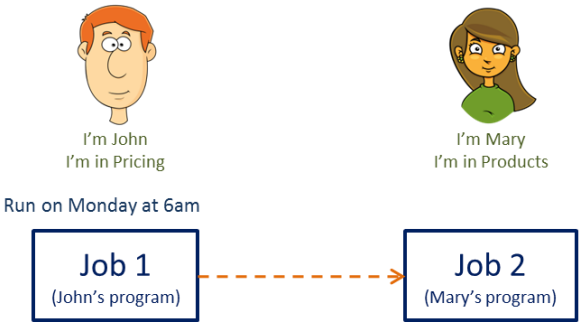


Figure 3 Mary's Report has a dependency on John's Report

c. James works in Finance and he needs John's data too, coincidentally he runs a monthly report on the same day as John and he would really like to have the report ready by the time he gets into the office too. But James has another dependency; he needs to wait for the arrival of General Ledger data in the form of a csv file. James has arranged that the finance team create an **Event Trigger** (a blank text file) in a shared location that notifies James that his data is loaded and ready.

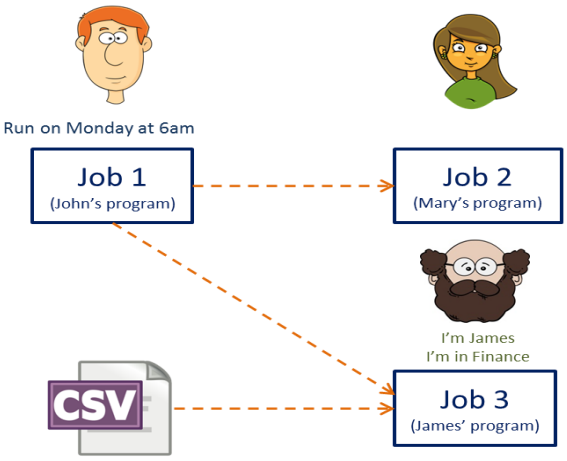


Figure 4 James needs John's data too, and data from a CSV file

d. When John, Mary and James get to work they need to know that their reports are ready. They need a form of **notification**

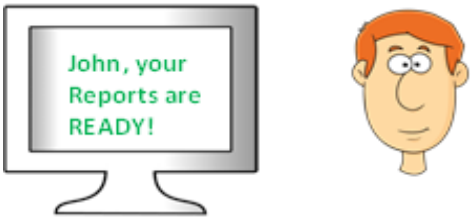


Figure 5 Notification

2. DI STUDIO AND GRID – *(licensed products to enable parallelism)*

SAS DI Studio allows one to rapidly develop standardized ETL processes through SAS code generation and to enable access to some SAS high performing technology like Grid. Grid (once installed) allows SAS to take advantage **Workload Balancing** to ensure the even distribution of processing across the server hardware's processing power.

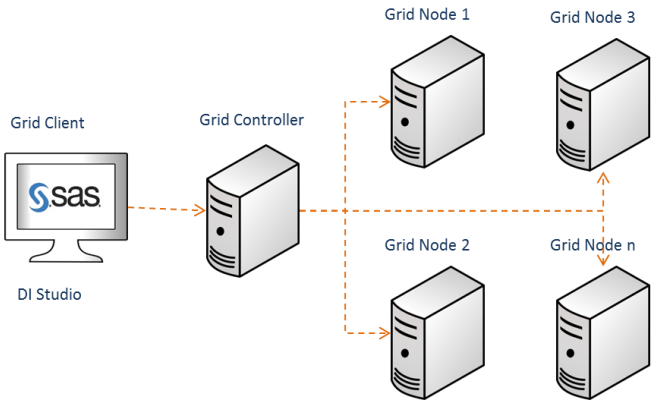


Figure 6 Simplified view of GRID

Through a 3rd party scheduler like Platform LSF we can schedule multiple jobs and Grid takes care of the load balancing by submitting each SAS DI Studio job to an available Grid node. Often multiple SAS DI jobs are running in parallel and in some cases a single SAS DI Studio job will submit multiple SAS execution steps to be run in parallel.

In our example; after John's Program runs and completes successfully, Mary's and James' program can run at the same time because they do not share any dependencies – assuming James' program has the CSV file he was waiting for!

Scheduling is usually left to SAS Administrators

Here is a typical job flow constructed in SAS Management Console and published to Platform LSF Scheduler

Each square (node) you see in this diagram is a DI Studio job; the nodes you see in a vertical straight line can be run in parallel and is managed by Platform LSF and Grid.

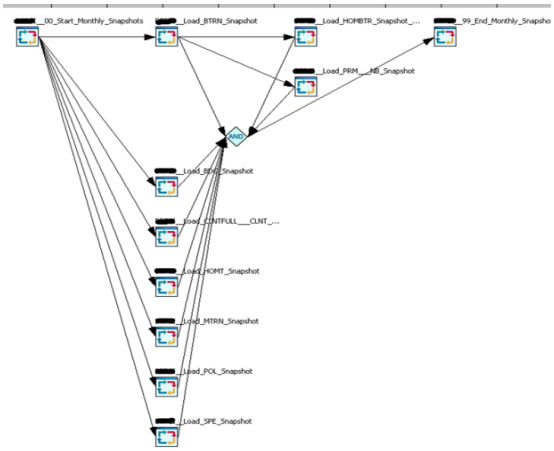


Figure 7 LSF Example

Parallel processing within a SAS DI Studio job.

(Re)introducing SAS DI Studio's Loop and Loop End transformations – they must be used together and have been together since DI Studio 3.3 (circa 2006).

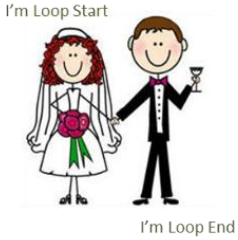


Figure 8 Loops must be used together

They can be found under the Control transformations tree and they act as iterators, A SAS DI Sub-job placed between the Loop Start and Loop End transformation will be repeated by a number you set as input to the Loop Start transformation.

