

SAS® DI: Introduction to User Written Transforms and Status Handling

Mike Miller, Ermas Consulting, Inc., Atlanta, GA

ABSTRACT

SAS® DI provides a feature rich environment for building and maintaining a data warehouse. The software comes with many ready to use transformations for loading tables, performing lookups and other common data warehousing techniques. It also provides the unique capability of allowing the user to create new transformations through the BASE SAS® programming language. Once defined, these transformations then become available for use by other processes and users. Through this capability, processing that is unique or too complex for the standard set of transformations can be encapsulated as a user written transform which then becomes reusable across processes. This paper is designed to provide an introduction on how to create a user written transform, the status handling capabilities provided by SAS® DI and how to link a user written transform into those status handling capabilities.

INTRODUCTION

BASE SAS® as a programming language provides an amazing range of flexibility and power. All of that capability is made available within SAS® DI through the use of user written transforms. In addition, SAS® DI provides a visual interface that encapsulates the user written code and allows it to be easily integrated into the overall process flow. Through the transform process icon, code options, input drop zones and output drop zones are surfaced as a standard set of controls that provide a visual interface to the user written code. All of these elements will be demonstrated through an example utility program created as a user written transform.

Well written code should take into account errors and processing exceptions. In SAS® DI, when errors, warnings or other job events occur they can be associated with specific actions such as sending an email notification. We'll look at this capability and how to incorporate it into a process flow and then demonstrate it within the context of a user written transform.

EXAMPLE USER WRITTEN TRANSFORM

To introduce the concept, this section presents a small utility program written as a user transformation. The program tracks the number of rows within a SAS® dataset and can be used as the basis for a more complex audit facility within a job flow. The intent is to keep the utility function simple, in order to highlight the features of the user written transformation. Because multiple processes could be attempting to write to the audit log at the same time, the example also highlights the use of the SAS® LOCK statement. For our example, the logging utility has the following requirements:

1. An input drop zone is used to define the table that needs to have a row count logged.
2. The following properties can be customized:
 - a. The audit/logging dataset where the logging should go, with a default name of ROWCOUNTS.
 - b. The maximum number of times to attempt to lock the logging data set for update. Because multiple processes may attempt to append information to the logging dataset at the same time, some type of locking facility is needed.
 - c. The number of seconds to wait after each lock attempt.

Drop Zones, Options AND %RCSET

There are several facilities for getting information in and out of the transform. The first concept to understand is the drop zone(s) that appears when a transform is placed on the job flow palette. These drop zones allow the user to connect a transform to a table or another transform. There are drop zones for both input and output. *Figure 1* shows the user transform on the job flow palette with the properties window open below. When the user places a table or transform into the drop zone it is linked to the transform code via a MACRO variable. Notice that the input drop zone, "Place table or transform here" is empty at this point, and that the process does not have an explicit output drop zone.

Instead, the logging dataset was defined as an option instead of an output drop zone. This brings us to the second method of getting information into the user transform, options. Shown at the bottom of *Figure 1* is the options tab from the transform properties dialog. This dialog can be displayed by right-clicking on the transform and selecting properties. The options tab allows the user to specify options that are also communicated to the transform via MACRO variables. As the figure shows, the user is allowed to specify four options. The audit/logging dataset, the

maximum number of times to attempt to lock the dataset for update and the number of seconds to wait after each lock attempt. The fourth option of creating the SYSLAST Macro Variable is a default, system defined option. This MACRO variable is set by SAS® DI with the name of the last dataset created by a transform. It is shown, but is not used in the example. This technique is often used to link the output from one transformation to the input of the next.

