Paper 039-2010

SAS® Data Integration Studio Tips and Tricks

Chris Olinger, d-Wise Technologies, Inc., Raleigh, NC
David Kratz, d-Wise Technologies, Inc., Raleigh, NC

ABSTRACT

Anyone who has used SAS® Data Integration Studio to process and transform data has invariably said to themselves "is there another way to do the things that I want to do?" SAS® Data Integration Studio is great at certain tasks such as standard transformations, visual representations of code, and impact analysis. However, it can be tricky to perform some industry standard tasks such as writing jobs that can be promoted and configured through external options files, loading metadata from non-supported external sources (such as Excel), and managing macros. This tutorial will explore some of the work-arounds that we have come to rely on while using the application. We will also explore the latest version of SAS Data Integration Studio to see if any of our pet peeves have been fixed.

INTRODUCTION

Over the last few years, our company has built various SAS warehouses using SAS® Data Integration Studio (DI Studio). We have found that the tool can be used to good effect, and that features like Impact Analysis, and some standard transformations, are valuable in implementing a warehouse. That said, the 3.4 version of DI Studio has some drawbacks that need consideration. Version 4.2 of DI Studio has a great many improvements in these areas, and is in our opinion, a superior tool. This paper explores both versions of these tools and provides a set of Tips and Tricks that can be leveraged in both versions. We hope that you will be able to use these techniques to help you get your job done and that you find them helpful. They have proven invaluable to us.

We will cover the following areas and provide tricks that can be used in various situations:

- Replacing an incorrect table in a job
- Reading and writing to the same table in a job
- Tricks to avoid the expression limit in the SQL transform
- Extract node tricks
- Custom date assignment in the SCD transform
- Leveraging external options files to customize jobs
- Using empty transform to set macro variables
- Managing macros and formats
- Managing metadata using an external source

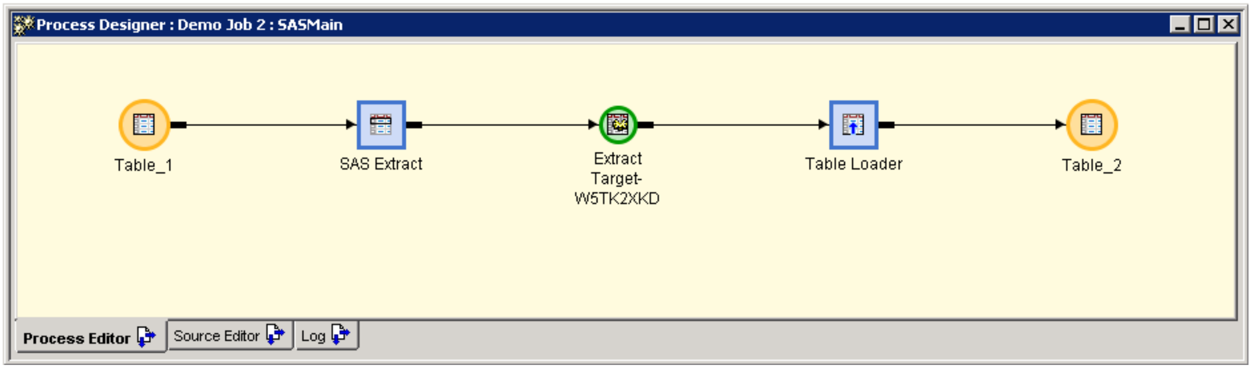
Note, with these tricks, your mileage may vary. They work well for us, but you must decide based on your situation if they are applicable or not. And now, onto the show!

REPLACING AN INCORRECT TABLE IN A JOB

THE SETUP

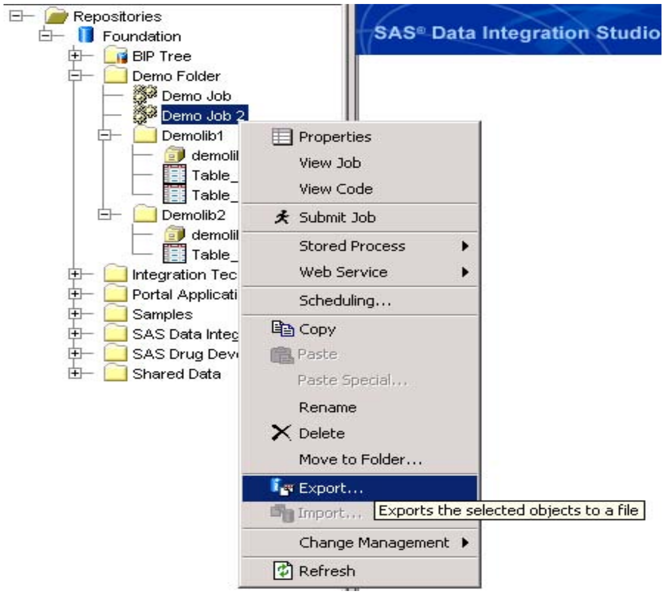
Let's say that there are several similarly named and similarly structured tables, and the wrong one has been used in a job. Or, let's say that there is a single job one wishes to reuse across multiple studies. In both cases it is now necessary to replace the table in a pre-extant job.