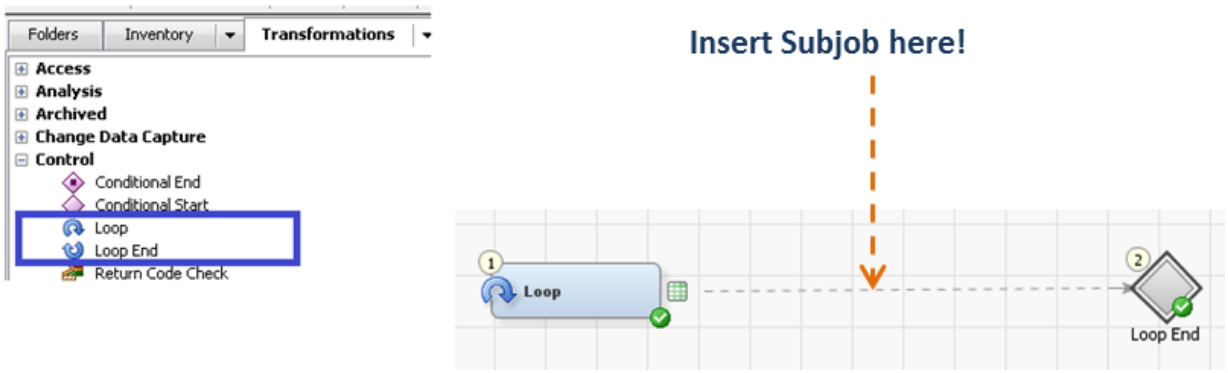


Figure 9 DI Studio Loops

Let's illustrate what is happening in this job to sub-job link by way of some Base SAS code

Base SAS Program

```
Data Control;
   Infile Cards;
   Input Name : $20.;
   Cards;
John
Mary
James
;
Run;

Data _Null_;
   Set Control;
   Call Symput(Compress('Loop_' || Put(_N_, 8.)), Strip(Name));
   Call SymputX('Loop_Max', _N_);
Run;

%Macro DemoLoop;
   %Do I = 1 %To &Loop_Max.;
      Data _Null_;
         Set Control;
         Putlog "NOTE: Name Selected for Handle Loop_&I. is:" Name;
         Where Name = "&&Loop_&I.";
      Run;
   %End;
%Mend;
%DemoLoop;
```

Each data entry represents a single iteration, here we have 3 entries

Each column is a parameter that is parsed to the macro below

This code represents the subjob with parameters parsed from the parent job

Log Entries

```
NOTE: Name Selected for Handle Loop_1 is :John
NOTE: There were 1 observations read from the data set WORK.CONTROL.
      WHERE Name='John';
NOTE: Name Selected for Handle Loop_2 is :Mary
NOTE: There were 1 observations read from the data set WORK.CONTROL.
      WHERE Name='Mary';
NOTE: Name Selected for Handle Loop_3 is :James
NOTE: There were 1 observations read from the data set WORK.CONTROL.
      WHERE Name='James';
```

The Control table in our example acts as a source table for the macro loop (DemoLoop) in the code. Each entry is a parameter to the macro loop and because there are three entries in this control table the loop will iterate three times and parse each of those parameters as macro variables. Iteration 1 parsed the value ‘John’; the second was ‘Mary’ and third was ‘James’. The number of records in the Control dataset controls the number of iterations of the DemoLoop code.