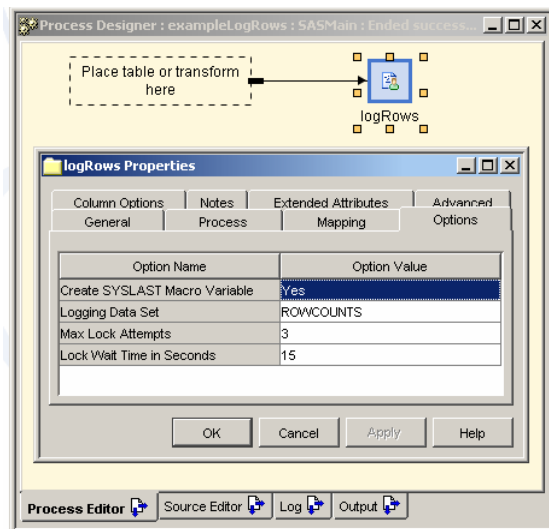


Figure 1 logRows Utility

There were a couple of reasons for making the logging dataset an option instead of a drop zone. First, the same metadata object cannot be represented on a process flow multiple times. To get around this requirement, multiple metadata objects pointing to the same physical table would need to be created. This would have made for a very messy process flow. Second, the value specified as the logging data set is likely to remain static. If in the future it is decided that it should never change, the option can be defined as non-modifiable.

Lastly, the transform needs to be able to communicate whether it was successful or if the transform generated any warnings or errors. This is done through calling the %RCSET MACRO which is included by SAS® DI when the coded is generated for the process. This topic is covered at the end of the paper as part of understanding status handling within SAS® DI. Below is a snippet of the source code generated by SAS® DI showing the MACRO interface variables highlighted in blue and the %RCSET MACRO highlighted in red.

- SYSLAST contains the name of the last dataset created
- _INPUT0 contains the table placed in the input drop zone.
- logDset contains the name of the logging data set.
- maxLockAttempts contains the maximum number of lock attempts to make.
- lockWaitSecs contains the number of seconds to wait between each lock attempt.

```

69      %let SYSLAST = %nrquote(source.FIX_RATE_DIM);
70
71      %let transformID = %quote(A51P0RQ0.BS0008HL);
72
73      %let TRANS_RC = 0;
74
75      /* Source table(s)/view(s) */
76      %let _INPUT0 = source.FIX_RATE_DIM;
77
78      /* Options */
79      LIBNAME prodUtil BASE "c:\sasdata\prodUtil" ;
80      %RCSET(&syslibrc);
81
82      %let logDset = prodUtil.ROWCOUNTS;
83      %let maxLockAttempts = %nrquote(3);
84      %let lockWaitSecs = %nrquote(15);

```

The `_INPUTn` variable is the default name generated for input drop zones and similarly `_OUTPUTn` is the default name generated for output drop zones. The next section shows how these names and prompts for the drop zones can be customized. Lines 82-84 show the MACRO variables created for each option defined for the logRows utility.

BUILDING THE USER WRITTEN TRANSFORM

Building the transform described above first requires writing a set of BASE SAS® code. That code can be developed directly in DI Studio, but from a development it can be easier to develop within either Enterprise Guide or the SAS® 9.1 Windows environment. When developing, the interface MACRO variables can be manually set at the top of the code to simulate the SAS® DI interface. The BASE SAS® code for the utility procedure is shown in the table below.

```
/* The _INPUT0 macro variable is in the form libname.dataset */
%let myLib = %upcase(%substr(&_INPUT0,1,%eval(%index(&_INPUT0,.)-1)));
%let myDataset = %upcase(%substr(&_INPUT0,%eval(%index(&_INPUT0,.)+1)));
data _null_;
    %put myLib=&myLib;
    %put myDataset=&myDataset; run;

proc sql noprint;
select nobs-delobs into :rowCount
from
    dictionary.tables
where
    upcase(memname)="&myDataset" and
    upcase(memtype)="DATA" and
    upcase(libname)="&myLib";

data newRecord;
    length rowCount 8
           tableName $255
           countTime 8
           libName $255;

    rowCount=&rowCount;
    tableName="&myDataset";
    countTime=datetime();
    libName="&myLib"; run;

%macro lockUpdate();
    %local updateComplete attempts;
    %let updateComplete=0;
    %let attempts=0;

    %do %until(&attempts>&maxLockAttempts or &updateComplete);
        /* Try to get exclusive access to the SAS data set. */
        lock &logDSet;

        %if &SYSLCKRC=0 %then %do;
            proc append base=&logDSet data=newRecord;
            lock &logDSet clear;
            %let updateComplete=1;
        %end;
        %else %do;
            %let sleep=%sysfunc( sleep(&lockWaitSecs,1) );
            %let attempts=%eval(&attempts+1);
            %put Lock attempt &attempts of &maxLockAttempts;
        %end;
        %if &updateComplete=0 %then %do;
            %RCSET(4);
        %end;
    %end;
%mend lockUpdate;
%lockUpdate; run;
```

Table 1 Code for Example User Transform

After selecting Tools->Transform Generator and naming the transform, the code can be cut-n-pasted directly into the wizard. The source code is stored by SAS® directly in metadata. These first two steps are shown in *Figure 2 Wizard Steps 1 and 2*. On the next step, the options are entered. *Figure 3* shows all three options entered in the background. In the foreground is the dialog window showing the dialog box for entering the max lock attempts option.