SAS 9.1.3, DI VERSION 3.4

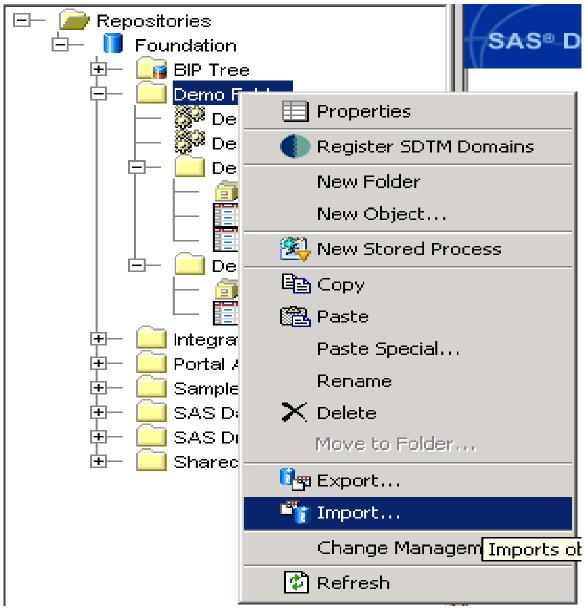


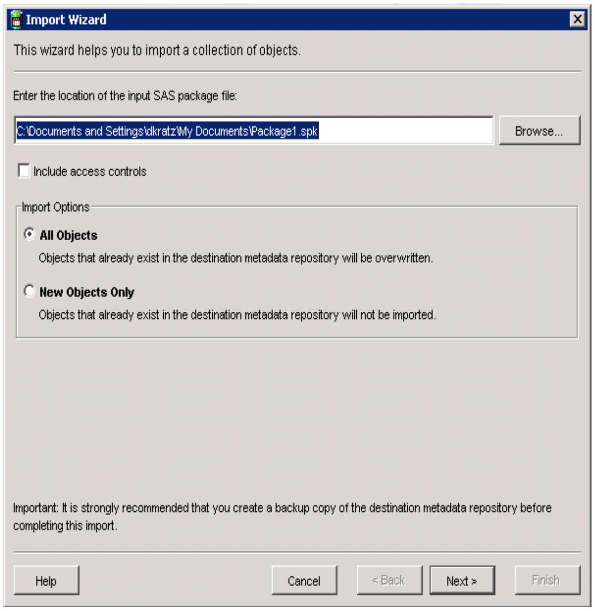
Above is a trivial example job in SAS 9.1.3. Table_1 is extracted and loaded into Table_2. The problem is that the source for the extract needs to be Table_3. This could be achieved by deleting the linkage between Table_1 and the extract node, and replacing the table with the correct one. However, this would discard the extract node's mappings, as well as messing up any assigned expressions. However, there is a better way: exporting and importing the job.

To export the job, simply right click on the job and select Export.



This will allow you to choose an appropriate export location and file name for the exported "spk" file. After selecting Export, proceed through the export process. It doesn't really matter what one names the export file, or where one puts it, so long as the file is easy to get back too and doesn't overwrite anything one would rather keep. After saving, select Import:





This wizard helps you to import a collection of objects.

Enter the location of the input SAS package file: C:\Documents and Settings\dkratz\My Documents\Package1.spk

☐ Include access controls

Import Options

● **All Objects**

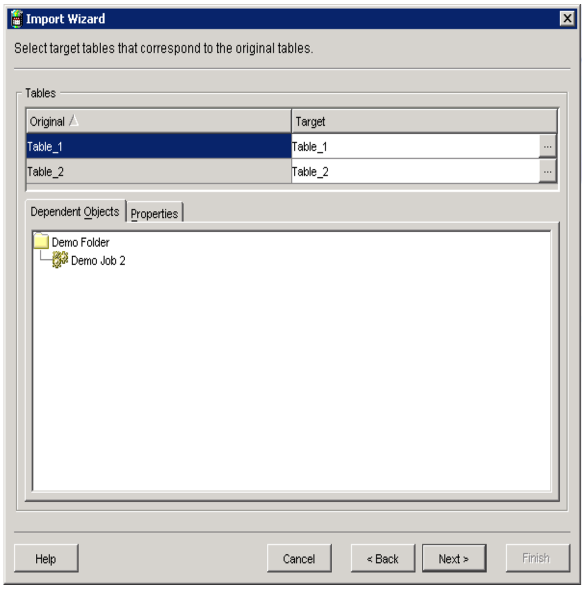
Objects that already exist in the destination metadata repository will be overwritten.

