Paper 1693-2014

Conditional execution 'Switch Path' logic in SAS® Data Integration Studio 4.6

Prajwal Shetty Deviprasad, Tesco

ABSTRACT

With the growth in size and complexity of organizations investing in SAS® platform technologies, the size and complexity of ETL subsystems and data integration jobs is growing at a rapid rate. As always data integration developers are pushed to come up with new and innovative ways to improve process efficiency in their ETL design to meet the increasingly demanding SLA's.

The ability or the feature to conditionally execute a code branch or switch code execution paths based on the runtime parameters in a SAS® Data Integration Studio job is an extremely useful technique for improving process efficiency in an ETL subsystem.

This paper presents an innovative technique to providing a parameterized dynamic execution custom transformation that can be easily incorporated into any SAS® Data Integration Studio job to provide conditional execution or process path switching capabilities.

INTRODUCTION

In an ETL world, the aim of any data integration task is to ensure all sources of business data are integrated as efficiently as possible. It is mainly concerned with the repurposing of data via transformations, it should be a value adding process and also should be the product of collaboration.

Modularization of a common or repeatable process is a fundamental part of the entire data integration layer design and development. This has been one of the key areas to focus in order to build an efficient data integration system.

SAS® DATA INTEGRATION STUDIO

SAS® Data Integration Studio, an ETL tool from SAS® provides the right framework, ready-to-use transformations and necessary utilities required for designing an efficient and well maintained ETL subsystem. A graphical user interface ETL tool such as SAS® Data Integration Studio provides many advantages over the use of handwritten user developed code. It is self-documenting, supports collaboration and code re-use. It also provides a common metadata layer simplifying and reducing the maintenance overhead of an ETL subsystem or any data integration task.

Over the years a standard blueprint has emerged for business intelligence solutions with regards to the data integration layer and is typically divided into 3 main sections:

1. Snapshot Layer – Area for extracting and storing the operational source data.
2. Staging Layer – Area for transforming the data as per the organization requirements.
3. Foundation Layer – Area for absorbing the transformed data into warehouse.

Evidently, the different layers perform different functions. The functions are generally realized by group of data integration jobs designed and maintained using SAS® Data Integration Studio. It is really important to understand that these jobs have to be built with real efficiency to cover all the important aspects of an ETL design.