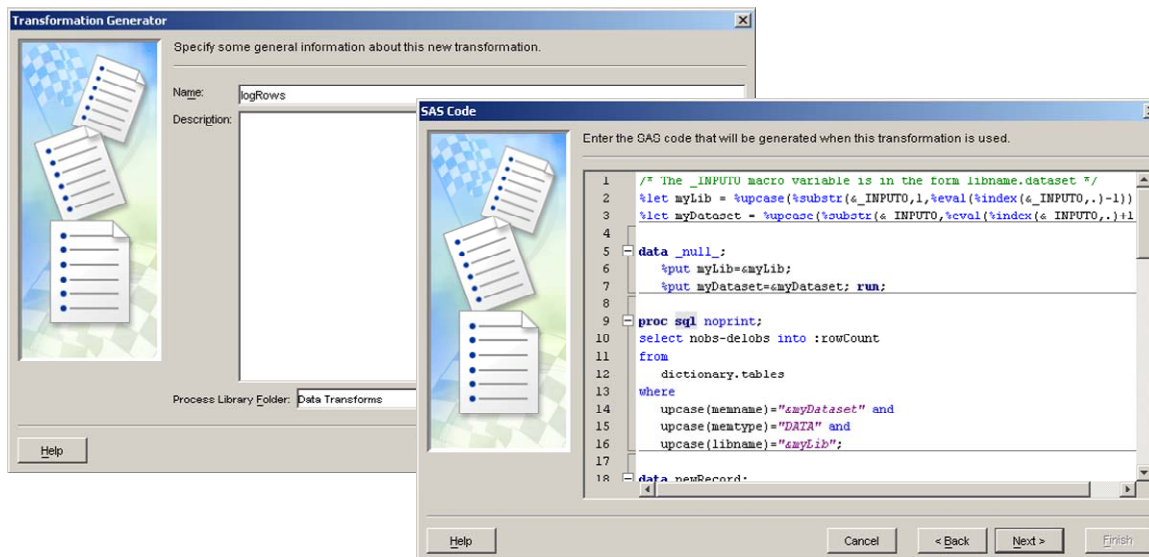


Figure 2 Wizard Steps 1 and 2

Note the dialog check boxes that determine whether the option is required and modifiable. Also notice the constraints dialog that allows the value to be constrained between integer values of 3 and 10. In the next step, the input and output drop zones are defined. In the case of this utility procedure, there is only one input drop zone.

