Paper 097-2012

Dynamic Conditional Processing In SAS® Data Integration Studio

Riku Jokinen, Solution Specialist, Aureolis Oy

ABSTRACT

Data driven conditional processes in SAS® has always been possible with macros. With SAS Data Integration Studio, you might find macro based solutions either unwanted due to most of the code not being visualized in the job, or impossible due to metadata not being able to follow the actual data. The Loop transformation can be used to dynamically conditionalize SAS code without using user written macros. This enables data driven DI Studio jobs that will perform certain parts of the code only if defined conditions apply. It also allows separate parts of the code to be run either iteratively or conditionally while maintaining metadata intact throughout the job.

INTRODUCTION

Often SAS programmers face a situation where their code must decide what to do next based on conditions that depend on data. These types of dynamic conditional processes are relatively straightforward in traditional Base SAS programming. In SAS Data Integration Studio such simple solutions are usually possible, but not recommended, as we will see in the following examples.

The reader should have some experience on using the Loop transformation in SAS Data Integration Studio to apply the ideas presented in the paper.

DYNAMIC CONDITIONAL PROCESS IN BASE SAS® PROGRAM

A telecommunications company constantly creates files which contains two different types of data. You can only tell which type after you've read the first line of the file. Files contain code “1”, if the following lines are for customers' text messages and “2” if they are for phone calls. A revenue assurance solution built on Base SAS code can easily read this type of code and decide what type of code to use to interpret the data into SAS table.

A simple solution would be to start by reading in the first line of data from every file. We then select the previously mentioned type code into SAS variable “type_code”. We can now create two different macros for reading the two types of data. Now we can simply call either one of these macros based on the macro variable “type_code”. Alternatively you could read the data with a single Infile statement, and create two alternative processes depending on the type_code within the same data step.

DYNAMIC CONDITIONAL PROCESS IN DI STUDIO JOB

Similar solution is naturally possible in Data Integration Studio. We start by using the "File Reader" transformation and an extract transformation "Select first row and type_code" to read the first line of data from the file, as shown in Figure 1. This is followed by a User Written transformation with added output table containing the exact code created in the Base SAS example. This transformation has two output tables for which the metadata is created according to the actual data depending of the file type. Using just the file reader transformation with "User written" option is not possible in metadata environment, since it allows only one output table.

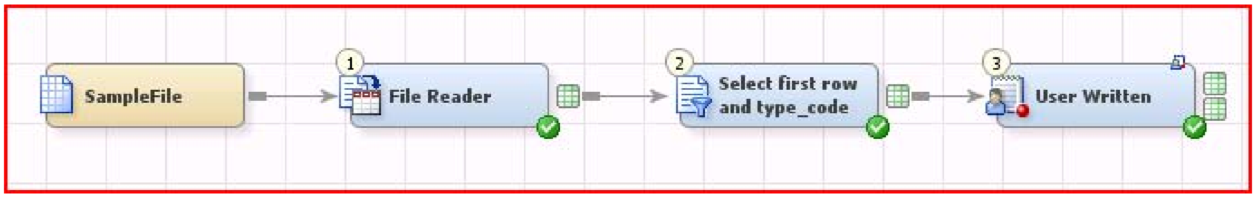

Figure 1: Using user written code to do read the same file into two different output tables

When this job is run, only one of the output tables would be populated. If this part should be followed by more data processing, we would have to make sure at least an empty table would be created for both types of outputs. But even then one half of the code would always be run unnecessarily, and could create further problems further down the data process job. With this solution, only sensible ways to avoid uncontrolled problems would be to transform all the code that follows the first User Written transformation into it. It is possible to create a temporary data layer which further jobs would call with an include statement. This isn't always possible however, and could complicate an otherwise simple process unnecessarily.

The more complex the data process at hand, the more complex this one particular User Written transformation becomes. In effect it would create a complete Base SAS program within a DI Studio environment, a solution best avoided if possible. Not only is such solution inelegant, but it is very difficult to maintain and develop. A key aspect of DI Studio is its ability to visualize the data process flow. A well commented DI Studio job is not only easy for even a newcomer to the solution maintain, but works as documentation as well. The job created above is neither. Another downside for such job would be the loss of metadata connection between input and output of the data within the job. This would prevent among other things from using the Analyze feature over the job.