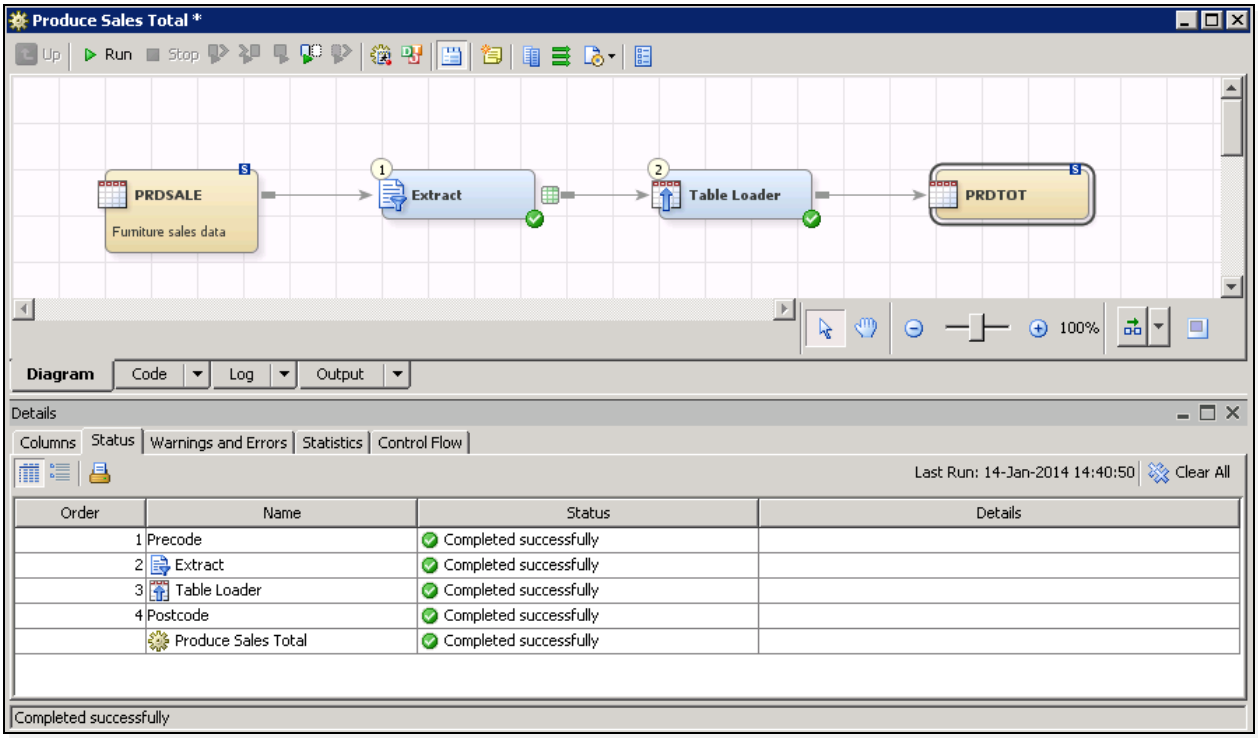


Figure 1. An example of a simple SAS® Data Integration job

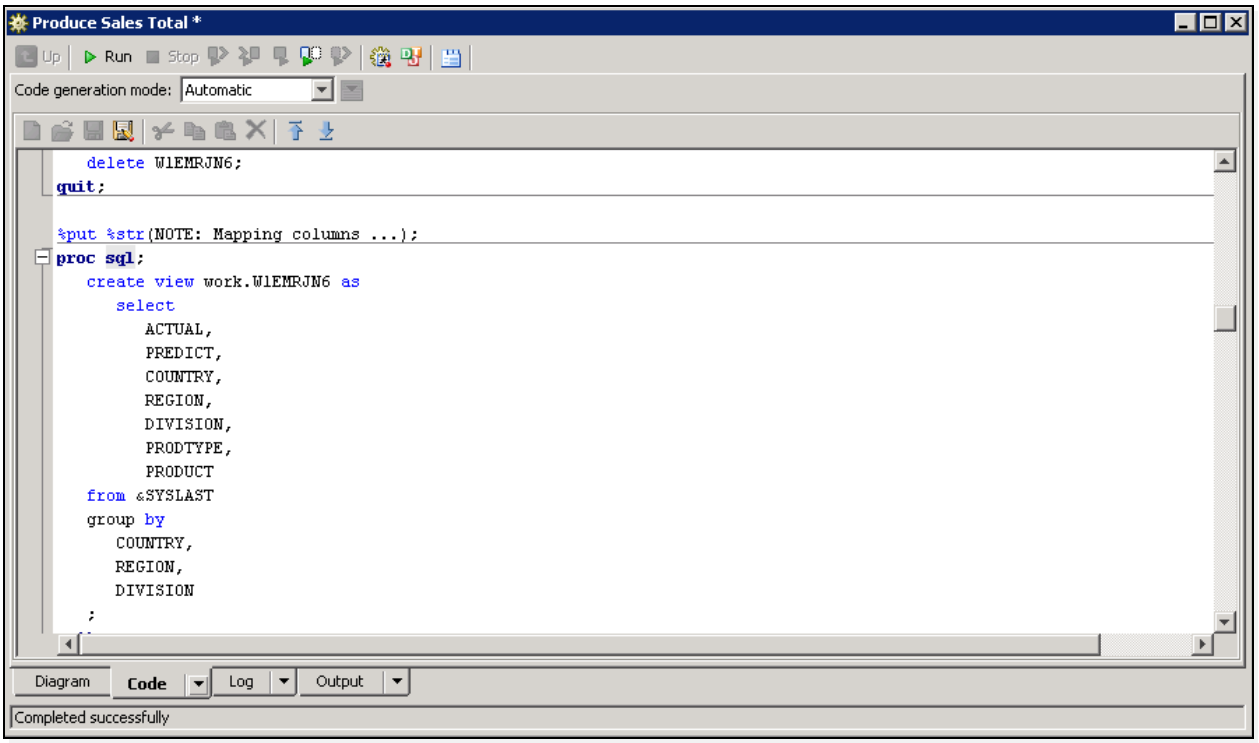


Figure 2. An example of SAS® Data Integration Studio automatic generated code

MODULARIZATION: CUSTOM TRANSFORMATIONS IN SAS® DI STUDIO

SAS® Data Integration Studio provides a graphical user interface with wide variety of inbuilt ETL transformations and utilities. All these transformations are easily available for a developer to readily use in the job. With advanced metadata integration, SAS® Data Integration Studio really provides the developer a power to visualize and build efficient data integration layer.

Apart from the inbuilt transformations, the custom transformation generator in SAS® Data Integration Studio provides a mechanism for modularization and re-use of objects. The custom transformation or sometimes called user generated transformation can be simply visualized as a SAS code encapsulated within a metadata layer to serve a specific purpose in data integration. This is a very useful and powerful feature and provides the developer the capability of designing and building a re-usable component in SAS® Data Integration Studio. It is really important to work within an available framework with maximum reusability in mind.

Benefits of custom transformations in SAS® Data Integration Studio:

- Custom transformation wizard makes it very easy to design and maintain.
- Easy to use 'Drag and Drop' feature.
- Easy metadata management.
- Repeatability and Modularization.

'SWITCH PATH': THE KEY THAT UNLOCKS THE DOOR

"How do I create a single SAS® Data Integration job that can conditionally execute a node/branch on one day and another branch on a different day?"

"How do I run day-wise algorithms in a single job?"

Above are some of the common questions within the SAS® Data Integration developer community and is a much debated topic. The problems like the above are normally solved by either using a 'user-written' code or by replicating the job with small changes to serve the purpose.

We have to understand that there are numerous overheads associated with the above approaches and can induce different problems at different layers of the process flow. These are discussed in detail along with a remedy in this paper.

To understand the concept better, let's understand the problem with a simple case study. The requirement at hand is simple and straight forward –

"Design a job to process a specific day variant of the algorithm on each day from Monday to Sunday".

Now let's understand the above requirement in detail and break the thought process towards design from a data integration developer's point of view:

- Firstly, we have to design a data integration job to execute the 'algorithm' assuming the source remains same.
- Also, as the requirement says there are 7 different versions (day variant) of the 'algorithm' and each has to be executed on the specific day of the week.

Before we start designing a solution, as a data integration developer there are certain principles to which the solution should adhere:

- Easy to develop and easy to maintain.
- No overhead of redundancy.
- High degree of modularization.
- Adhere to code standards.
- Follow the ETL principles.
- Easy to support under the configuration management process.
- Easy for documentation and reports.

Now that we have seen all the principles, the requirement can be visualized as shown below:

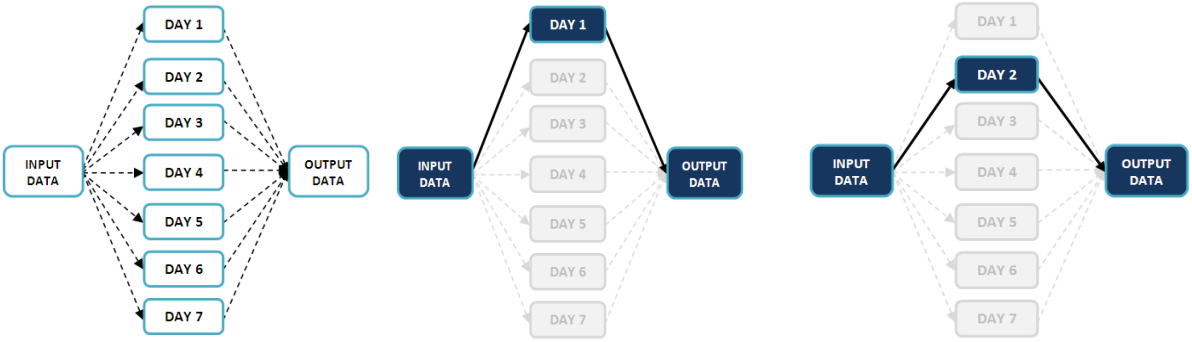


Figure 4. Case study - Day wise 'SWITCH PATH' (conditional) processing

Considering all the above, there are 2 possible standard approaches to design a solution:

1. Design a single data integration job using a 'user-written code' with multiple 'IF' statements. The 'user-written code' will then process only specific algorithm based on the day of the run.
2. Design multiple versions of the same job each scheduled to execute the day variant algorithm based every day.

Unfortunately, there are drawbacks associated with each of the approach listed above.

The first approach uses a single job but with a 'user-written code' which is not recommended in a well-designed ETL subsystem for reasons like maintainability, modularity and repeatability.

The second approach uses no 'user-written code' but introduces multiple versions of the same job and hence adds an overhead of code duplication, redundancy and increased maintenance.

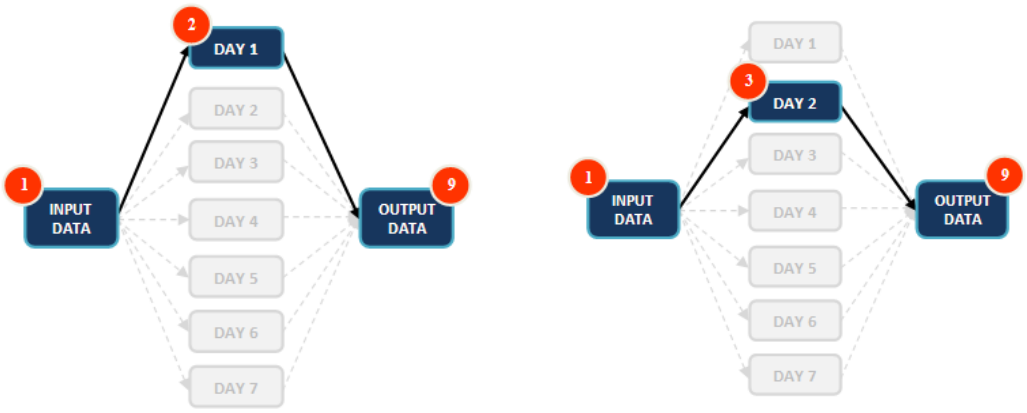


Figure 5. Case study – Understanding the 'Path' and 'Switch'

As shown in the flow diagram above there are multiple branches associated with a specific algorithm and it has to be conditionally processed on a specific day.

The node #1, node #2 and node #9 together form the first branch (or the path) and node #1, node #3 and node #9 together form the second branch (or the path) in the entire process flow. Similarly, all 7 branches can be visualized between node#1 and node #9.

Each branch of different nodes represents the 'day-wise' processing as discussed in the requirement. Each 'path' has to be 'switched on' or 'switched off' based on the day parameter to fulfill the requirement.

Now this process of conditionally switching the group of nodes or a branch based on a parameter during the runtime

is called '**SWITCH PATH**' processing.

Unfortunately for us, SAS® Data Integration Studio as of now doesn't provide a built-in ready to use transformation which can conditionally execute only some of the nodes depending on an evaluation of a conditional statement within in a single data integration job as shown above.

Now '**SWITCH PATH**' comes to our rescue and opens up a new dimension to solve the above problem in SAS® Data Integration Studio.

'**SWITCH PATH**', is an innovative technique implemented using SAS® Data Integration Studio custom transformations. It is intelligently built to conditionally execute specific 'branch/node' in SAS® Data Integration Studio. The transformation provides a reusable module for solving the conditional execution limitations of standard SAS® Data Integration Studio jobs.

NOTE – *The term 'SWITCH PATH' is used as a generic term to refer the conditional execution logic in some context and also to refer to the custom transformation in other contexts of this paper.*

CAN I CREATE A 'SWITCH PATH' CUSTOM TRANSFORMATION?

'SWITCH PATH' custom transformation is designed using custom transformations built specifically to handle the conditional execution of nodes. It includes 3 custom transformation components:

1. Switch Start – Should be at the start of the branch
2. Switch Branch End – Should be at the end of each branches
3. Switch End – Should be at the end of all branches after Switch Branch End

To answer the question - Yes! You can also create a 'SWITCH PATH' custom transformation all by yourself in SAS® Data Integration Studio in easy steps as explained below:

1. In SAS® Data Integration Studio, under 'File' menu click on 'New' and then on 'Transformation'.
2. Give 'Switch Start' as the name and place it under the right inventory category – preferably 'Control'
3. On clicking 'Next' it will open up a code placeholder. Copy paste the below code. The paper will discuss the code in detail in later sections

```
/*****************************************************************
* Transformation - Switch Begin                                  *
*****************************************************************/

/* Display Parameters */
%put NOTE: Starting to execute SWITCH...;
%put NOTE: Parameter received during the runtime is
%sysfunc(trim(&_input_parameter.))...;

/* Takes the runtime parameter for the conditional execution */
%let node=&input_parameter.;

/* Turn off the warning messages */
OPTIONS NOQUOTELENMAX;

/* The below macro variable auto-increments and checks for every node */
/* Default should be always 1 */
%let run_node=1;
%let check_node=1;

%put NOTE: Initialization of variables complete...;

/* Begin of Switch_Start */
%macro switch exec(run node=);
        %put NOTE: Inside the parent macro and about to compile the child macro...;
        %put NOTE: Evaluating the branch %trim(&check node.)...;
        %if &run_node.=&check_node. %then %do;
                %put NOTE: Inside the child macro and compiling macro number
%trim(&run_node.)...;
                %macro switch_node_all;
                        %put NOTE: Executing the child macro now...;
                        /* End of Switch_Start */
```