This is how we implement the same thing in SAS DI Studio

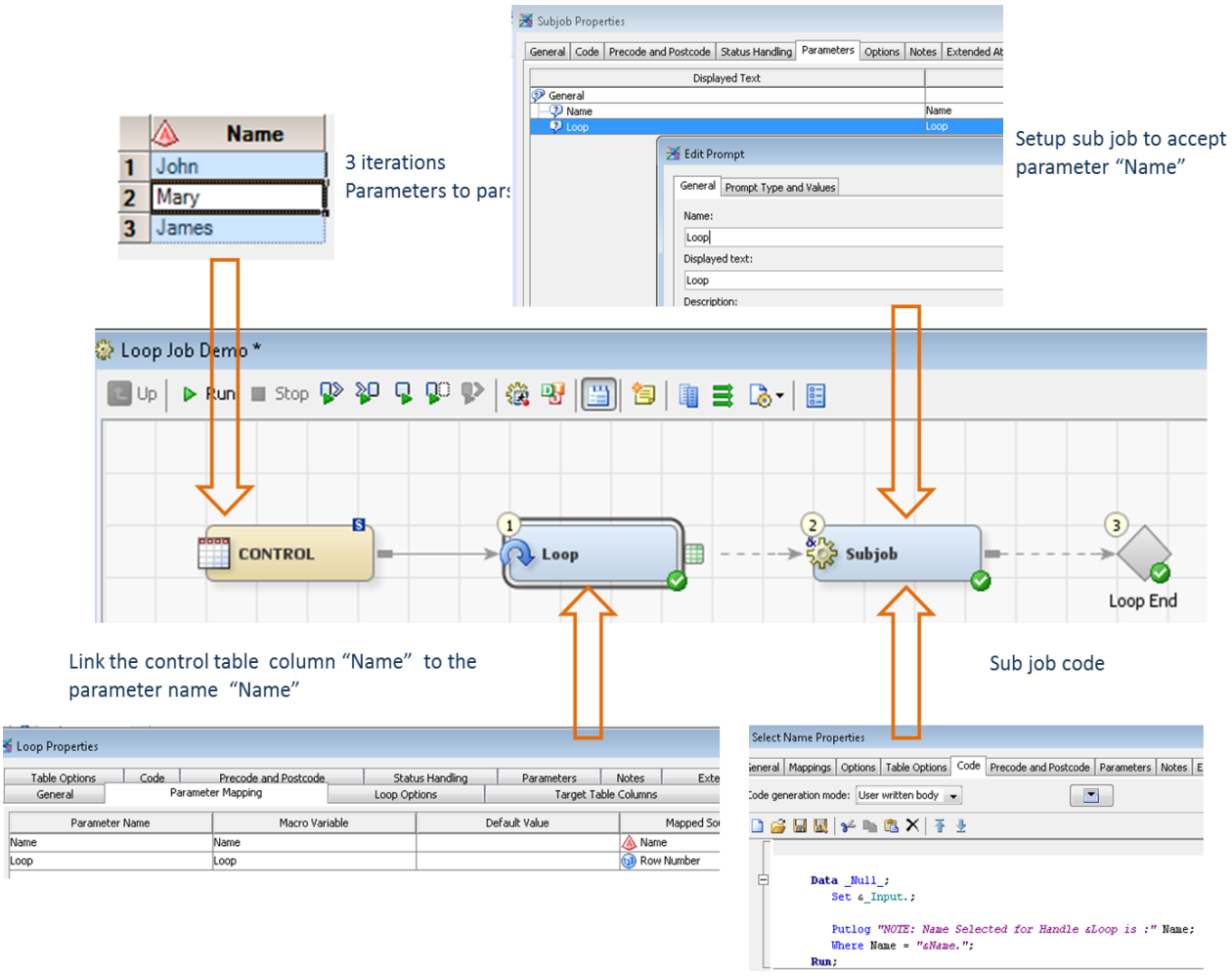


Figure 10: Looping configuration in DI Studio

- Loop Transformation parses the column “Name” value to the Subjob (our DemoLoop)
- Each iteration executes with a parsed parameter

Log Entries

```
NOTE: Name Selected for Handle 1 is :John
NOTE: There were 1 observations read from the data set STAGING.CONTROL.
      WHERE Name='John';
NOTE: Name Selected for Handle 2 is :Mary
NOTE: There were 1 observations read from the data set STAGING.CONTROL.
      WHERE Name='Mary';
NOTE: Name Selected for Handle 3 is :James
NOTE: There were 1 observations read from the data set STAGING.CONTROL.
      WHERE Name='James';
```

Let's illustrate how this is run sequentially; each iteration is essentially a job that runs Base SAS code. Now imagine that each Job has run in the times illustrated below

Figure 11 Running jobs concurrently

Enable Grid/Connect

Although SAS programs are executed sequentially SAS does include products that allow us to run the segments of code in parallel as subroutines or child processes. These products are SAS/Connect and SAS/GRID

When executing these subroutines in parallel SAS will spawn a new vanilla SAS session for each subroutine to run. This means that all the macros, macro variables, libraries, formats and work tables you have assigned or created in the parent SAS session will not be available to the child SAS session – the subjob. But with a few SAS/Connect statements we can code the parsing of these artefacts from the parent to the child routines using Base SAS. Here are a few guidelines to link the artefacts between Parent and Child SAS sessions

1. **Macros**

   Macros created in the Parent SAS session are saved to a SAS catalog entry under the Parent SAS code's work library, these entries can be copied up to the Child SAS session using **Proc Catalog Copy** routine

2. **Macro Variables**

   A new vanilla SAS program will not know what its parent SAS program assigned as macro variables. In order to parse these variables from parent to child SAS program you must use the ***%syslput*** statement in your code

3. **Libraries**

   If you set libraries as Pre-assigned under SAS Management Console these libraries will be immediately accessible to the child SAS sessions.

   Alternatively to parse assigned libraries from the parent session to the child SAS session is to use the ***inheritlib*** option in the **RSUBMIT** statement

4. **Formats**

   To inherit User Defined Formats created in the parent SAS Session's WORK library append the Parent SAS session's Work library to **FMTSEARCH** option in the child session.

5. **Work Tables**

   To make the Parent SAS session work tables accessible by the child SAS process assign the Parent Work library as a permanent library in the child SAS program or use **Proc Upload** to copy the tables up – to parse child work table to the parent session use the inverse **Proc Download** procedure. Or you could save child session work tables permanently.

Going into finer detail of implementing this in Base SAS is not in scope for this paper. Now let's try the same code generation in SAS DI Studio…. double-click "Loop Start" transformation

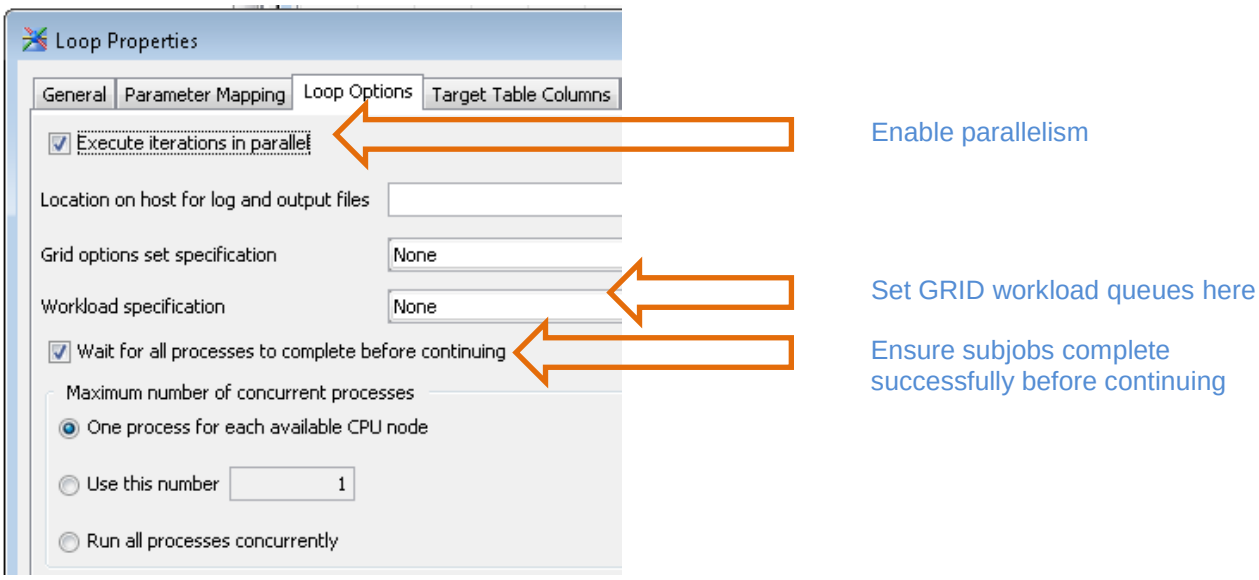


Figure 12: Setting your GRID options in DI Studio

THAT'S IT!!! EASY!!!

