Paper 1179-2017

# SAS® Data Integration Studio – Take Control with Conditional & Looping Transformations

Harry Droogendyk, Stratia Consulting Inc.

## ABSTRACT

SAS Data Integration Studio jobs are not always linear. While Loop transformations have been part of DI Studio for ages, only more recently has SAS Data Integration Studio included the Conditional Control transformations to control logic flow within a job. This paper will demonstrate the use of both the Loop and Conditional transformations in a real world example.

.

## INTRODUCTION

The SAS Data Integration Studio ( DIS ) graphical user-interface allows the creation of complex processes without a great deal of typing. Built-in "transformations" are included to do most common SAS data processing tasks. Once dragged onto the palette and connected to data objects, or to each other, the code generated by the transformations is customized by specifying options through the GUI. When desired functionality isn't available through the standard transformations, users can create user-defined custom transformations or user-written code nodes to include custom code directly in the job.

Until the most recent releases of DIS, it was only possible to conditionally execute portions of a job by using macro code in a user-written code node or by co-opting the loop transformation to generate zero or one loop iterations.

Why bother with the rigmarole of building jobs using DIS and its transformations? Isn't it much easier to bang the code out yourself? At times it would be simpler to code jobs by hand, but doing so would preclude one of the significant advantages of using DIS. When DIS is used to create data processing and reporting jobs a very useful metadata "trail" is created. Once the job and the associated data structures are saved in the metadata, it's much easier to assess the impact of any potential changes to the job or data structures using the "Analyze" functionality of DIS. Additional metadata reporting and insight can be generated using the results returned from other metadata queries via tools like PROC METALIB.

This paper will walk through the process to create a real-world job that utilizes the Loop and Conditional Control transformations. Not all possible options of the Loop/Loop End and Condition Start/End transformations will be covered in this paper.

## BUSINESS NEED

Daily text files are received from a third-party vendor, imported into SAS and loaded into a Teradata database. Since the delivery of the text files is not as consistent as we would like, a control table is used to maintain the date of the last text file processed. Each day the scheduled job reads the control table, extracts the last processed date and attempts to load files for each day between the day following the last processed date to today's date, date1 - dateN. Since files must be processed in date order, when a missing file is encountered, the job must stop importing data. However, even if we detect a missing file, we do want to continue checking for the presence of each date's file until dateN is reached so we can report on all the missing files.

In a typical looping scenario, two jobs are created:

1. The inner job is the one to be looped over. The inner job is defined with at least one parameter input which is used to determine the date to process.
2. The outer job sets the parameter inputs, defines the loop and includes the inner job within the Loop and Loop End transformations to cycle over the inner job N times.

In the business situation described above, the outer job reads the control table and creates a SAS dataset of the dates between date1 and dateN. This dataset is used as input into the Loop transformation which creates the parameters required by the inner job – in this case the parameter is a SAS date value. The inner job accepts the parameter and looks for a file for the date value received. If the file is present, it will be imported and loaded into Teradata. If it is not present, an email is generated and the import / loading code is skipped.

## DIS JOB DIAGRAMS

The three figures below illustrate the DIS jobs to be discussed in the remainder of the paper.

The "&" on the Inner_Parameterized_Job icon indicates that the job has a parameter. Since the Inner_Paramaterize_Job is included in the Outer_Loop_Job flow, only the Outer_Loop_Job is deployed, hence the blue arrow in the icon.

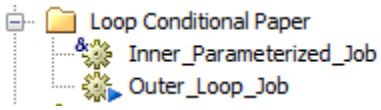



**Figure 1. DIS jobs in folder structure**

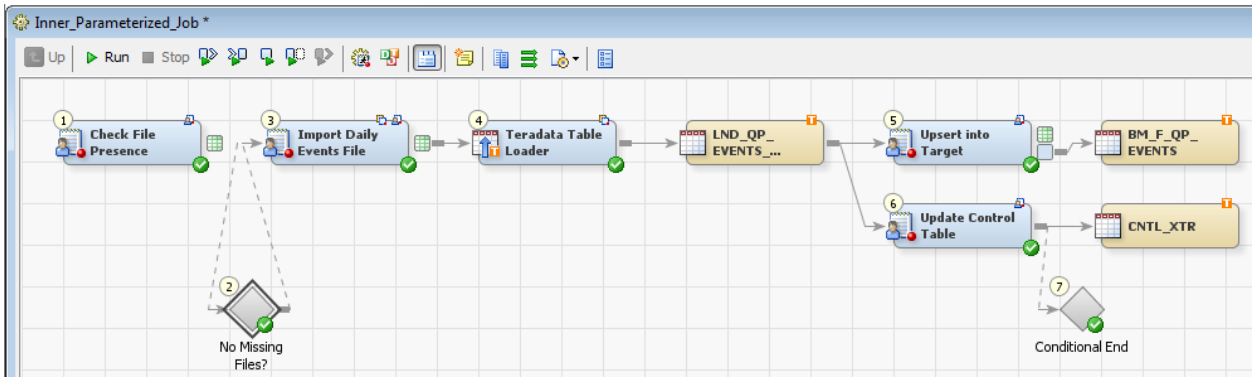



**Figure 2. Inner parameterized job details**

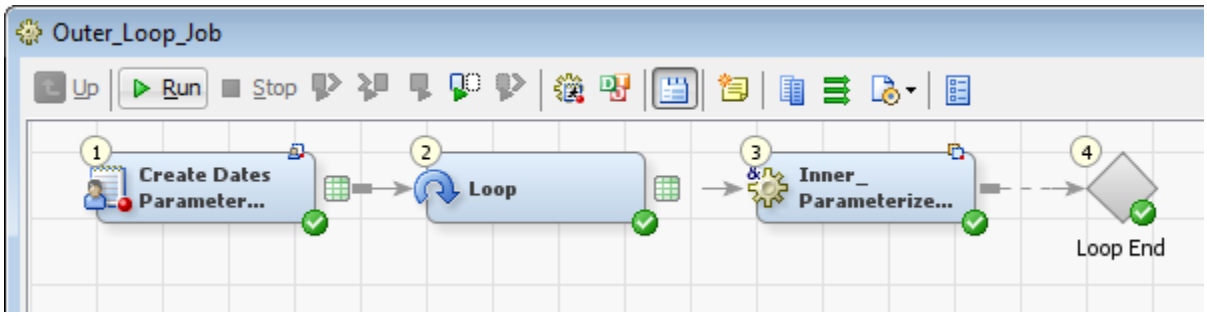



**Figure 3. Outer looping job details**

## CREATING THE INNER LOOPED JOB

As seen in Figure 2, Inner_Parameterized_Job contains a number of transformations and metadata table entries, much like you'd see in any job. The two items of interest for the purposes this paper are the definition of the job parameter and the presence of the Conditional Start / End transformations.