○ **New Objects Only**

Objects that already exist in the destination metadata repository will not be imported.

Important: It is strongly recommended that you create a backup copy of the destination metadata repository before completing this import.

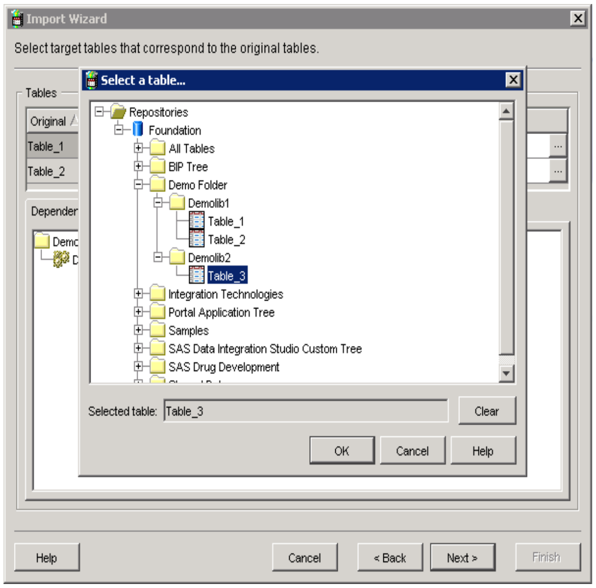
As one proceeds through the import dialogue, they will be asked to select values for the values in metadata. This will include the tables. By clicking on the box with the ellipses next to a table name, one will be given a dialogue to select a table. Whichever table is selected will replace the original. As long as that table shares column names with table being replaced, the mapping and expressions will be preserved.



Select target tables that correspond to the original tables.

Tables

Original	Target
Table_1	Table_1
Table_2	Table_2

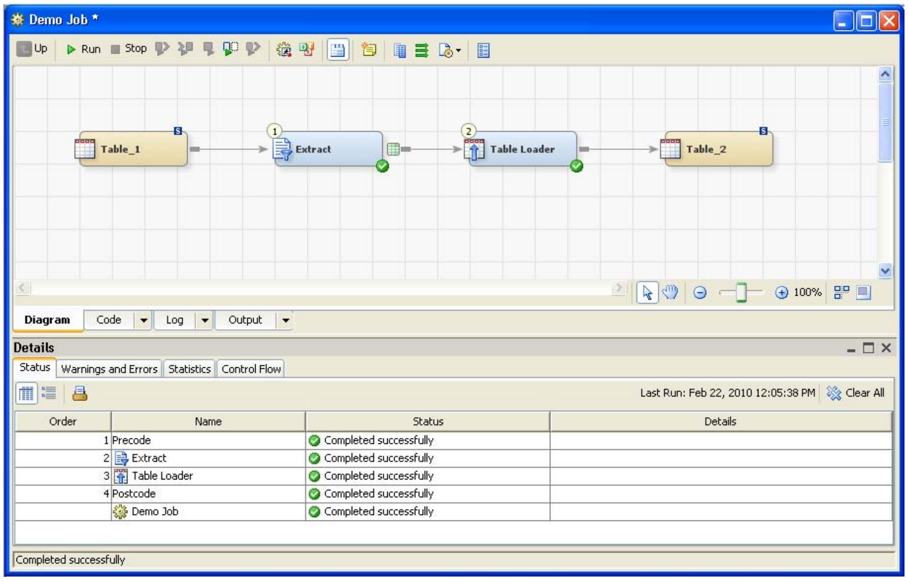


Select target tables that correspond to the original tables.

Selected table: Table_3

SAS 9.2, DI VERSION 4.2

SAS 9.2 makes this very easy. This is the same job as before, but in 9.2.