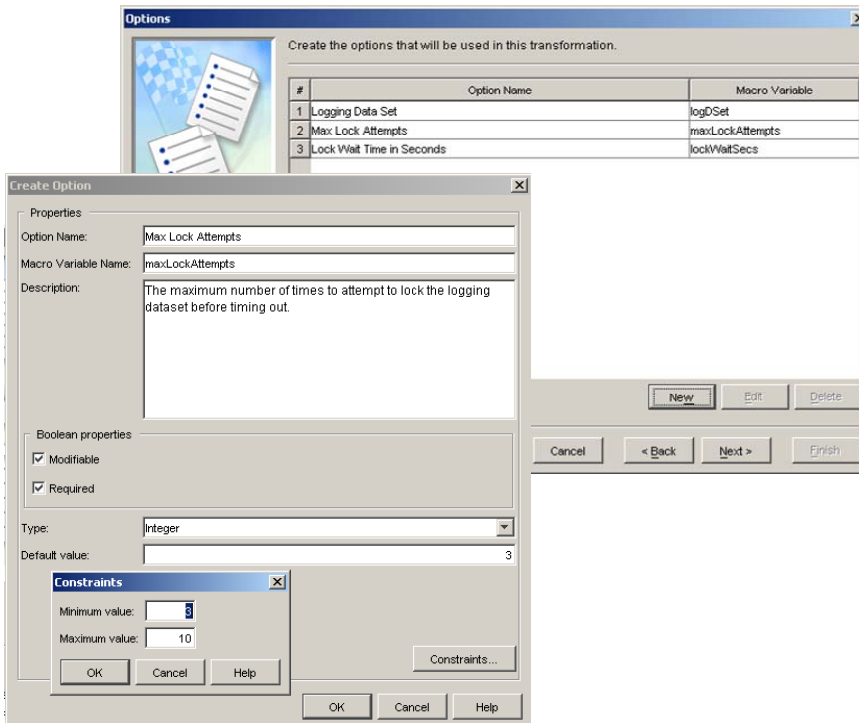


Figure 3 Entering the Options

Figure 4 shows the drop zone step and the dialog box (bottom left) that allows the macro variable names and prompts to be customized. The prompt “Place table or transform here” is the prompt that appears on the job palette. The final two steps (not shown) determine which library is associated with any data generated by the transform. That completes creating a user written transform; the transform now appears within the inventory of transforms and can be used across jobs.

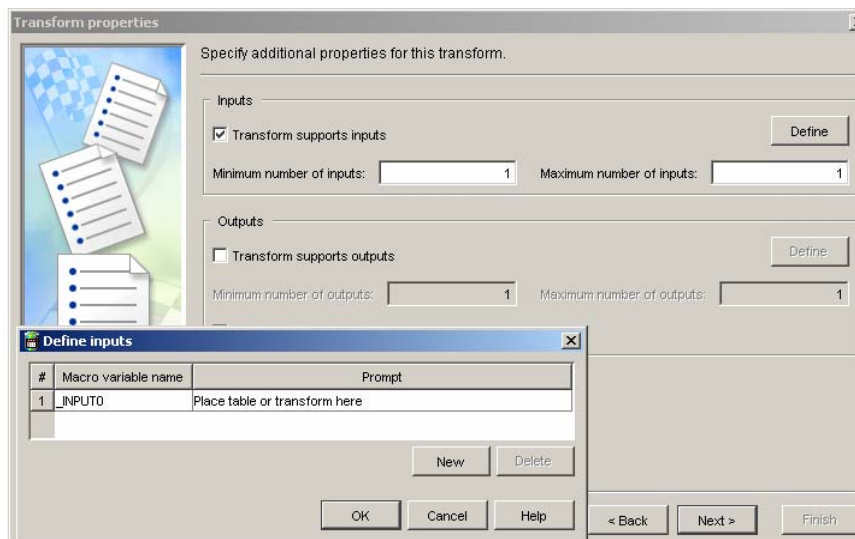


Figure 4 Entering the Drop Zones

STATUS HANDLING

There are two primary points where status handling can occur, at the job level, checking the status of the entire job and at the individual transformation level, checking the status of each transformation. Within a job, explicit checking can be performed by using the return code check transformation. Many transformations, such as the table loader, have built-in checking on the status handling tab of the transformation. The two components to status handling are a condition to check and a resulting action to be taken if that condition occurs.