### DEFINING THE JOB PARAMETER

The outer job will read the control table to determine the dates to be processed, create a dataset of all dates required and loop over those dates, executing the inner job for each date found. For each iteration of the loop, the outer job must pass the date value for that iteration to the inner job so the inner job knows what to process. DIS uses parameters, or prompts, to pass the values required. There's nothing mysterious about these parameters, they're really just macro variables and use the same mechanism employed by prompted EG processes or Stored Process prompts. See http://www.stratia.ca/papers/stp_cascading_dynamic_prompts.pdf for an example of STP prompts.

Job parameters are defined in the Parameters tab of the Job Properties. Click New Prompt:

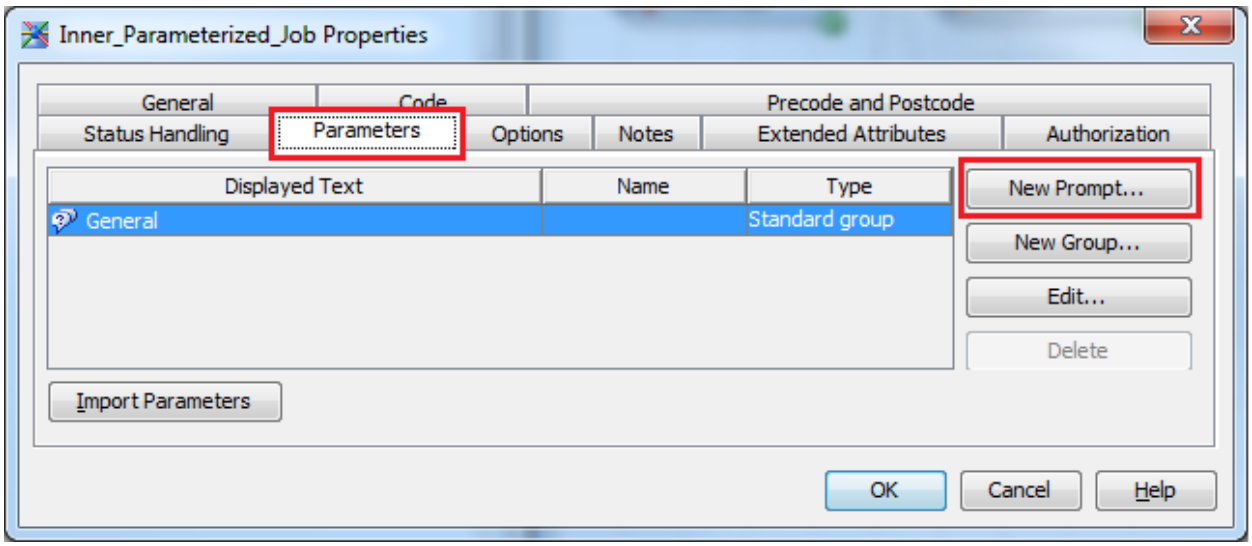



**Figure 4. Job Properties**

Enter the name of the prompt in the General tab. The name must be identical to the name of the macro variable the code will use when it is referencing the date it is processing. In this case, the code will reference **&file_dt**. Since internal SAS date values will be passed in, specify a Prompt Type of Numeric in the Prompt Type and Values tab. Click OK. The Parameters tab will display the prompt particulars as seen in Figure 6.