SAS DI Studio takes care of the generation of code and all the items we had to configure in the Base SAS sample. We can now see the benefit of running jobs in parallel by the following image.

The total runtime is the value of the longest running job.

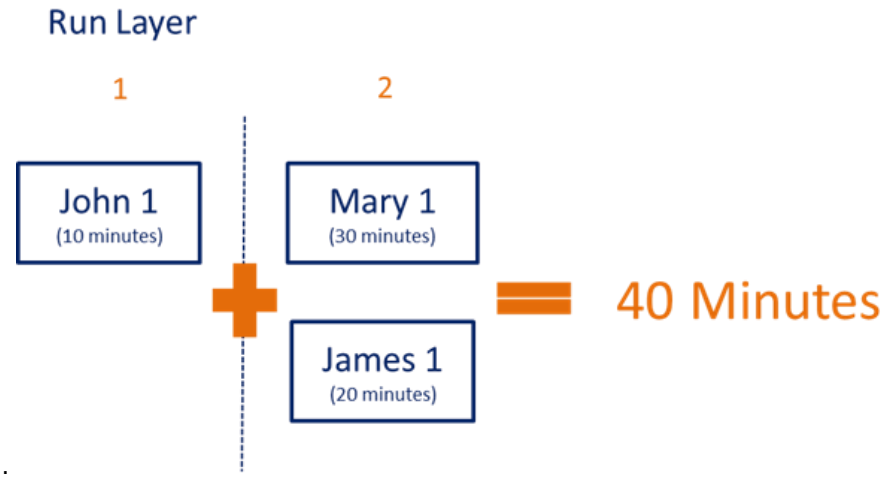


Figure 13 Running jobs in parallel

3. CUSTOM CODE – *(How to mimic a batch scheduler)*

Let's get into the remainder of the components we need to build a scheduler using SAS DI Studio.

a. Control/Interface Table

Report Name	SAS program	Log
Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing
Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product
James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance

Table 1 Control Table

Registered in a SAS dataset are the programs we want to run, we give it a name and where we want our logs to be saved. We have three iterations (John, Mary and James Reports) and three different parameters ("Report Name", "SAS Program" and "Log") we can parse.

b. Batch Server Command Line

Each Report we run must be run as a vanilla session using the appropriate SAS executable to run the report code.

We will use the SAS Batch command line as created in SAS Management Console. To find the command line to execute do the following:

- Go to SAS Management Console
- Under Server Plugin expand "SAS Batch Server" and Right Click "SAS Data Step Batch Server" and click "Properties" and click "Options"

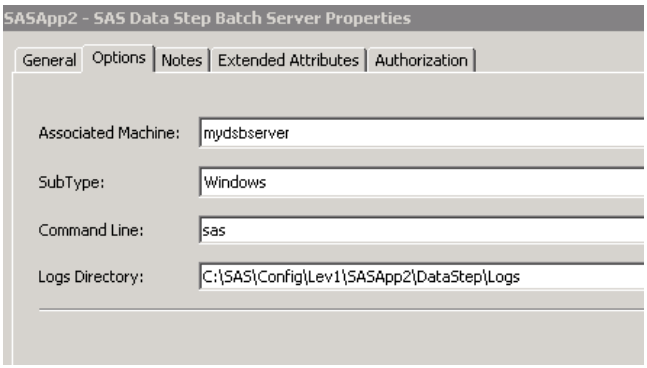


Figure 14: SAS Batch Server configuration

We expand the command line to include entries from our control table as parameters and we can add a few parameters of our own.

We will parameterize the following with command line switches:

- SAS program to run (after –sysin switch)
- Log location and name (after –log switch)
- Additional switches, like –noxcmd (for security reasons)

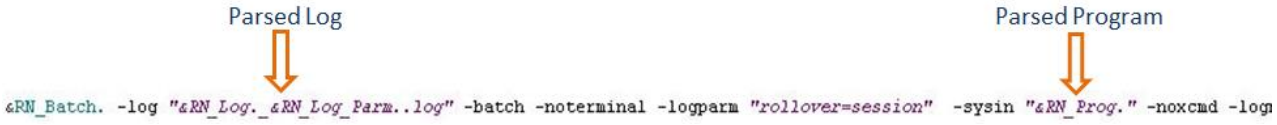


Figure 15 SAS batch command line

c. Assigning Dependencies

We need to instruct SAS DI Studio the order in which we want it to run the SAS programs.

Let's start by numbering the programs to run sequentially as "Report ID".

We add a new column called "Depends On" that relates to the "Report ID" a report is depending on.

"Depends On" and "Report ID" are new parameters that we can now parse

Report ID	Report Name	SAS program	Log	Depends On
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	1
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	1

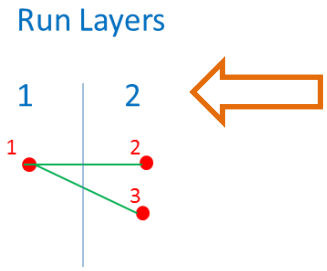
Table 2 Control Table with Dependency

Looking at the dependencies we can establish a Run Order, i.e. Report IDs 2 and 3 cannot run without the successful completion of Report ID 1. Therefore Report ID 1 will run 1st and Report IDs 2 and 3 will run second

Report ID	Report Name	SAS program	Log	Run Order / Run Layer
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	1
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	2
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	2

Table 3 Control Table with Run Order

Notice that some programs can run at the same time. We can run "Marys Report" and "James Report" simultaneously. They have no dependencies on each other! For purposes of this paper we will call this relationship a run layer



We will have 2 Run Layers

Figure 16 Run Layers

d. SAS DI Studio

We will need multiple SAS DI jobs and subjobs to support parallelism and scheduling we want to emulate

L1 is the parent of L2 and L2 is the parent of L3.