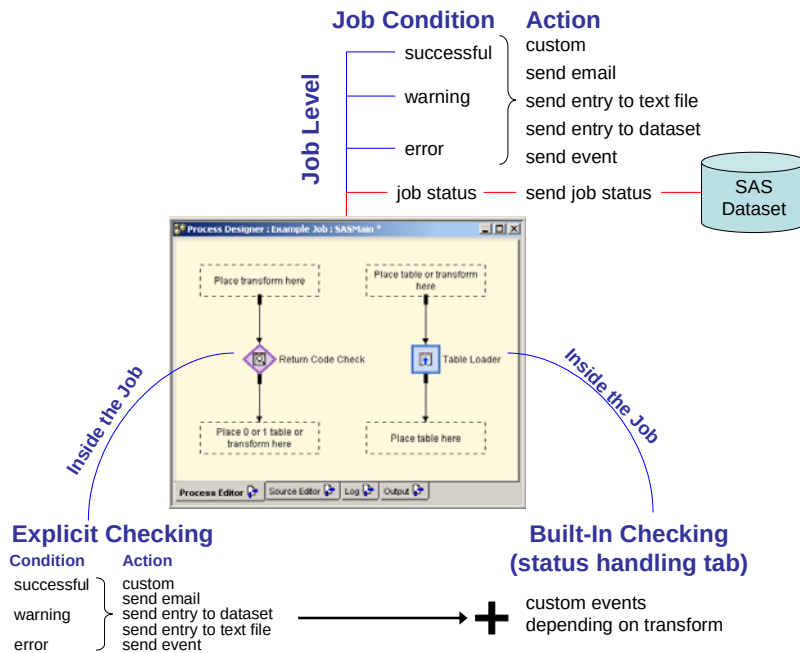


Figure 5 Default Status Handling Model

Figure 5 Default Status Handling Model, provides an illustration of this framework. SAS® DI provides great flexibility in terms of the conditions that can be checked and the resulting actions. These options can be configured from the Tools->Configure Status Handling main menu and are shown in *Figure 6 Configure Status Handling*. Most any combination of conditions and actions can be configured which then become available on the status handling tab.

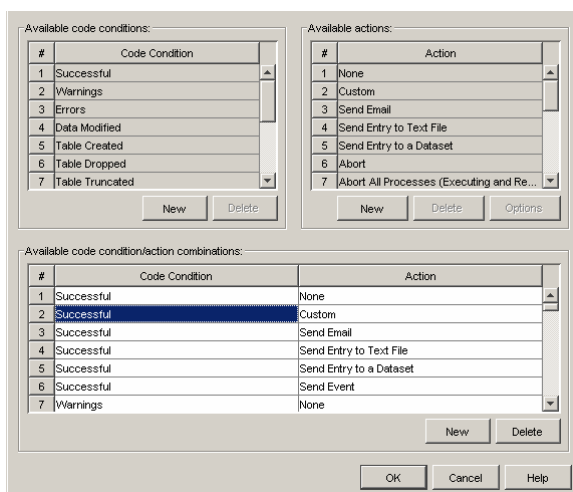


Figure 6 Configure Status Handling

Job Level Status Handling

Status handling at the job level can be enabled by right clicking within the job and selecting properties. From the status handling tab on the properties dialog, the four default condition options are available (as displayed at the top of *Figure 5* and in *Figure 7 Job Status Handling Options*): successful, warnings, errors and job status. For each of these conditions, there are multiple actions that can take place. Again, the conditions and actions available can be configured by the administrator. The default actions that are available at the job level include abort, custom, send email, send entry to text file, send entry to dataset, send event and send job status.

The job status condition is a special case and only has one action, send job status that is accessible through the job properties status handling tab. The other job status conditions can have any number of actions associated with them such as send email, send entry to dataset or send event as examples. This paper examines three of these actions in detail: send job status, send email and send entry to dataset.

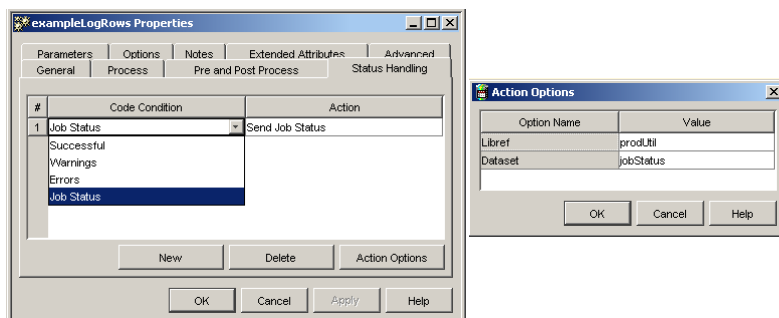


Figure 7 Job Status Handling Options

Job Status Condition and Send Job Status Action

The job status condition is available through the job properties status handling tab. This condition is “true” at the end of each job. The send job status action is the only action that can be associated with the job status condition, although the send job status action can be associated with other conditions. The send job status action writes a record to a SAS® dataset. This dataset is defined through the action options button and is shown to the right in *Figure 7*. The dialog prompts for a LIBREF and the name of the dataset.

There are two setup steps required before the send job status action can be used. First, the dataset needs to be defined ahead of time. The definition expected by SAS® DI is shown below.