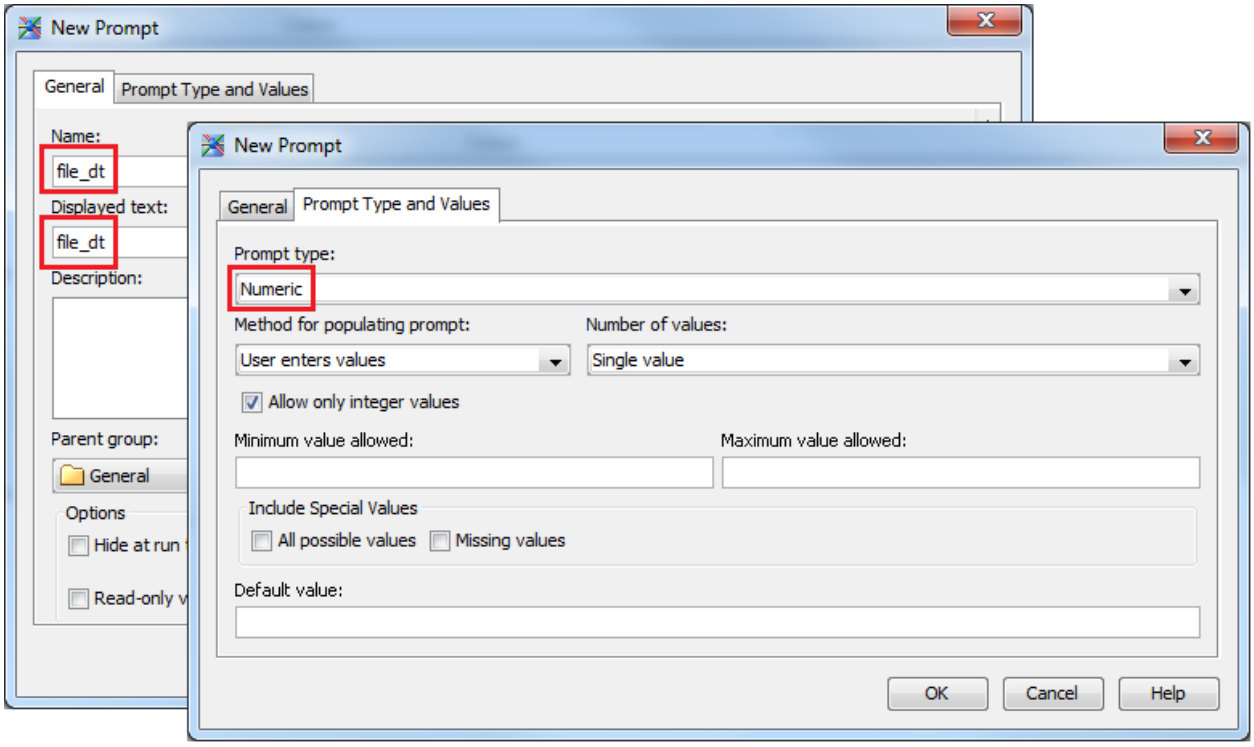



**Figure 5. Job Properties – defining a parameter**

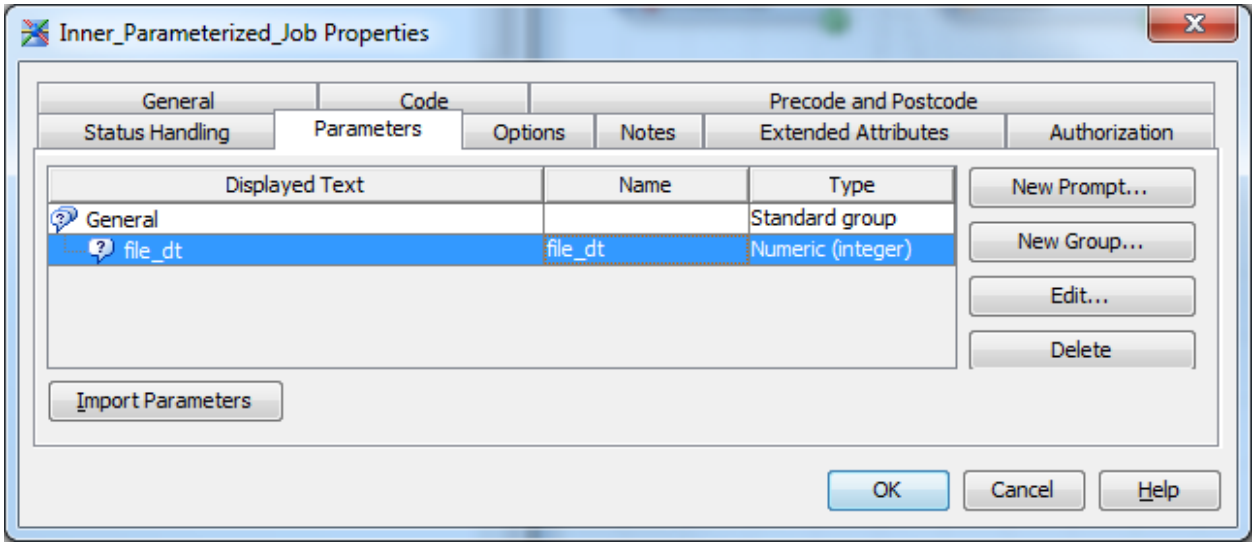



**Figure 6. Parameter defined**

#### USING THE JOB PARAMETER

The first transformation in the inner job shown in Figure 2 is a user-written code node which utilizes the **&file_dt** macro variable containing the parameter value. The **&file_dt** value is substituted in the filename macro variable to create the filename pattern. Via the PIPE option of the FILENAME statement, the Unix verb **ls** is used to list the files in &fileloc that satisfy the &filename pattern.

The results of the **ls** command are read into the data step via the INFILE / INPUT statements. If the file is found, an observation is written to the SAS data set **_files_to_process**. When the data step has consumed the results of the Unix list files command, the macro ***%check_files*** is invoked to determine the next steps.

```
%let fileloc    = /data/sharedall/mifeeds/quickplay;
%let filename   = QuickPlay_BELL-DAILY-Events-TV5-TEST-%sysfunc(putn(&file_dt, yymmddn8.))*.csv;

%put Looking for filename: &filename;

filename ls pipe "ls &fileloc./&filename";
data _files_to_process;
    length path_file $256;
    infile ls;
    input;
    put _infile_;

    * if the asterisk is found in the ls output, it means the file was not found ;
    if index(_infile_,'*') = 0 then do;
        path_file = _infile_;
        output;
    end;
run;

%check_files
```

**Figure 7. Using the &file_dt parameter value**

As seen in Figure 8, ***%check_files*** uses a utility macro **%attrn** ( see Appendix ) to obtain the number of observations in **_files_to_process**, the SAS data set populated with the successful results of the Unix list files command. If **&files_found** = 0, an email is sent to interested parties to inform them of the missing file.

The criteria set out in the **Business Need** section required that if a file was missing, the process should continue to check any remaining dates, but no subsequent files should be imported even if they are found. To support this requirement , the macro variable **&only_check_files** is set to **Y** when a file is found to be missing and *never reinitialized*. **&only_check_files** will be used in the Conditional transformation to determine if the file import and database table load in the rest of the job should execute.

```
%macro check_files;

    %global files_found
            only_check_files;

    %let files_found = %attrn(_files_to_process,nobs);

    %if &files_found = 0 %then %do;

        %email(
           Report_Key    = Quickplay CSV Issue,
           email_from    = %str(t                    ),
           Subject       = Missing data file &filename,
           greeting      = %str(Quickplay folks:),
           message       = %bquote(Please (re)send &filename., not found on Bell servers.),
           bye           = %bquote(Regards,`Bell Client Insights` `&etls_jobname)
            );

        /*
            The looping process is still dependent on files with consecutive dates being processed.  If a
            file is missing, going back to process it is a manual process, so set a switch so all we do
            from here on is check for the presence of files.
        */

        %let only_check_files = Y;

    %end;

%mend;
```

**Figure 8. %check_files macro**

#### SETTING UP CONDITIONAL PROCESSING

From the Transformations tab, drag the Conditional Start and Conditional End transformations to the palette. Using the up/down arrows in the Control Flow tab in the Details section, position the Conditional Start transformation immediately before the first transform to be conditionally executed and the Conditional End transformation immediately after the last transform to be conditional executed.