4. Click ‘Next’ and create a new prompt of type ‘Text’ with name as ‘Input_Parameter’ as shown below.

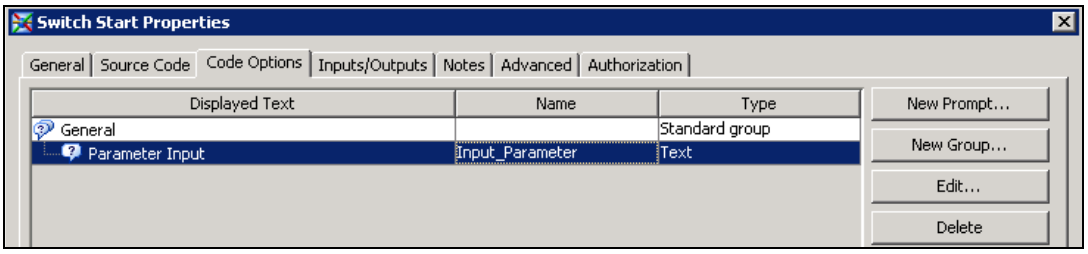


Figure 6. Prompts

5. Click ‘Next’ and provide the parameters as shown below (based on your requirements)

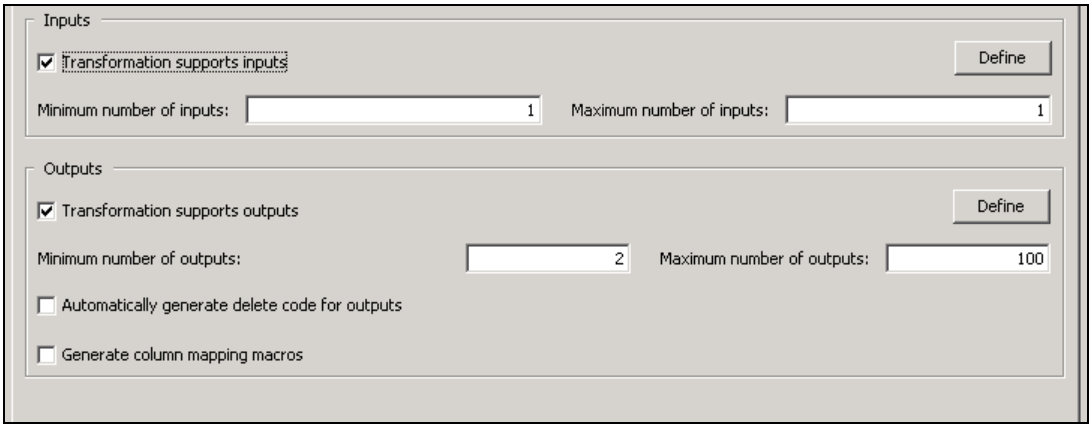


Figure 7. Inputs/Outputs

6. Click ‘Next’ and complete the process of creating the custom transformation.

7. Similarly create ‘Switch Branch End’ with the below code. No prompts required and the parameters in Inputs/Outputs should be fed as shown below.

```
/*************************************************************
* Transformation - Switch Begin

*************************************************************/

                        /* End of code block */
                        %put NOTE: Execution complete for the child macro now...;
                %mend;
        %end;
        %let check node=%eval(&check node.+1);
        %put NOTE: Evaluating the branch %trim(&check_node.)...;
        %if &run node.=&check node. %then %do;
                %put NOTE: Inside the child macro and compiling macro number
%trim(&run_node.)...;
                %macro switch_node_all;
                        %put NOTE: Executing the child macro now...;
                        /* Begin of macro */
```

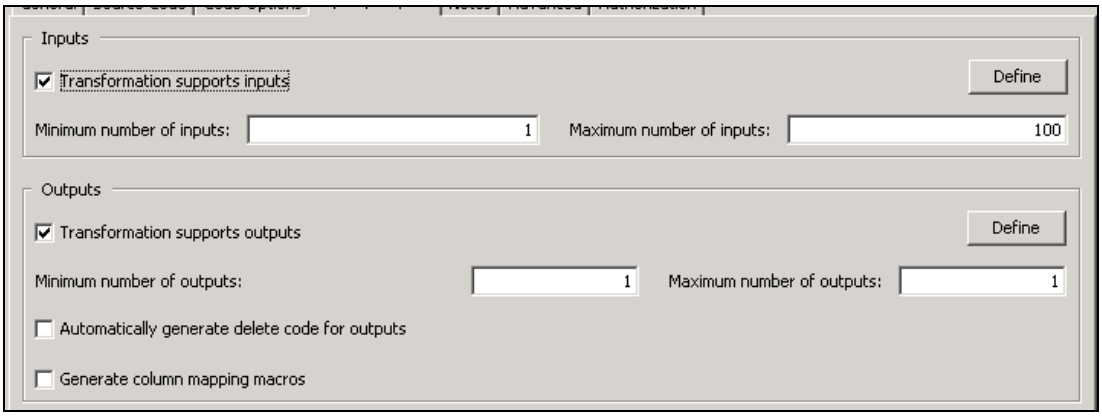


Figure 8. Case study – Understanding the 'Path' and 'Switch'

8. Similarly create 'Switch End' with the below code. No prompts required and the parameters in Inputs/Outputs should be fed as shown below.

```
/*************************************************************
* Transformation - Switch End

*************************************************************/

               /* End of macro */
               %put NOTE: Execution complete for the child macro now...;
               %mend;
       %end;
       %switch_node_all;
%mend switch_exec;

/* Execute the Switch */
%put NOTE: Executing the Switch now with Branch %sysfunc(trim(&node.))...;
%switch exec(run node=&node.);

%put NOTE: Completing the execution of SWITCH...;
```