USING LOOP TRANSFORMATION TO CREATE A DYNAMIC PROCESS

Consider a situation, where a table LOG tells whether the table SOURCE contains information we need to include in our data process. We simply read the LOG table for such information in the "Read log" Extract transformation, as shown in Figure 2. The result must contain a single row of data, if we are to include table SOURCE in our job, and no data at all, if not. The table TARGET is updated according to this. Regardless of whether the conditional process was run or not, the job continues from "Continue the process…" Extract transformation.

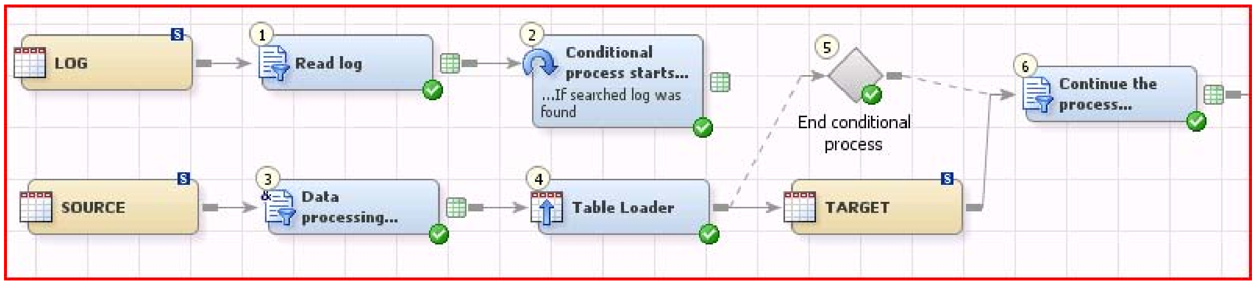

Figure 2: Using Loop transformation to create "run once or not at all" conditional process

The idea behind this solution is that Loop transformation is run according to how many rows of data is in the table it reads for its looping parameter. In this example it is the target table of the "Read log" transformation. It doesn't matter which variable is used, since the number of loops run is defined by the number of rows in the data, not by its contents. Thus, if there is no data at all, the process inside the loop is skipped. If there is just one row, then it is run once.

To demonstrate a more complex situation, we go back to our Base SAS example. A screenshot of the process is shown below in Image 3. The transformations described are in brackets. We start by putting an Extract transformation after the File Reader transformation (1 & 2). The output table for it contains just one row and at least one variable, the type_code. We then split the data into two parts with Splitter transformation (3) according to type_code. Now one of these tables should always contain a value of the type_code, while the other one is empty.

We then create a parallel processes for both types of data. The Loop transformation (4 & 8) is followed by a File Reader transformation (5 & 9), which is now used to read the whole file. This transformation must include loop parameters as well, which are used to determine whether the loop is run or not. At this point the data is now available for further processing, and can differ from other parallel processes in the same job. Finally we close both loops with the Loop End transformation (7 & 11).