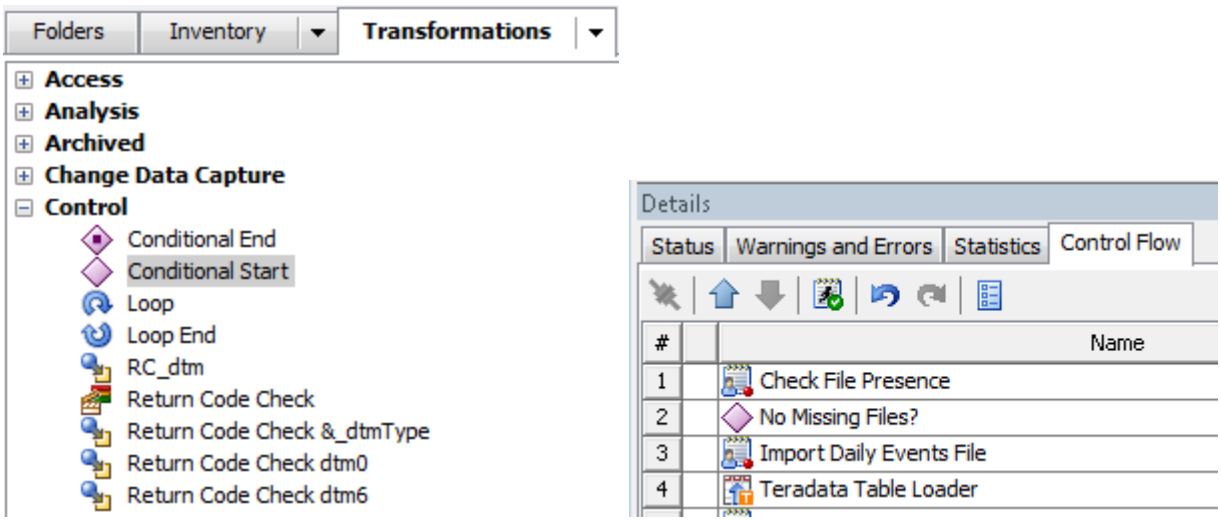



**Figure 9. Select Conditional Start and position it within the flow**

Double click the Conditional Start transformation to access its properties. Provide a meaningful name and set the condition. Note the condition specifies the **&only_check_files** macro variable created in the first code node. If the

condition is *false*, the job logic will jump to the Conditional End transformation, skipping all steps between the Conditional Start / End transformations.