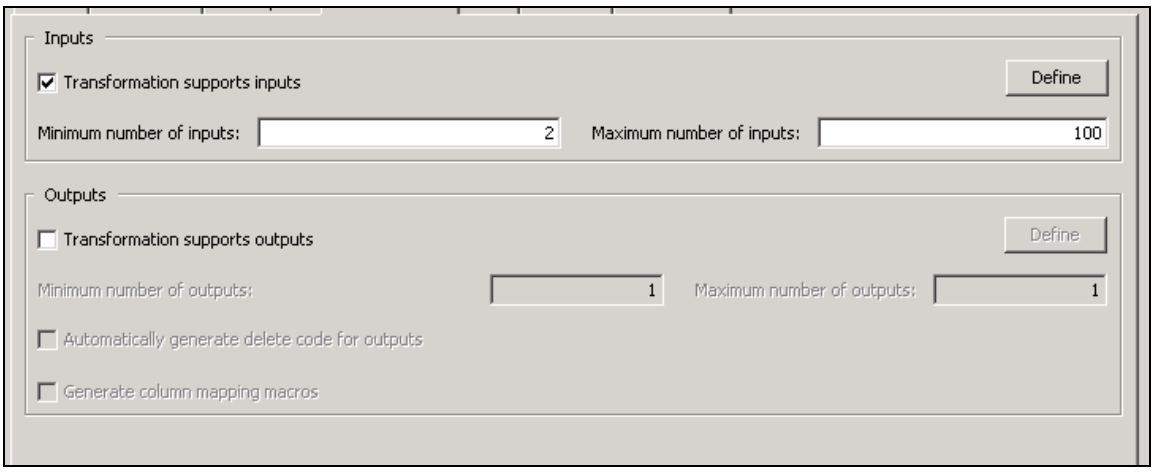


Figure 9. Inputs/Outputs

SCENARIO 1: 'JUMPING THE BRANCH'

Now that we have the right context from the case study and also have a 'SWITCH PATH' custom transformation created, let us focus to get into the real world of implementing the requirement and designing a data integration job in SAS® Data Integration Studio using 'SWITCH PATH'

The requirement as visualized before is simple and can be implemented in SAS® Data Integration Studio simple steps given below:

a. Create a new empty job with a name 'SGFExample – SWITCH PATH – Multi Branches' in SAS® Data Integration Studio.

b. Create and add a parameter 'SGF_PARAMETER' table to the job and contains only the 'day of the week' as shown below

   The values can range from 1 to 7 where a value of '1' represents MONDAY and value '7' represents SUNDAY. In the example we have chosen a value '1' to simulate the execution of branch (path) 1.

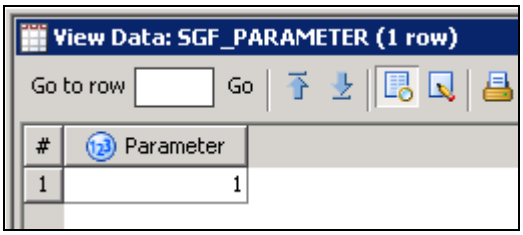


Figure 10. SGF_PARAMETER – The table content

c. Drag a 'user-written code' transformation from 'Inventory' tab to the created job and add a simple SAS code as a user written body to extract the data from the parameter table 'SGF_PARAMETER'. The node will also generate the random numbers for simulating the branch execution when we execute the job each time.

```
/* Read and roll the numbers */
data &_input;
set & input;
if parameter=7 then parameter=0;
parameter=parameter+1;
run;

/* Permanent */
proc sql noprint;
        select parameter into : input parameter from & input;
quit;

%put WARNING: The Input Parameter from the parameter table = %trim(&_input_parameter.),
so BRANCH %sysfunc(trim(&_input_parameter.)) should run ideally!;
```

d. Drag the ‘Switch Path’ transformations into the newly created job and arrange the nodes in the order shown below. Please note that the order of the nodes between ‘Switch Start’, ‘Switch Branch End’ and ‘Switch End’ is really important. They should always be in one after another like a branch.