The CONTENTS Procedure						
Data Set Name		PRODUTIL.JOBSTATUS			Observations	16
Member Type		DATA			Variables	10
Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Label	
10	endtime	Num	8	DATETIME18.	Job End Time	
4	etl_table	Char	32		Table Name	
8	JOB_RC	Num	8		Job Return Code	
6	jobname	Char	80		Job Name	
7	jobstat	Char	80		Job Status	
5	lib	Char	32		Library	
3	rafter	Num	8		No. Records After	
2	rbefore	Num	8		No. Records Before	
9	sttime	Num	8	DATETIME18.	Job Start Time	
1	user	Char	32		User ID	

Second, the job needs to have access to the LIBREF that is being used, in our case PRODUTIL. There are two ways to make this accessible to the job. First, is to define this LIBREF under the Pre and Post Process tab for the job. This is shown in *Figure 8* with the edit window open displaying the LIBNAME definition. Alternatively, the library could have been pre-assigned outside of SAS® DI through the autoexec.sas file or SMC facility.

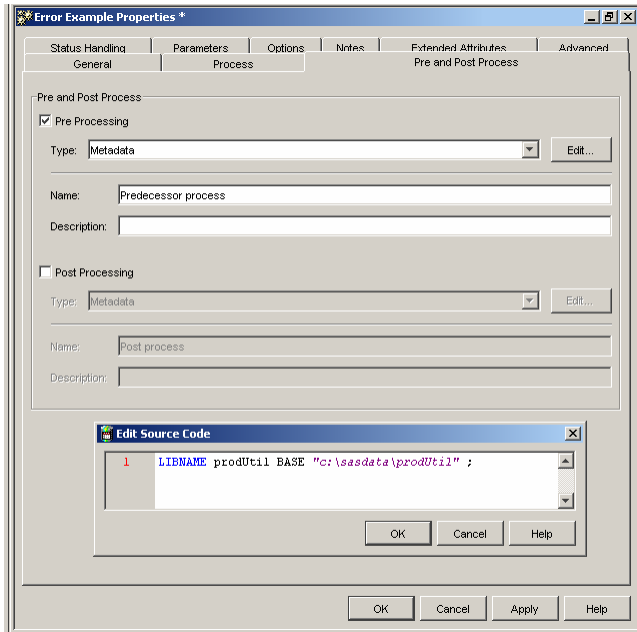


Figure 8 Pre and Post Process

Figure 9 below shows two rows in the send job status dataset for the same job. The first row shows the job completing successfully, the second row shows the job ending with a warning.

#	user	rbefore	rafter	etl_table	lib	jobname	jobstat	job_rc	stime	endtime
1	sassrv	0	0	0	0	exampleLogRows	Job Successful	0	15JUL07:15...	15JUL07:15:1...
2	sassrv	0	0	0	0	exampleLogRows	Job Ended with Warnings	4	15JUL07:15...	15JUL07:15:1...

Figure 9 Send Status Dataset

Send Entry to Dataset Action

The send entry to dataset action also provides a facility for logging information to a SAS dataset. Unlike the send job status action, the send entry to dataset action is not as tightly preconfigured to log certain information. Like the send job status action, the dataset must be predefined, it must follow a defined format, and a LIBNAME reference must be made available to the job. The definition for the dataset is very simple. It has two columns as shown in the table below: a time stamp of when the entry was sent and a message field for the entry.

Column	Type	Length	Example Value
datetime	character	16	23JUN07:16:45:36
message	character	80	Successful

The message that is sent can be configured through the action options button. Figure 10 below shows a message being configured to also include the job id and name in addition to the text "Successful". The LIBNAME and dataset where to send the entry are configured through this same dialog. MACRO variables available in the job, such as

JOBID and ETL_JOBNAME can be included in the message string. The ETL_JOBNAME variable is available because the send job status action has been configured for the same job.