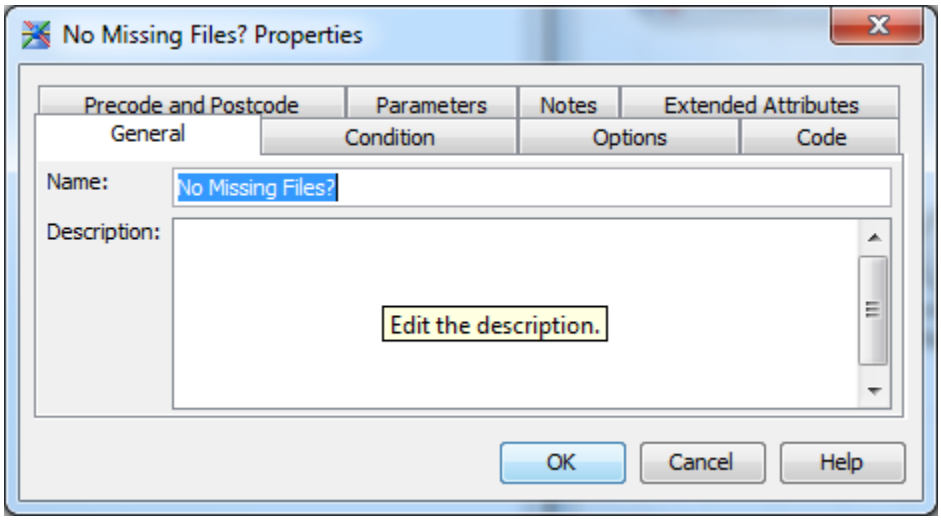



**Figure 10. Name the condition**

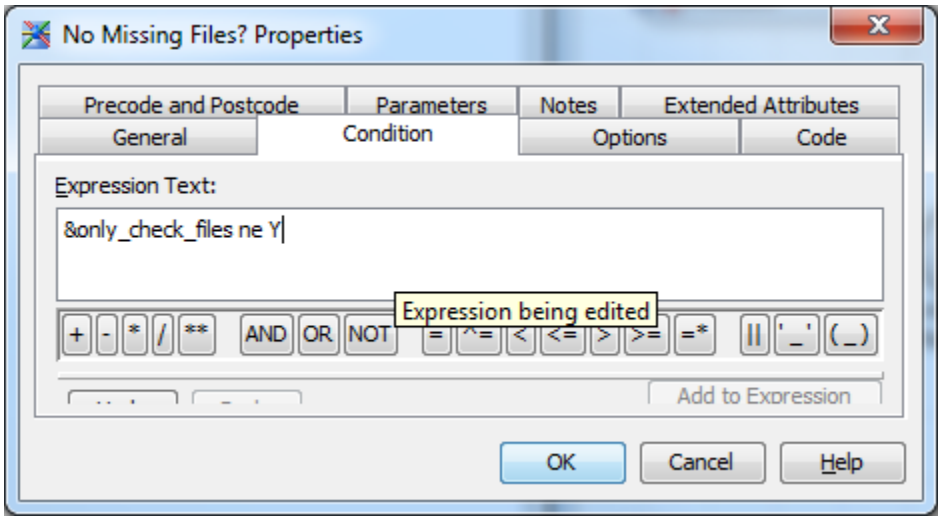
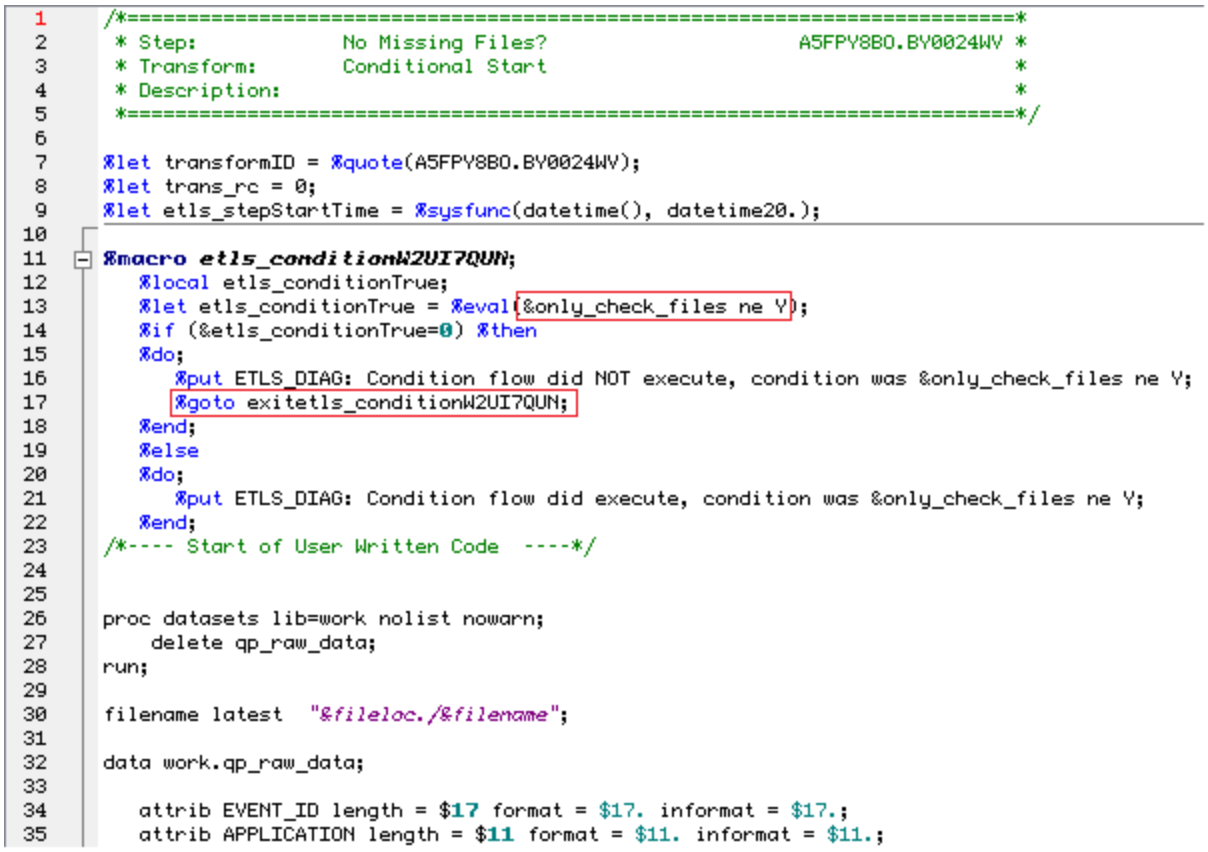



**Figure 11. Specify the condition**

Once the Conditional Start transformation properties have been set, click the Code tab to see the *generated* code, it may be different than expected!

Lines 1-22 in Figure 12 are the Conditional Start code. Note the condition specified in the properties appears within an %eval() function. The %eval function will ensure a value of 1 or 0 is generated, true or false. If the condition evaluates to 0 ( false ), a macro **%goto** is executed to transfer control to the label **exitetls_conditionW2UI7QUN**.

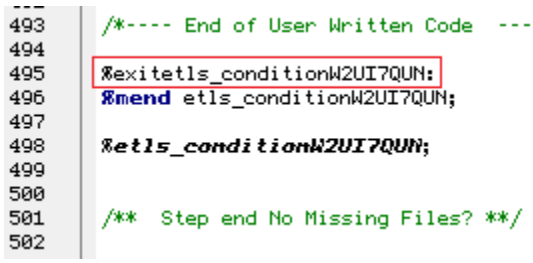
Code found in lines 23 and following are the code from the job transformations between the Conditional Start and Conditional End transformations. DIS essentially "copies" the code from the enclosed transformations when generating the Conditional Start code.



```
1  /*=============================================================================*
2   * Step:            No Missing Files?                  A5FPY8BO.BY0024WV *
3   * Transform:       Conditional Start                                   *
4   * Description:                                                         *
5   *=============================================================================*/
6
7  %let transformID = %quote(A5FPY8BO.BY0024WV);
8  %let trans_rc = 0;
9  %let etls_stepStartTime = %sysfunc(datetime(), datetime20.);
10
11 %macro etls_conditionW2UI7QUN;
12    %local etls_conditionTrue;
13    %let etls_conditionTrue = %eval(&only_check_files ne Y);
14    %if (&etls_conditionTrue=0) %then
15    %do;
16       %put ETLS_DIAG: Condition flow did NOT execute, condition was &only_check_files ne Y;
17       %goto exitetls_conditionW2UI7QUN;
18    %end;
19    %else
20    %do;
21       %put ETLS_DIAG: Condition flow did execute, condition was &only_check_files ne Y;
22    %end;
23 /*---- Start of User Written Code  ----*/
24
25
26 proc datasets lib=work nolist nowarn;
27     delete qp_raw_data;
28 run;
29
30 filename latest  "&fileloc./&filename";
31
32 data work.qp_raw_data;
33
34    attrib EVENT_ID length = $17 format = $17. informat = $17.;
35    attrib APPLICATION length = $11 format = $11. informat = $11.;
```