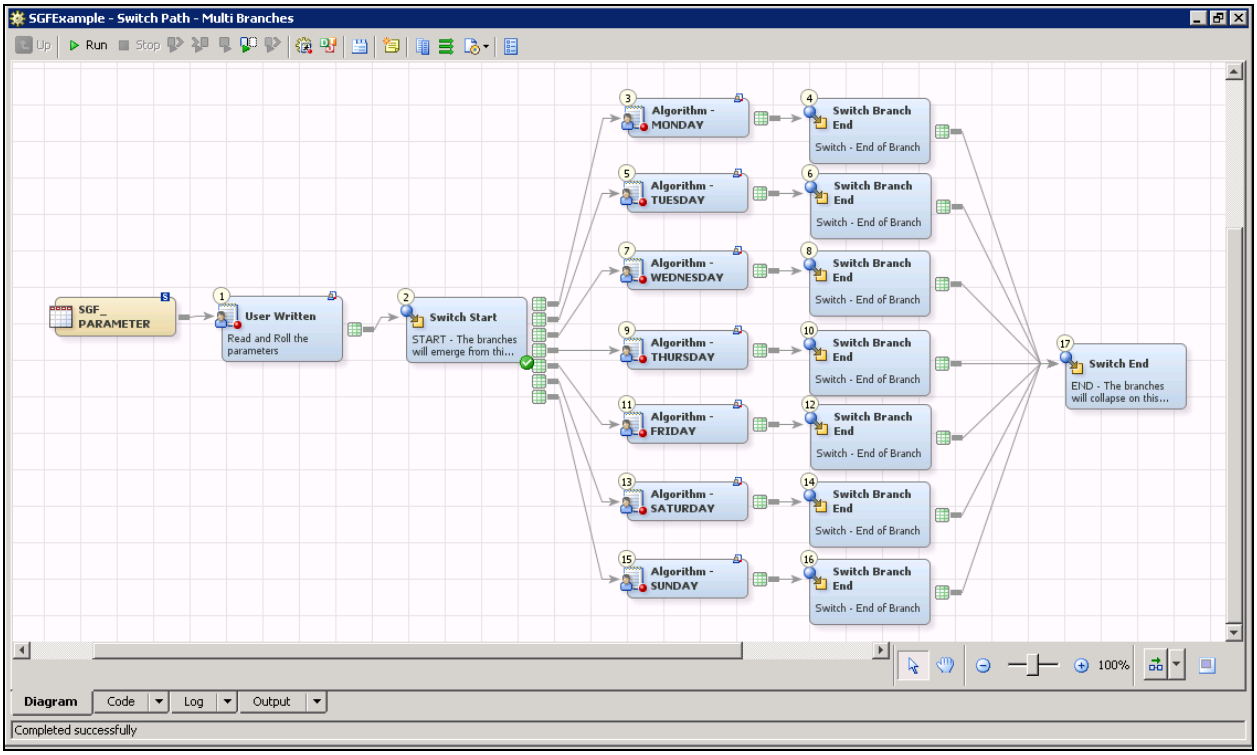


Figure 11. The job implemented using a ‘SWITCH PATH’ transformation in SAS® DI Studio

Node #3, #5, #7, #9, #11, #13, #15 represents a pseudo algorithm for each branch:

```
%put WARNING: BRANCH 1 - Weekday MONDAY - executed as parameter passed during the runtime
was &input_parameter;
%put ************************************;
%put Executing <Algorithm - Code Block>...;
%put ************************************;
```

e. Save the job. The job is ready and can be tested to simulate the conditional execution as per the requirement.

As required and designed the node #2 to node #17 forms the 7 branches that execute the specific algorithms at node #3, or #5, or #7, or #9, or #11, or #13, or #15 based on the parameter set at node #2.

The 7 branches can be visualized from above job as shown below:

Path 1 – [node #02 > node #03 > node #04 > node #17]

Path 2 – [node #02 > node #05 > node #06 > node #17]

Path 3 – [node #02 > node #07 > node #08 > node #17]

Path 4 – [node #02 > node #09 > node #10 > node #17]

Path 5 – [node #02 > node #11 > node #12 > node #17]

Path 6 – [node #02 > node #13 > node #14 > node #17]

Path 7 – [node #02 > node #15 > node #16 > node #17]

Now based on the parameter set, one of the branches will execute and serves the purpose of executing the specific algorithm on day of the week.

SOURCE CODE

```
/*************************************************************************
 * Job:             SGFExample - SWITCH PATH                             *
 *************************************************************************/
/* Get the Parameters */

/*=======================================================================*
 * Step:            Switch Start                                         *
 *=======================================================================*/
/* Takes the runtime parameter for the conditional execution */
%let node=&input parameter.;

/* The below macro variable auto-increments and checks for every node */
/* Default should be always 1 */
%let run_node=1;
%let check_node=1;

/* Begin of Switch_Start */
%macro switch exec(run node=);
       %if &run_node.=&check_node. %then %do;
              %macro switch_node_all;
                     /* End of Switch_Start */

                     /*---- Start of User Written Code ----*/
                     %put Executing <Algorithm - Code Block>...;
                     /*---- End of User Written Code ----*/

/*=======================================================================*
 * Step:            Switch Branch End                                    *
 *=======================================================================*/
                     /* End of code block */
              %mend;
       %end;
       %let check node=%eval(&check node.+1);
       %if &run_node.=&check_node. %then %do;
              %macro switch node all;
                     /* Begin of macro */

                     /*---- Start of User Written Code ----*/
                     %put Executing <Algorithm - Code Block>...;
                     /*---- End of User Written Code ----*/

/*=======================================================================*
 * Step:            Switch Branch End                                    *
 *=======================================================================*/
                     /* End of code block */
              %mend;
       %end;
       %let check node=%eval(&check node.+1);
       %if &run_node.=&check_node. %then %do;
              %macro switch_node_all;
                     /* Begin of macro */
.
/* Other Branches */
.
/*=======================================================================*
 * Step:            Switch End                                           *
 *=======================================================================*/
                     /* End of macro */
              %mend;
       %end;
       %switch_node_all;
%mend switch exec;

/* Execute the Switch */
%switch_exec(run_node=&node.);
```

NOTE - The code shown above is ‘cut-down’ version of the DI generated code for easy read and it describes how exactly the ‘SWITCH PATH’ actually works.

UNDERSTANDING THE CODE

Let us understand the code now in chunks to realize the mechanism behind the 'SWITCH PATH' technique.

As an example, we will consider only one branch (path) - Path 1.

Path 1 – [node #02 > node #03 > node #04 > node #17]

The path 'Path1' starts with node #2. The 'Switch Start' transformation node has the initialization of parameters required and the beginning of the parent macro block 'switch_exec' with a parameter 'run_node'. The parent macro block within contains the child macro 'switch_node_all' opening block as seen below.

```
/* Begin of Switch_Start */
%macro switch_exec(run_node=);
       %if &run_node.=&check_node. %then %do;
              %macro switch_node_all;
                     /* End of Switch_Start */
```

Please note that the above piece of code simply opens two macro blocks without actually closing them in the same node with a conditional 'if' statement. Let's not worry about the reason for this 'if' statement now, it will be clear in later sections.

The next in the path is node #3 and is the actual code to be executed and in the example above it is a simple user-written code with put statements.

```
                     /*---- Start of User Written Code ----*/
                     %put Executing <Algorithm - Code Block>...;
                     /*---- End of User Written Code ----*/
```

NOTE - Only a simple 'user-written code' node is used in the example, we can use any transformation or group of transformations before the 'Switch Start' and 'Switch Branch End' nodes.