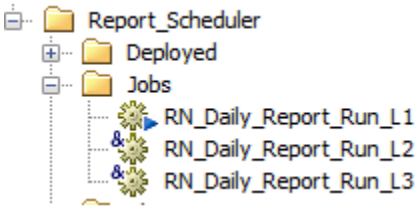


Figure 17 SAS DI Studio Metadata Jobs

Let's break down how each job and subjob is configured

In job RN_Daily_Report_Run_L1 (establish the number of Run_Layers to run)

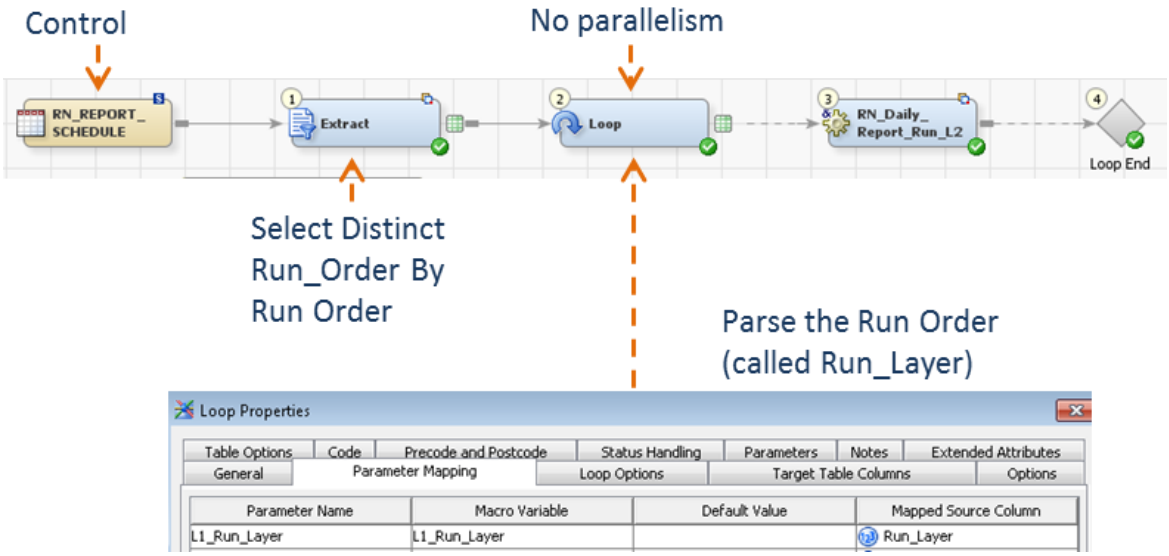


Figure 18 Level 1 DI Studio Job

- Select Distinct Run_Layer by Ascending Run_Layer

  We make sure that Reports intended to run first run first
- No parallelism, each Run Layer must run sequentially
- Parse **Run_Layer** value to Subjob RN_Daily_Report_Run_L2

In job RN_Daily_Report_Run_L2 (in the current Run_Layer, what programs can we run in parallel)

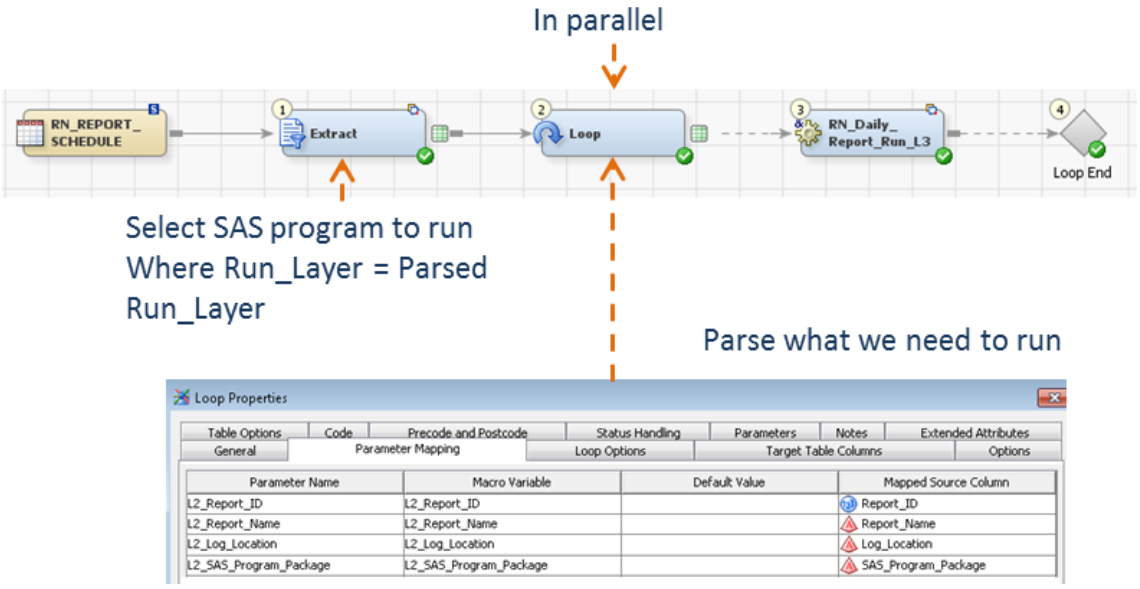


Figure 19 Level 2 DI Studio Job

- Parse SAS program name, log location, Report ID to Subjob RN_Daily_Report_Run_L3

In job RN_Daily_Report_Run_L3 (execute the program using the SAS Batch command)

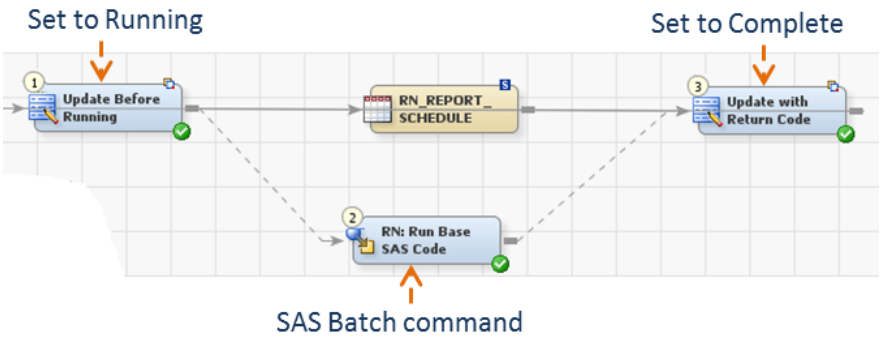


Figure 20 Level 3 DI Studio Job

- We track our SAS programs with additional columns “Report Status” and “Update Time”
- If we are reading and updating a central table then how do we ensure we do not lock out a READ while an UPDATE is happening?