**Figure 12. Conditional Start generated code**

After scrolling to the end of the Conditional Start generated code, the **%goto** label is found. If the condition if false, all the code from lines 23 to 494 will be skipped.



```
493 /*---- End of User Written Code  ----*/
494
495 %exitetls_conditionW2UI7QUN:
496 %mend etls_conditionW2UI7QUN;
497
498 %etls_conditionW2UI7QUN;
499
500
501 /**  Step end No Missing Files? **/
502
```

**Figure 13. End of Conditional Start generated code with %goto label**

Save and exit the inner parameterized job.

## CREATING THE OUTER LOOPING JOB

The outer looping job accesses the control table to determine the last file date successfully imported and loaded. A data step **DO** loop populates a SAS dataset for each day from that date until today. The Loop transformation consumes the SAS dataset to generate parameter values and invokes the inner parameterized job.

Create a new job and perform the following steps:

#### USER-WRITTEN CODE NODE TO CREATE DATES TABLE

Drag a user-written code node to the palette and double click to access its properties. Change Code Generation Mode to “All user written”. Enter the code to read the control table and generate the data set with the required date values. Note the data set and column names.

```
/*  CNTL_XTR records the last successful run, bump it by one.  */

%let _xtrFromDTM = %sysfunc(datetime(),datetime19.);

proc sql noprint;
    select max(xtrToDTM)
      into :_xtrFromDTM
      from mi_cntl.cntl_xtr;
quit;

data qp_dts;

    do file_dt = datepart("&_xtrFromDTM"dt ) + 1 to today() ;
       output;
    end;

run;
```

**Figure 14. Generate looping date values**

Right click the green grid ( denotes work data set created ) on the right end of the user-written code node, select Properties.

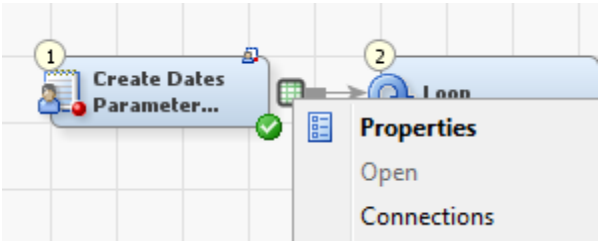



**Figure 15. Accessing properties of user-written work table**

In the Physical Storage tab, give the work table the same name as specified in the code, **qp_dts**.

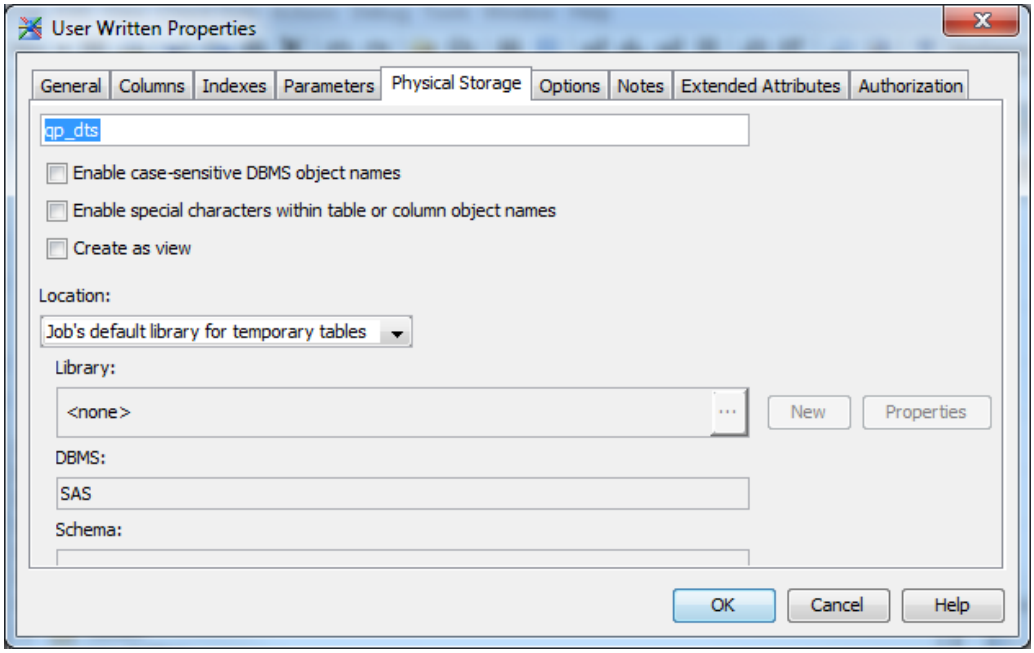



**Figure 16. Name the user-written work table**

Click the Columns tab and define the **file_dt** column created in the user-written data step code.