The next in the path is the node #4 'Switch Branch End' transformation. This node initially closes the child macro 'switch_node_all' which has been kept open from the 'Switch Start' node.

```
/*============================================================================*
 * Step:            Switch Branch End                                         *
 *============================================================================*/
                     /* End of code block */
              %mend;
       %end;
       %let check_node=%eval(&check_node.+1);
       %if &run_node.=&check_node. %then %do;
              %macro switch node all;
                     /* Begin of macro */
```

This implicitly means that we have now encapsulated the above 'user-written' code inside the child macro when we see node #2, node #3, and node #4 together as shown below:

```
/* Begin of Switch_Start */
%macro switch_exec(run_node=);
       %if &run_node.=&check_node. %then %do;
              %macro switch_node_all;
                     /* End of Switch_Start */

                     /*---- Start of User Written Code ----*/
                     %put Executing <Algorithm - Code Block>...;
                     /*---- End of User Written Code ----*/
/*============================================================================*
 * Step:            Switch Branch End                                         *
 *============================================================================*/
                     /* End of code block */
              %mend;
       %end;
       %let check node=%eval(&check node.+1);
       %if &run node.=&check node. %then %do;
              %macro switch_node_all;
                     /* Begin of macro */
```

Additionally the ‘Switch Branch End’ node apart from closing the child macro block, it also opens up a new child macro block with same name ‘switch_node_all’ as seen above.

The interesting piece to this is – The child macro has the same name – ‘switch_node_all’ at every branch, but is always within the conditional ‘if’ statement. The above snippet gives a clear picture of this.

The next in the path is node #17 ‘Switch End’ transformation. This node as the name suggests ends the branch and closes the child macro block and the parent macro block ‘switch_node_all’ as seen below.

The node finally also executes the parent block by passing the value to the parameter ‘run_node’.

```
/*============================================================================*
 * Step:            Switch End                                                *
 *============================================================================*/
                      /* End of macro */
              %mend;
       %end;
       %switch_node_all;
%mend switch exec;

/* Execute the Switch */
%switch_exec(run_node=&node.);
```

The parameter passed is just the ‘day of the week’ and depending on the match (this is the reason for the ‘if’ statements in the code), the SAS macro engine will then compile and execute only the desired child macro within the parent macro.

This to us means that we have executed only the right branch based on the parameter provided! – ‘**SWITCH PATH**’

Now go back to page 10 to see the entire code to visualize the entire picture.

In a nutshell, ‘SWITCH PATH’ transformations enclose each branch ‘code’ as child macros with a big parent macro. The one of the child macros is then picked by the parent macro based on the parameter passed and hence serving the purpose of conditional execution.

Please note that in reality when the code goes through compilation and execution phase, only one child macro is compiled and only one child macro is executed by the parent macro. This is the reason why we have same names for the child macro.

This means that we can possibly have ‘infinite’ number of branches with this approach and it will never compile ‘infinite’ macros, will never introduce any overhead of code duplication, no overhead of redundant macro compilation etc. as only one child macro block is compiled and executed every time a parameter is passed.

The same job on execution produces the log as shown below. The parameter supplied during the run was 1 to simulate ‘Path 1’.

```
.
.
WARNING: The Input Parameter from the parameter table = 1, so BRANCH 1 should run ideally!

NOTE: Starting to execute SWITCH...
NOTE: Parameter received during the runtime is 1...
NOTE: Initialization of variables complete...
NOTE: Executing the Switch now with Branch 1...
NOTE: Inside the parent macro and about to compile the child macro...
NOTE: Evaluating the branch 1...
NOTE: Inside the child macro and compiling macro number 1...
NOTE: Evaluating the branch 2...
NOTE: Evaluating the branch 3...
NOTE: Evaluating the branch 4...
NOTE: Evaluating the branch 5...
NOTE: Evaluating the branch 6...
NOTE: Evaluating the branch 7...
NOTE: Evaluating the branch 8...
NOTE: Executing the child macro now...

WARNING: BRANCH 1 - Weekday MONDAY - executed as parameter passed during the runtime was 1
         ************************************
         Executing <Algorithm - Code Block>...
         ************************************
```

```
NOTE: Execution complete for the child macro now...
NOTE: Completing the execution of SWITCH...
.
.
```

As see above the job compiles and executes only one branch – Path 1.

This approach using 'SWITCH PATH' derives the entire benefit of existing SAS framework by using SAS® Macro Programming and SAS® Data Integration Studio to solve the problem of conditional processing in a single job.

SCENARIO 2: 'FILTERS'

There are many possible uses of the 'SWITCH PATH' and another such example is shown below which solves real-time 'filter' problem using the 'SWITCH PATH' transformation.

The problem statement is simple – *Load all the product details into the table on few days and only few product details on other days.*

Since the problem tells about 'conditional processing' – 'SWITCH_PATH' is apt and useful for solving the problem. Below is the job designed to solve this.

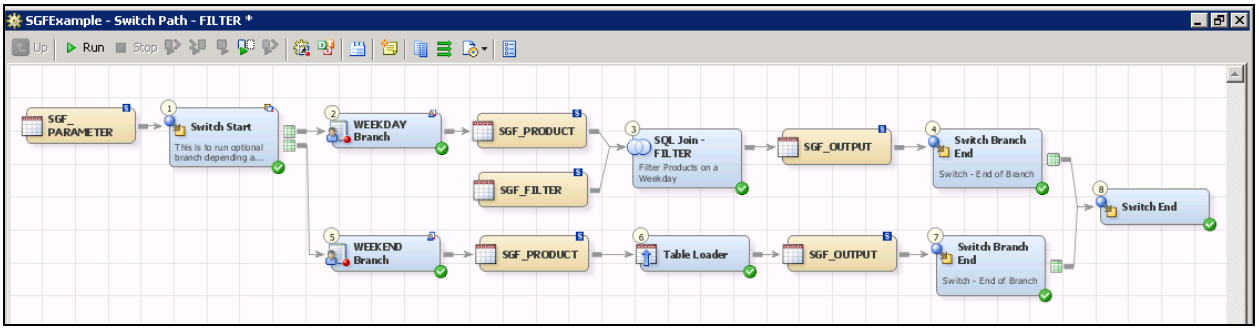
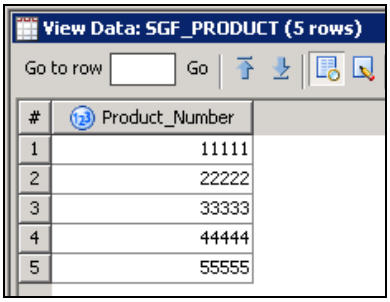


Figure 12. An example of a job with a FILTER

The above job produces a filtered dataset when 'Path 1' executes which is node #1, node #2, node #3, node #4 and node #8. When 'Path 2' executes the output is unfiltered as shown below.