Report ID	Report Name	SAS program	Log	Run Order	Report Status	Update Time
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing\Pricing_12SEP16_0600.log	1	Complete	06:30
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product\Product_12SEP16_0631.log	2	Running	06:31
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance\Finance_12SEP16_0631.log	2	Complete	06:45

Table 4 Control Table with status updates

e. Alert to job statuses

Our Report Status can hold the following values

Report Status	Description
Success	Job Competed successfully
Ended with warnings	Job Competed successfully but the log had warnings in it
Failed	Job was aborted
Did not run	A job this job is dependent on failed or did not run

Table 5 Job completion statuses

If a scheduled SAS program fails then a dependent scheduled SAS program will not run. If a SAS Scheduled program did not run than any other dependent scheduled SAS programs will not run in turn.

Once the Report Scheduler has completed running all the reports an email is sent to each registered notification email with the SAS code completion status. Emails are registered as a parameter here too.

Report ID	Report Name	SAS program	Log	Depends On	Email Notification
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing		john@mycompany.com
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	1	mary@mycompany.com
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	1	john@mycompany.com james@mycompany.com

Table 6 modified Control Table with email notifications

If John's program failed then we don't want Mary and James' SAS programs to run so we add some custom logic to DI Studio to check the completion status of a dependent Report.

Here is the modified subjob RN_Daily_Report_Run_L3.

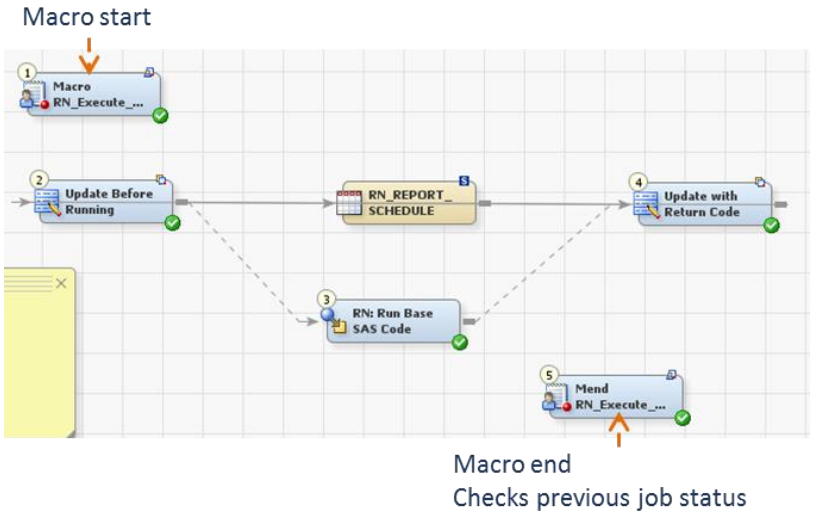


Figure 21 Modified Level 3 DI Studio Job

- Macro Start and End will only allow the DI contents to run if the previous SAS program was successful or has no dependencies

f. Date component

Let's modify the control table further and add a parameter to denote several date schedules. Not everyone will schedule their programs to run on the same day unlike our convenient example.

Time Event	Description
2016/12/02 02DEC2016	Will only ever run once
Monday	Will run ever Monday
Daily	Runs everyday
Beginning End Beginning:3	Runs at the beginning of the month Runs at the end of the month Runs on the third day of the month
Quarterly Quarterly:5	Runs at the beginning every quarter Runs on the fifth day each quarter

T able 7 Time events

Report ID	Report Name	Run Date	SAS program	Log	Depends On	Email Notification
1	Johns Report	Beginning	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing		john@mycompany.com
2	Marys Report	Beginning	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	1	mary@mycompany.com
3	James Report	Beginning	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	1	john@mycompany.com james@mycompany.com

Table 8 Control Table with Reports scheduled to run at the beginning of the month

g. Allowing Triggers

James' SAS code cannot run until his csv file arrives otherwise it will fail

We need to expand the interface table even further with another parameter.

Report ID	Report Name	Run Date	SAS program	Log	Depends On	Email Notification	File Trigger
1	Johns Report	Beginning	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing		john@mycompany.com	
2	Marys Report	Beginning	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	1	mary@mycompany.com	
3	James Report	Beginning	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	1	john@mycompany.com james@mycompany.com	F:\SAS\Logs\GL_&sydate9.

Table 9 Control table with a file trigger

The file trigger event detected will be polled by custom code in SAS DI Studio code included in the batch scheduler. Every five minutes the scheduler will check for the registered trigger file until it finds it. Here we used a macro variable ***&sysdate9***. as a suffix. The trigger file will need to arrive in the format GL_29JAN2017. This ensures that James' program runs after the latest trigger.

4. **BONUS** WHAT IS THE INDIRECT DEPENDENCY PROBLEM?

There is an efficiency flaw in this design; our scheduling is based on dependent reports running together and as we have seen any SAS program registered in the control table with the same calculated Run Order (Run_Layer) will run at the same time (thanks to Grid and Connect), as is the case with James' and Mary's scheduled programs

Let's expand that example further, John has another independent report to schedule, and James needs data from that too in another Report.

Report ID	Report Name	SAS program	Log	Depends On
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	1
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	1
4	Johns Report 2	F:\SAS\Programs\Pricing_2.sas	F:\SAS\Logs\Pricing	
5	James Report 2	F:\SAS\Programs\Finance_2.sas	F:\SAS\Logs\Finance	4

Table 10 Expanded Control Table example

The scheduler will assign the Run_Order as follows

Report ID	Report Name	SAS program	Log	Run Order (Run_Layer)
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	1
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	2
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	2
4	Johns Report 2	F:\SAS\Programs\Pricing_2.sas	F:\SAS\Logs\Pricing	1
5	James Report 2	F:\SAS\Programs\Finance_2.sas	F:\SAS\Logs\Finance	2

Table 11 Expanded control table with Run Orders

The new Reports are assigned the same Run Orders because “Johns Report 2” has no dependency and should run first but at the same time as “Johns Report”. Meanwhile “James Report 2” is dependent on “Johns Report 2” and nothing else and will run second at the same time as “Marys Report” And “James Report”.