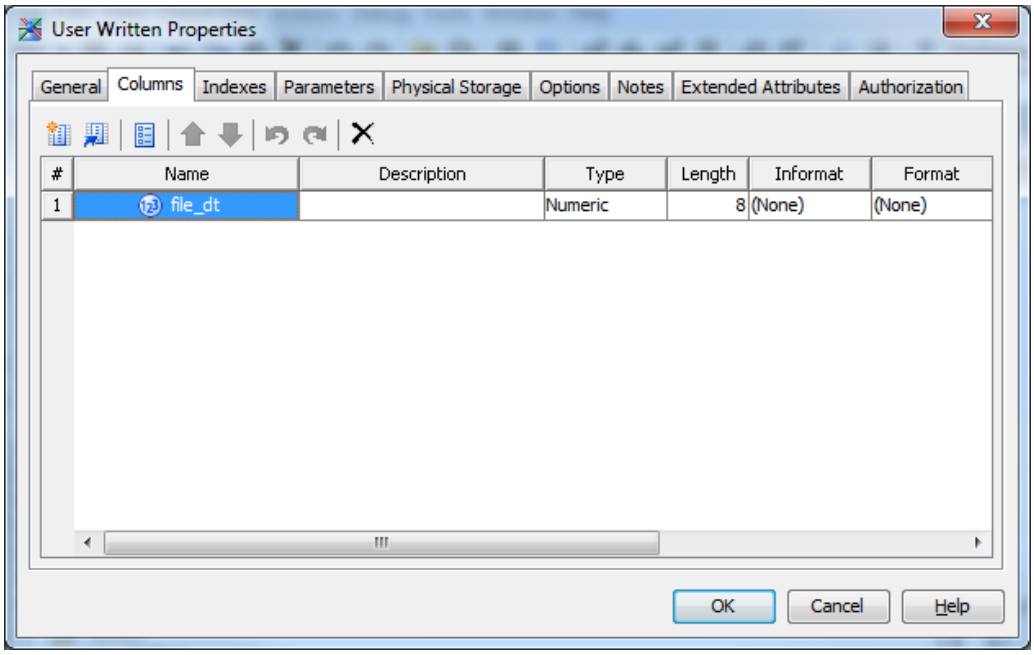



**Figure 16. Define user-written work table column(s)**

**CREATE THE LOOP – ORDER IS IMPORTANT !**

1. Drag the Loop transformation to the palette from the Control section in the Transformations tab
2. From the folders tab, drag the newly completed Inner_Parameterized_Job to the palette, connecting it to the Loop transformation. This step may take a few seconds to complete.
3. Drag the Loop End transformation from the Control section, connecting it to the Inner_Parametized_Job.
4. Double click the Loop transformation to access its properties, click the Parameter Mapping tab shown in Figure 17. The Loop transformation is connected to both the user-written code node and the Inner_Parameterized_Job. By virtue of those connections, the Loop transformation knows the:
    a. parameter name / macro variable required by Inner_Parameterized_Job and displays it
    b. data set variables available from the user-written work table
5. Select the **file_dt** column in the Mapped Source Column drop down
6. Click the Code tab in the Loop transformation, scroll down several screens and shudder – aren't you happy you don't have to debug that ?!
7. Click OK
8. Save the outer job and deploy. When the outer job is deployed, all of the generated SAS code from the inner job will also be included in the .sas file that results.

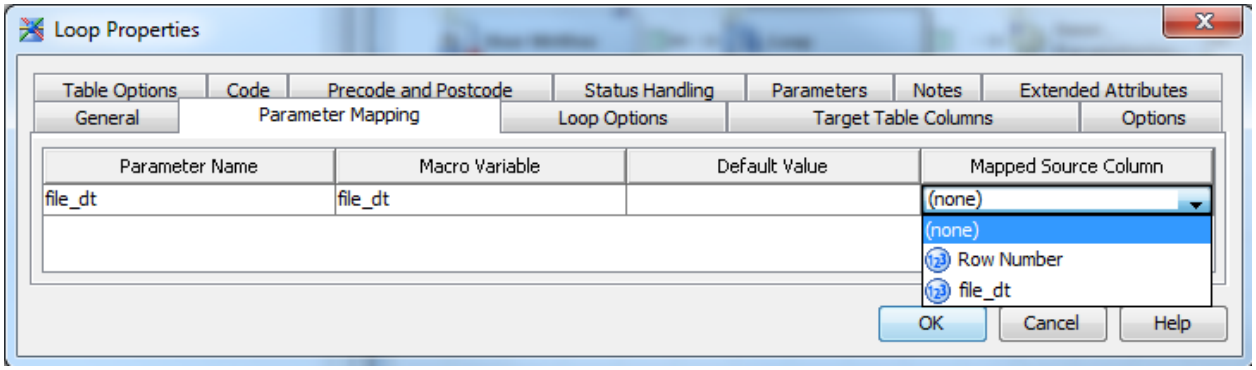



**Figure 17. Loop transformation properties**

## UNEXPECTED BEHAVIOUR

Do you recall what the generated code looked like in the Conditional Start transformation? In addition to the conditional code, all the code generated by the transformations between the Conditional Start and End transformations was also included. For that reason, when the job is executed within DIS, the transformations between the Conditional Start and End will never turn green to show they're executing. Instead, the Conditional Start icon will stay green until *all* the conditional code is completed.

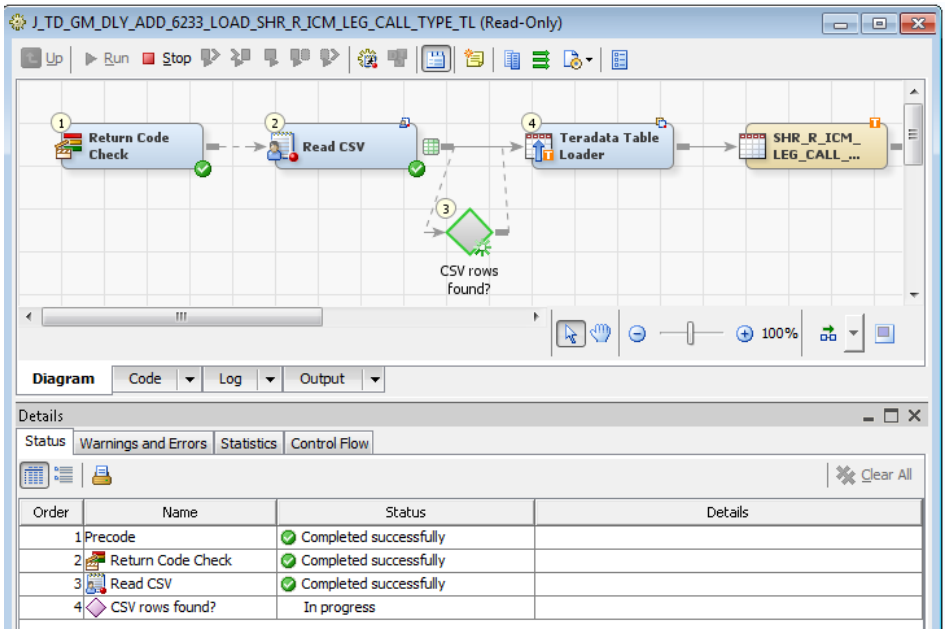



**Figure 18. Executing Conditional code in DIS**

For the same reason, if the job encounters an error in the conditionally executed code, the Conditional Start icon will turn red and the error will be reported as occurring in that step. Note the condition evaluation results returned in the Details for the Conditional transformation, i.e. *ETLS_DIAG: Condition flow did execute, condition was 10 > 0*