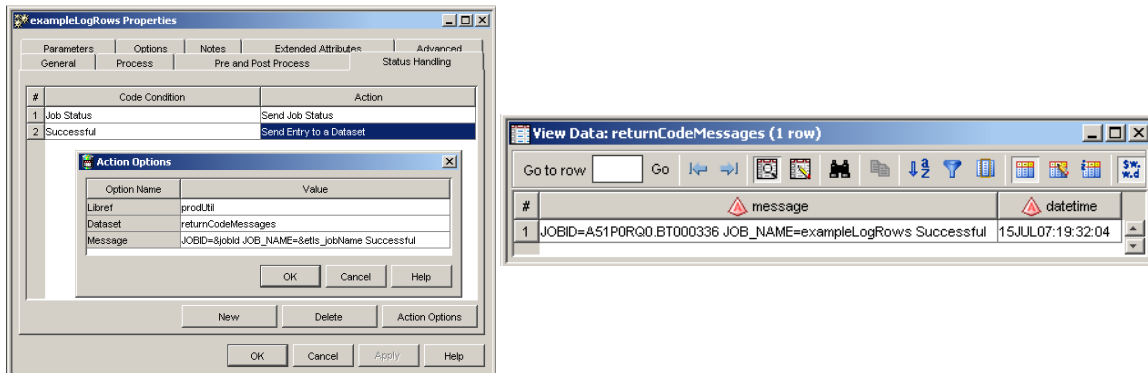


Figure 10 Send Entry to Dataset Action Options

The messages are appended to the SAS® dataset and an example message is shown to the right in *Figure 10*.

Send Email Action

Before using the send email action, an SMTP server or other SAS® supported facility for sending email must be enabled. For an SMTP server, the following options must be added to the ..\Lev1\SASMain\sasv9.cfg file.

```
-emailsys SMTP
-emailhost <host name or ip address>
-emailid <sending email userid>
-emailpw <password>
-emailport <25 for SMTP>
```

To configure the send email action, a comma separated list of email addresses is supplied along with the text message. *Figure 11* shows the send email action being configured along with the email message that is generated to the right.

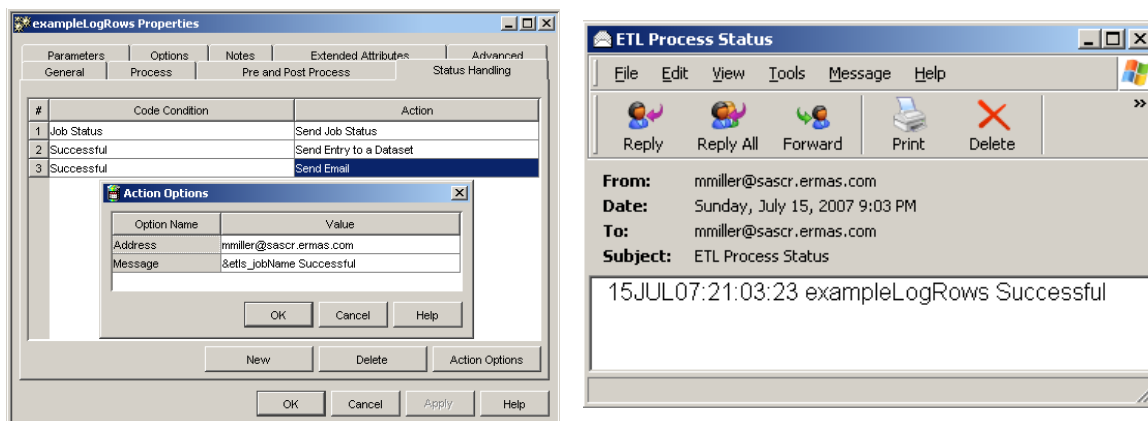


Figure 11 Send Email Action Option

Transform Level Status Handling and the Abort Action

The previous discussion was centered on status handling at the job level, but the same concepts are applicable at the transform level. As *Figure 5* depicts, status handling can be performed either via the return code check transform or through the status handling tab built-in to many transforms. At the transform level, many additional conditions

become available depending on the type of transform. For example, the table loader transform has the following additional conditions: data modified, table created, table dropped and table truncated.

There is one additional action that will be covered in this introduction, the abort action. This action is most applicable to status handling at the transform level and so is discussed in this section. When configuring the abort action option, the dialog as shown in *Figure 12* is displayed. In this example, a return code of 999 is entered as the value for the return code.