Let’s illustrate that further with some run times…

What? John’s and James’ new reports are not dependent on their existing reports? But because these reports are independent they will be assigned the same Run Order and run at the completion of the longest running report. “John 2” completed in 5 minutes but “James 2” must wait for “John 1” to complete before beginning execution. In other word, Run Layer 2 can only begin after the completion of Run Layer 1.

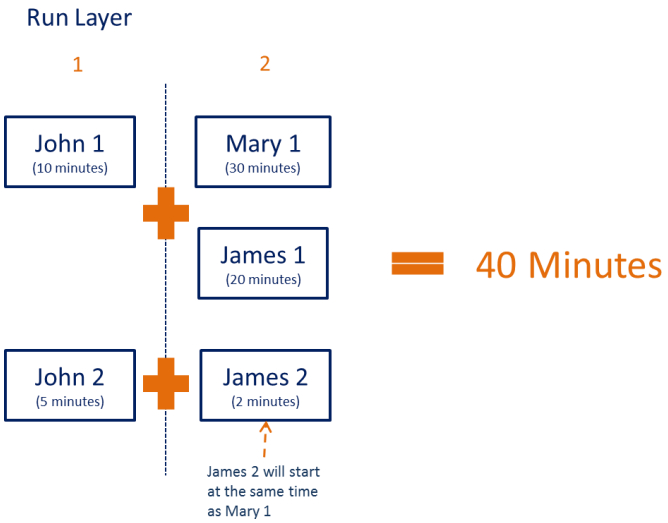


Figure 22 Indirect Dependencies

Solution: We split the schedule into Groups

We calculate a new parameter at run time called “Run_Group”.

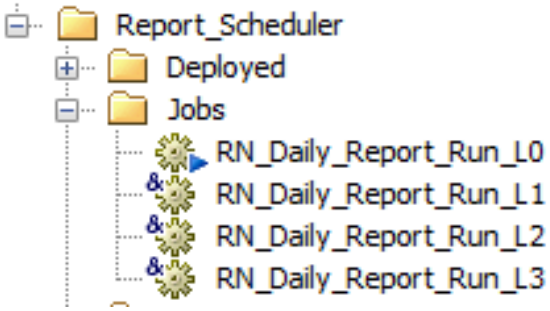
Report ID	Report Name	SAS program	Log	Run Layer	Run Group
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	1	1
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	2	1
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	2	1
4	Johns Report 2	F:\SAS\Programs\Pricing_2.sas	F:\SAS\Logs\Pricing	1	2
5	James Report 2	F:\SAS\Programs\Finance_2.sas	F:\SAS\Logs\Finance	2	2

Table 12 Expanded Control Table with Run Groups

We add a another Parent DI Studio Job to the hierarchy

L0 is the parent of L1.

We are now able to split the DI Studio jobs further across the Grid.



Let's take a closer look at the job RN_Daily_Report_Run_L0 (establish what reports must run together)

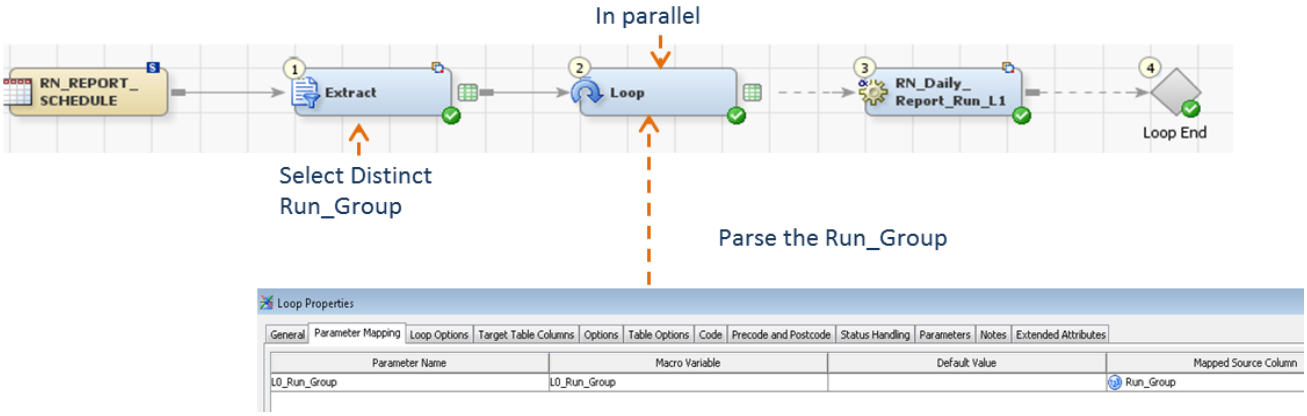


Figure 23: Adding SAS DI Job level 0

- Select Distinct Run_Group - to determine how many groups we will need
- Run Groups 1 and 2 will run in parallel
- Parse **Run_Group** number to Subjob RN_Daily_Report_Run_L1

In essence we have separated the control table into two independent control tables.

Report ID	Report Name	SAS program	Log	Run Layer
1	Johns Report	F:\SAS\Programs\Pricing.sas	F:\SAS\Logs\Pricing	1
2	Marys Report	F:\SAS\Programs\Product.sas	F:\SAS\Logs\Product	2
3	James Report	F:\SAS\Programs\Finance.sas	F:\SAS\Logs\Finance	2

Table 24 Group 1

Report ID	Report Name	SAS program	Log	Run Layer
4	Johns Report 2	F:\SAS\Programs\Pricing_2.sas	F:\SAS\Logs\Pricing	1
5	James Report 2	F:\SAS\Programs\Finance_2.sas	F:\SAS\Logs\Finance	2

Table 25 Group 2

And now the scheduling picture to us will look like this….