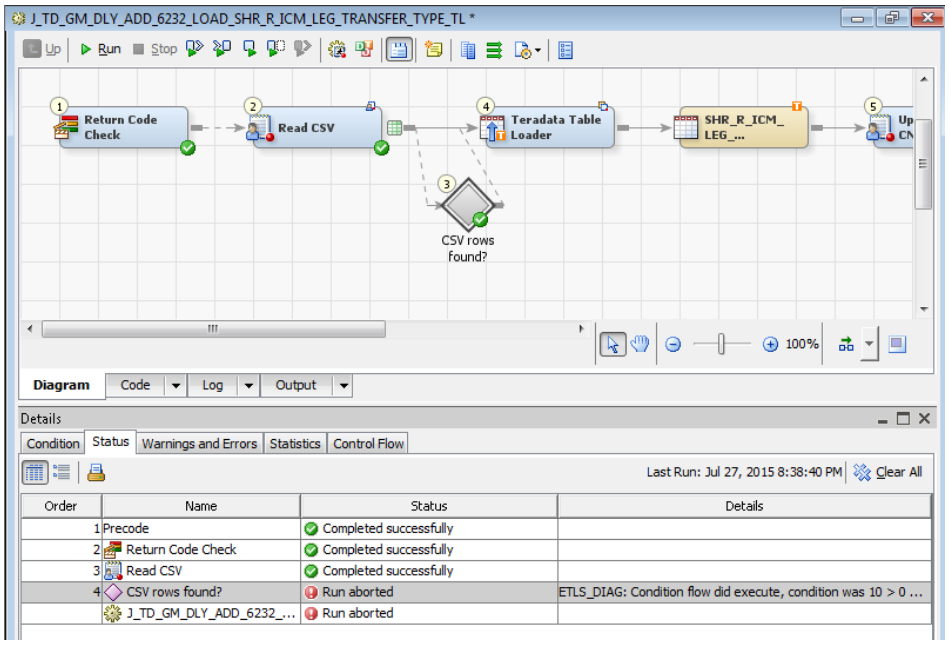



**Figure 19. Errors when executing Conditional code in DIS**

## CONCLUSION

The Loop and Conditional transformations are very useful additions to the DIS programmers' toolkit. The flexibility they afford allow more complex tasks to be developed in DIS without resorting to ( as much ) user-written code, decrease the job maintenance effort and, as a bonus, make you look smart. ☺

In addition to the vanilla implementation demonstrated in this paper, the looping functionality can take advantage of a grid environment by running iterations in parallel.

## REFERENCES

SAS Online Documentation. Available at http://support.sas.com/documentation/onlinedoc/etls

Dhillon, Rupinder, Looping in SAS DI Studio, TASS presentation. Available at http://www.torsas.ca/attachments/File/12142012/Dhillon-LoopingInDIStudio.pdf

## CONTACT INFORMATION

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

Harry Droogendyk
Stratia Consulting Inc.
conf@stratia.ca
www.stratia.ca/papers

## APPENDIX

**%attrn** macro – downloaded from Roland Rashleigh-Berry's repository found at
http://www.datasavantconsulting.com/roland/

```
/*<pre><b>
/ Program   : attrn.sas
/ Version   : 1.0
/ Author    : Roland Rashleigh-Berry
/ Date      : 04-May-2011
/ Purpose   : Function-style macro to return a numeric attribute of a dataset
/ SubMacros : none
/ Notes     : This is a low-level utility macro that other shell macros will
/             call. The full list of attributes can be found in the SAS
/             documentation. The most common ones used will be CRDTE and MODTE
/             (creation and last modification date), NOBS and NLOBS (number of
/             observations and number of logical [i.e. not marked for deletion]
/             observations) and NVARS (number of variables).
/
/             This macro will only work correctly for datasets (i.e. not views)
/             and where there are no dataset modifiers. If you need to subset
/             the data using a where clause or subset by using other means then
/             apply the subsetting and create a new dataset before calling this
/             macro.
/
/ Usage     : %let nobs=%attrn(dsname,nlobs);
/
/=============================================================================
/ PARAMETERS:
/-------name------- -------------------------description-------------------------
/ ds                Dataset name (pos) (do not use views or dataset modifiers)
/ attrib            Attribute (pos)
/=============================================================================
/ AMENDMENT HISTORY:
/ init --date-- mod-id ----------------------description-------------------------
/ rrb  13Feb07         "macro called" message added
/ rrb  30Jul07         Header tidy
/ rrb  17Dec07         Header tidy
/ rrb  04May11         Code tidy
/=============================================================================
/ This is public domain software. No guarantee as to suitability or accuracy is
/ given or implied. User uses this code entirely at their own risk.
/=============================================================================*/

%put MACRO CALLED: attrn v1.0;

%macro attrn(ds,attrib);
  %local dsid rc err;
  %let err=ERR%str(OR);
  %let dsid=%sysfunc(open(&ds,is));
  %if &dsid EQ 0 %then %do;
        %put &err: (attrn) Dataset &ds not opened due to the following reason:;
        %put %sysfunc(sysmsg());
  %end;
  %else %do;
        %sysfunc(attrn(&dsid,&attrib))
        %let rc=%sysfunc(close(&dsid));
  %end;
%mend attrn;
```