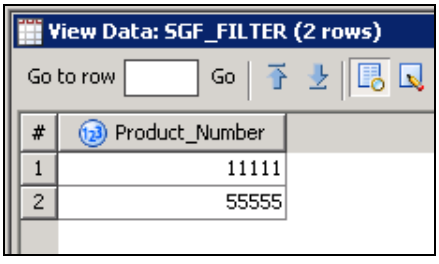
The Source data – SGF_PRODUCT table contains 5 rows representing the dummy product numbers.



#	Product_Number
1	11111
2	22222
3	33333
4	44444
5	55555

Figure 13. Input dataset SGF_PRODUCT

Filter data – SGF_FILTER contains only 2 rows representing the data that needs to be filtered as said in the requirement.



#	Product_Number
1	11111
2	55555

Figure 14. Filter dataset SGF_FILTER

When ‘Path 1’ is executed by passing the parameter ‘1’. The data from SGF_PRODUCT is filtered to have only those rows which are present in SGF_FILTER and hence SGF_OUTPUT has only 2 rows.

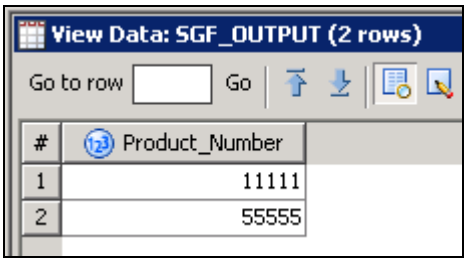


Figure 15. An example of a job with FILTER

When ‘Path 2’ is executed now by passing the parameter ‘2’ as the execution takes node #1, node #5, node #6, node #7 and node #8. The data from SGF_PRODUCT is not filtered and directly loaded from the source data and hence SGF_OUTPUT has all the 5 rows as shown below.

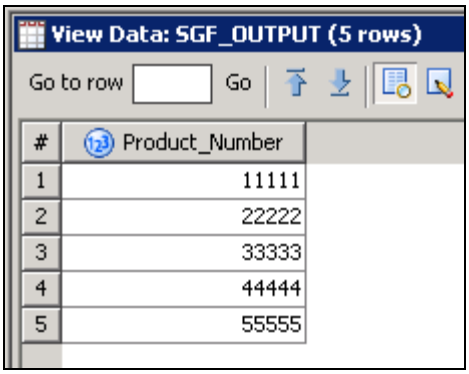


Figure 16. An example of a job with FILTER

By using ‘SWITCH PATH’ transformations we can load all data or filtered data based on the parameter passed during the runtime with just one job.

The above example highlights the real and classic benefit of ‘SWITCH PATH’ with merely no restrictions on number of branches and nodes.

BENEFITS

- Support ‘any’ number of conditional branch executions.
- Easy to use with ‘drag and drop’ feature.
- Easy to create one.
- Completely customizable.
- Easy metadata management.
- No code redundancy.
- No multiple macro compilation and execution.
- No maintenance overhead
- Easy to upgrade and add features.
- Self-documenting like other components in SAS® DI Studio.
- Supports Export/Import.

CONCLUSION

The addition of 'SWITCH PATH' has expanded our options and has opened up a new way to visualize the conditional execution of branch/nodes in SAS® Data Integration Studio. In particular, the 'SWITCH PATH' custom transformation now allows us to implement infinite number of branches without having to worry about maintenance and performance.

Being able to conditionally execute/switch a branch within a job opens a huge opportunity to merge multiple logical steps efficiently into a single job.

The 'SWITCH PATH' technique presented in this paper takes advantage of these innovations to completely eliminate the previous limitations in SAS® Data Integration Studio. As the examples illustrate, the technique can serve many purposes in day to day business needs for a data integration developer as it is completely reusable.

DISCLAIMER: The contents of this paper are the work of the author and all of the steps explained in this paper works as expected on SAS 9.3 with SAS® Data Integration Studio 4.6.

REFERENCES

SAS® Data Integration Studio – Creating and Using a Generated Transformation, Jeff Dyson, Financial Risk Group, Cary, NC

SAS® Data Integration Studio 4.6 UserGuide – SAS Publishing

SAS® Macro Design Patterns - Mark Tabladillo Ph.D., MarkTab Consulting, Atlanta, Associate Faculty, University of Phoenix

ACKNOWLEDGEMENTS

Thanks to SAS Institute colleague James Marshall D, both for his encouragement to pen this paper and for the perceptive reviews of the proposal and the final paper.

Thanks to all colleagues and my managers in Tesco HSC and UK for all the support.

Special thanks to Gregory Nelson, for being my SAS Global Forum 2014 mentor and for his thorough reviews.

Finally thanks to Sowmya, my wife and my parents, for supporting my work.

CONTACT INFORMATION

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

Prajwal Shetty Deviprasad
Tesco Hindustan Service Centre
#80, 81, EPIP Area Whitefield Bangalore, India – 560066
Email: Prajwal.Deviprasad@in.tesco.com
neoprajwal@gmail.com
Mobile: +919901724056 (India)