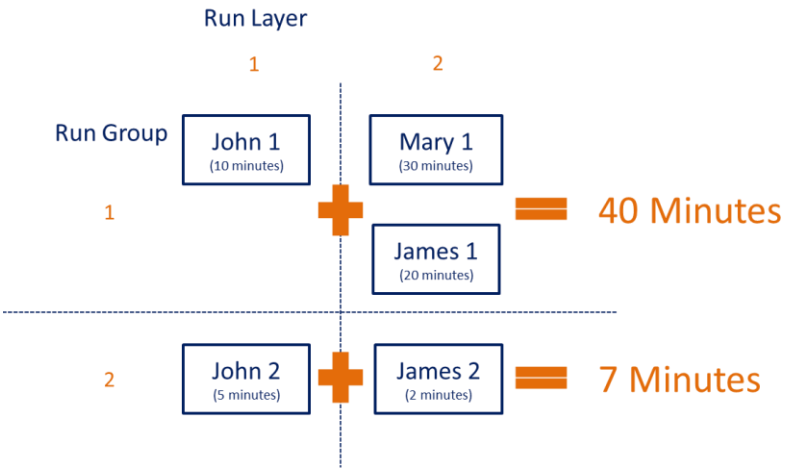


Figure 26: Fully optimized schedule

CONCLUSION

Using a bit of know-how and imagination you can use SAS DI Studio and Grid (or Connect) to perform not just ETL but powerful applications to enable the business to function faster and more efficiently. Building your own scheduler is just one of the practical applications I have built through the years and it can leverage high performance technologies like SAS Grid and Connect. If you are licensed for either product SAS DI Studio will generate the code for you. If you are not licensed this custom scheduler will still function, but the parallelism is lost.

My SAS Grid scheduler can handle any number of SAS programs, calculate their run order and decipher which reports must run in groups.

How it calculates the Run_Layers and Run_Groups would be the subject of another paper.

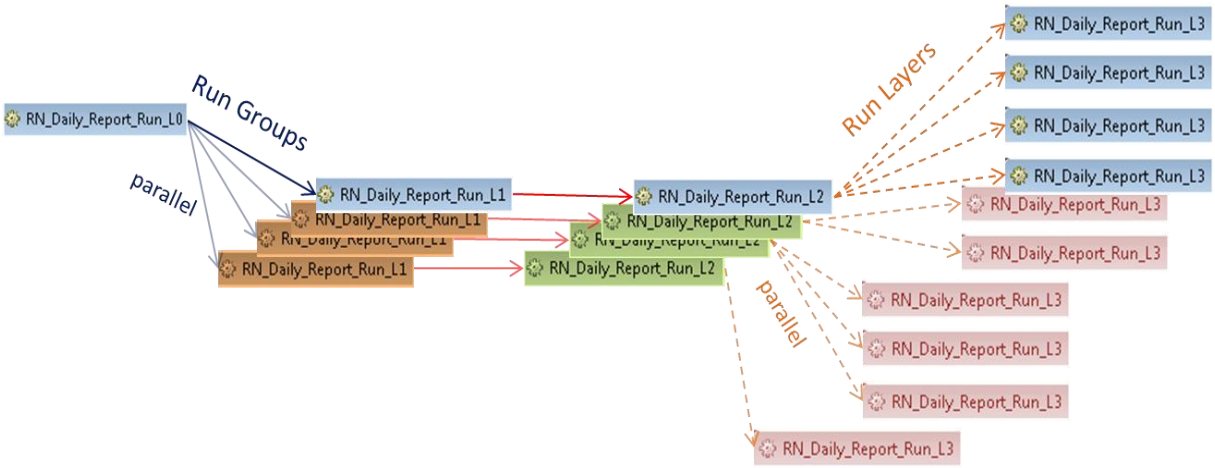


Figure 27 Schedule when it is running

Here is a screen dump of the installed components in DI Studio to make this work

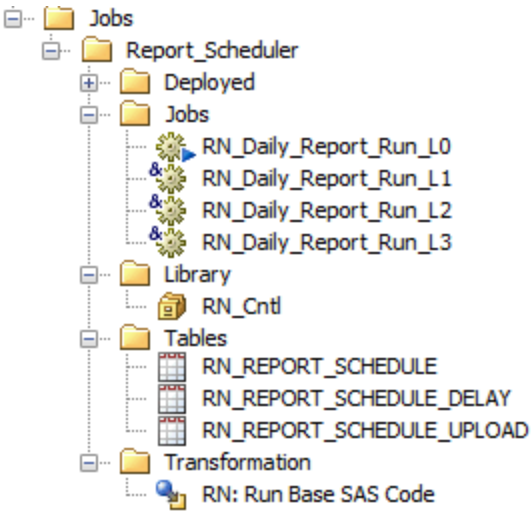
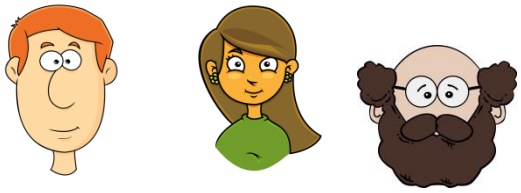


Figure 28 Metadata Folder Tree structure for installed components

SPECIAL THANKS TO JOHN, MARY AND JAMES FOR PARTICIPATING



CONTACT INFORMATION

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

Patrick Cuba
Principal Consultant
Cuba BI Consulting
+61 (0) 458 912 634
patrickcuba8@gmail.com
https://au.linkedin.com/in/patrickcuba

REFERENCES

SAS® Data Integration Studio, https://support.sas.com/documentation/onlinedoc/etls/

Administering SAS Data Integration Studio, https://support.sas.com/documentation/cdl/en/bidaag/69541/HTML/default/viewer.htm#n1ls1shqj6td2hn1a1p1u8ylcqoy.htm

SAS Grid Topology, http://support.sas.com/documentation/cdl/en/gridref/67371/HTML/default/viewer.htm#p1wtsd898mxckin1mmbemp5uzwds.htm

Defining a SAS DATA Step Batch Server, http://support.sas.com/documentation/cdl/en/scheduleug/68697/HTML/default/viewer.htm#p1nzohoe0nkyqfn17uori4m82dwl.htm

Scheduling in SAS® 9.4, http://support.sas.com/documentation/cdl/en/scheduleug/68697/HTML/default/viewer.htm#titlepage.htm