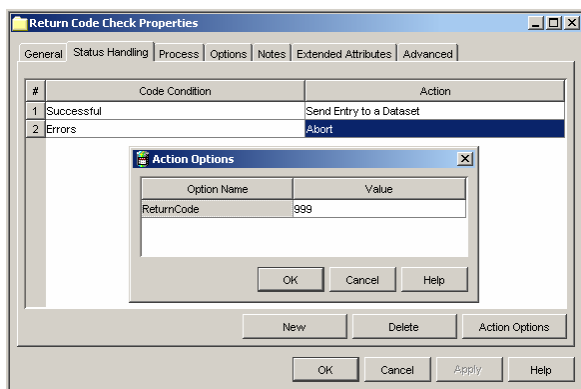


Figure 12 Abort Action Options

At the transform level, this generates the following code:

```
%if (&TRANS_RC ge 5) %then
%do;
  /* Abort job with a return code of %str(999) */
  data _null_;
    abort return %str(999);
  run;
%end;
```

The TRANS_RC macro variable contains the highest return code from the transform. The code checks that this return code is greater than 5, which would indicate a SAS® error. The abort return statement is then executed to pass back to the operating system the condition code of 999 that was entered in the action options dialog in *Figure 12*. The difference between the abort action at the transform level and the abort action at the job level is when and what return code is checked. At the transform level, it is done after the transform has executed and the TRANS_RC return code for the transform is checked. At the job level, the final job return code, JOB_RC, is checked as the final step in the job. This is shown in the code snippet below.

```
%macro etls_jobRCchk;          /* Check return code for job */
  %if (&JOB_RC ge 5) %then
  %do;                          /* Abort job with a return code of %str(999) */
    data _null_;
      abort return %str(999);
    run;
  %end;
%mend etls_jobRCchk;
%etls_jobRCchk;
```

This segues into our last topic of incorporating return codes into a user written transform.

%RCSET Macro

There are two MACRO variables that SAS® DI maintains that track the return codes within the job, TRANS_RC and JOB_RC. Below is the definition from the online help within SAS® DI for these two variables.

- TRANS_RC - This variable is cleared at the beginning of generated code for each transformation. The RCSET macro resets the TRANS_RC variable after each library assignment statement and after the main

generated code for the transformation. If there is more than one processing step involved in transformation code, the TRANS_RC variable is set to the highest value.

- JOB_RC - This variable is set to 0 at the top of the job. It is not cleared as the code for the job is executed. At the end of the job, the RCSET MACRO sets the JOB_RC variable to the highest return code value of the entire job.

The %RCSET MACRO is automatically included in a SAS® DI job to manage setting these two MACRO variables. The MACRO can be called within user written code and it is worth taking a look at the code. The MACRO maintains the highest return code generated for each transform and the highest return code overall for the job.

```
%macro rcSet(error);  
  %if (&error gt &trans_rc) %then  
    %let trans_rc = &error;  
  %if (&error gt &job_rc) %then  
    %let job_rc = &error;  
%mend rcSet;
```

Referring back to the very beginning of the paper in *Table 1*, the user written code for the example transform calls the %RCSET MACRO and passes a return code of 4 to indicate a warning when the transform is unable to lock and update the logging dataset. Choosing the value 4 is critical because of the way actions define success, error or warning. Referring to the table below, if a warning email needs to be generated when a certain condition occurs, then the %RCSET MACRO should be called with a value of 4. Calling the MACRO with any another value will indicate either a success, error or an undefined status.

Return Code	Status	Job Status Field for the Send Job Status Action
0	Success	Job Successful
4	Warning	Job Ended with Warnings
>4	Error	Job Ended with Errors
1,2,3 or <0	Undefined	Will result in a 'blank' being populated in the jobstat column by the send job status action

CONCLUSION

Two powerful SAS® DI concepts were introduced in this paper, the user written transform and the built-in status handling capabilities that come with SAS® DI. Through the user written transform it is possible to extend SAS® DI through BASE SAS® code and make those extensions readily available to other users and processes. The SAS® DI status handling capabilities enable the software to capture, track and send out notifications when errors, warnings or other status conditions occur. Through the use of the %RCSET MACRO it is possible to link a user written transform into the general status handling capabilities provided by SAS® DI.

CONTACT INFORMATION

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

Name: Mike Miller
Enterprise: Ermas Consulting Inc.
Address: 4080 McGinnis Ferry Road, Suite 101
City, State ZIP: Alpharetta, GA 30005
Work Phone: 704 607-8990
Fax: 770 390-9094
E-mail: mmiller@ermas.com
Web: <http://www.ermas.com/>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.