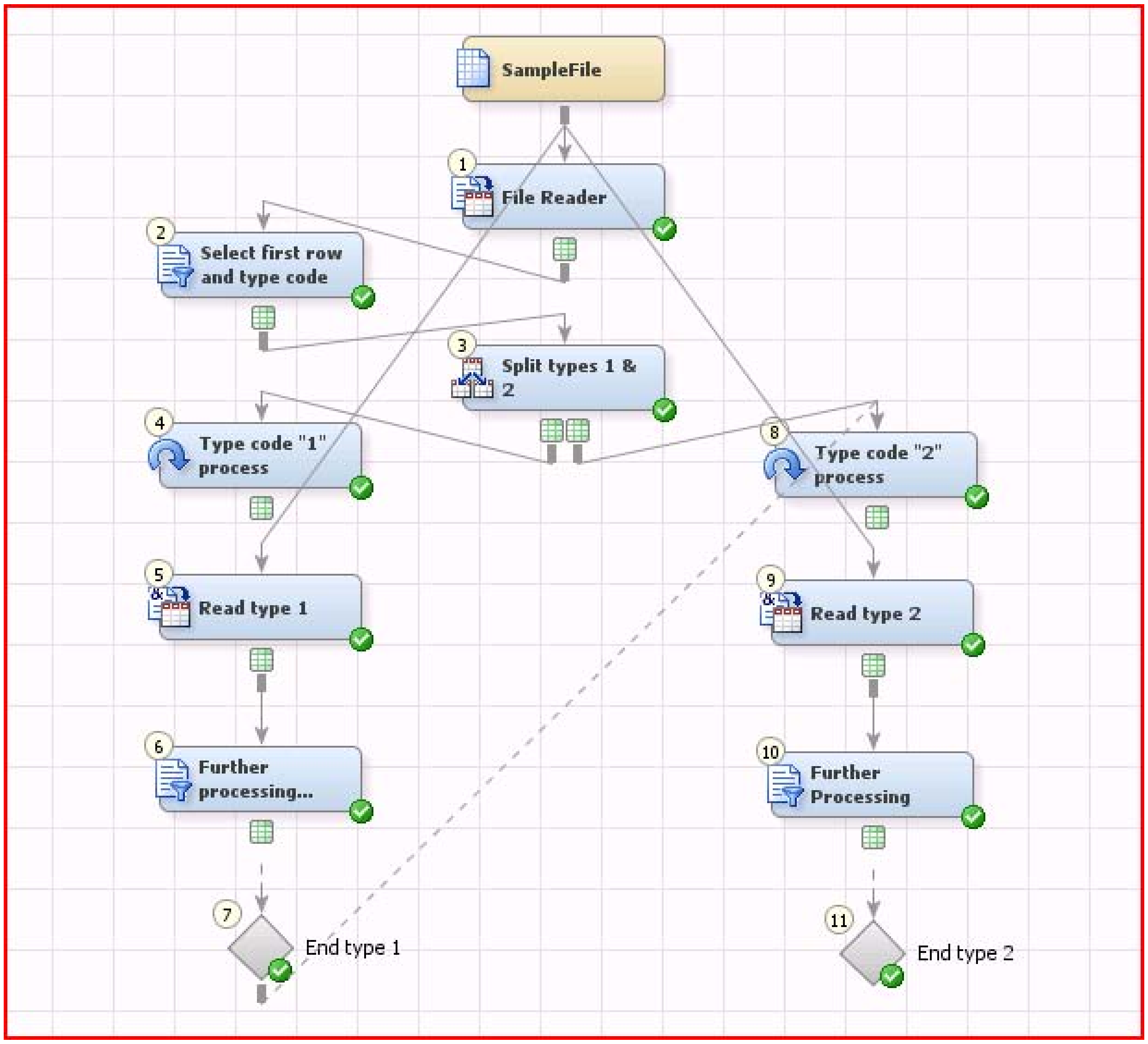

Figure 3: Parallel processes in Data Integration Studio

While this example contains just two parallel conditional processes, there is no limit to their number. As our first example demonstrates, you can create a simple "run once or not at all" process as well. Often you might want to read the processed data up until a certain point, and find out if certain requirements within the data apply before continuing with the process. Then the job would either run entirely or stop before the loop, depending on the conditions set by the programmer. In such situation it might be useful to use another parallel process as an error handling arm.

CONCLUSION

With these types of jobs it is easy to maintain metadata and improve usability of the Data Integration Studio jobs. The ability to use the Analyze feature in DI Studio is a significant improvement over traditional Base SAS programming in regards of the maintenance and development of ETL and other processes. It also secures that metadata matches the physical data. Therefore this type of process is highly recommended over user written macro programming when doing conditional processes.

ACKNOWLEDGMENTS

Special thanks to Jouni Javanainen from Fennia Mutual Insurance Company for his contribution and comments.

CONTACT INFORMATION

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

Riku Jokinen
Aureolis Oy
Panorama Tower, Hevosenkenkä 3
02600 Espoo, Finland
riku.jokinen@aureolis.com