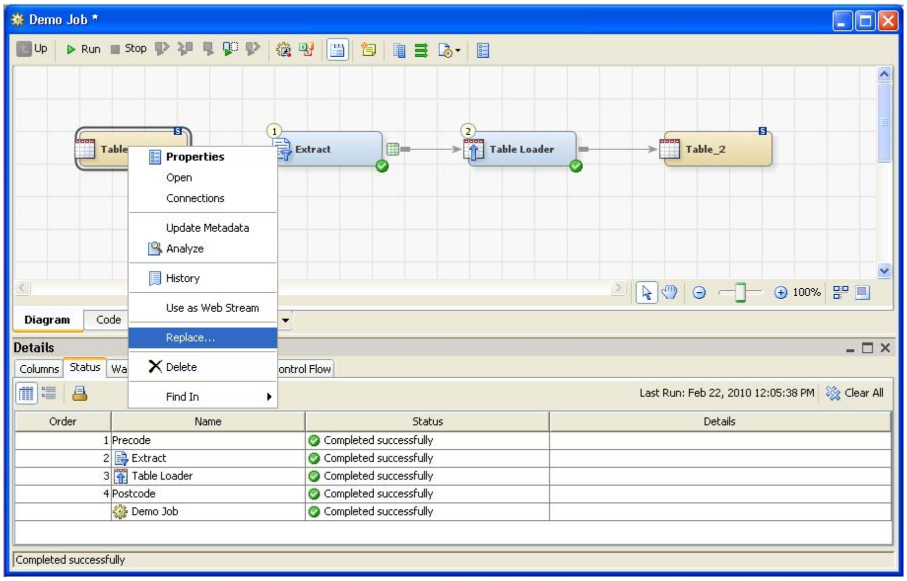


Last Run: Feb 22, 2010 12:05:38 PM

Order	Name	Status	Details
1	Precode	Completed successfully	
2	Extract	Completed successfully	
3	Table Loader	Completed successfully	
4	Postcode	Completed successfully	
	Demo Job	Completed successfully	

Completed successfully

Simply right click on the table and choose replace.

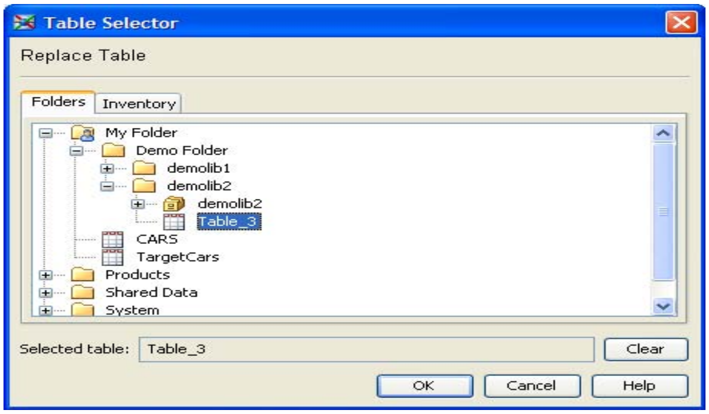


Last Run: Feb 22, 2010 12:05:38 PM

Order	Name	Status	Details
1	Precode	Completed successfully	
2	Extract	Completed successfully	
3	Table Loader	Completed successfully	
4	Postcode	Completed successfully	
	Demo Job	Completed successfully	

Completed successfully

From the Table selector dialogue that comes up, choose the replacement table.



Selected table: Table_3

It’s important to note that replace works best, in both 9.1.3 and 9.2, when the tables that are being swapped share column names. **Only when this is the case will mappings and expressions survive the procedure**. If there is only a partial sharing, only the mappings of those columns and the expressions involving only those shared columns will survive.

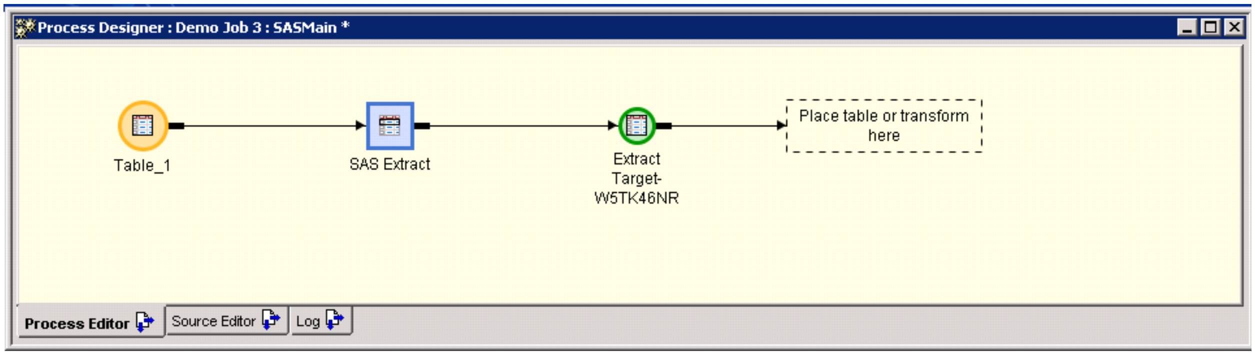
Final note: this procedure can be used to swap any accessible table. That being the case, it can be used to swap to the same table located in a different library, if that was the goal.

READING AND WRITING TO THE SAME TABLE

THE SETUP

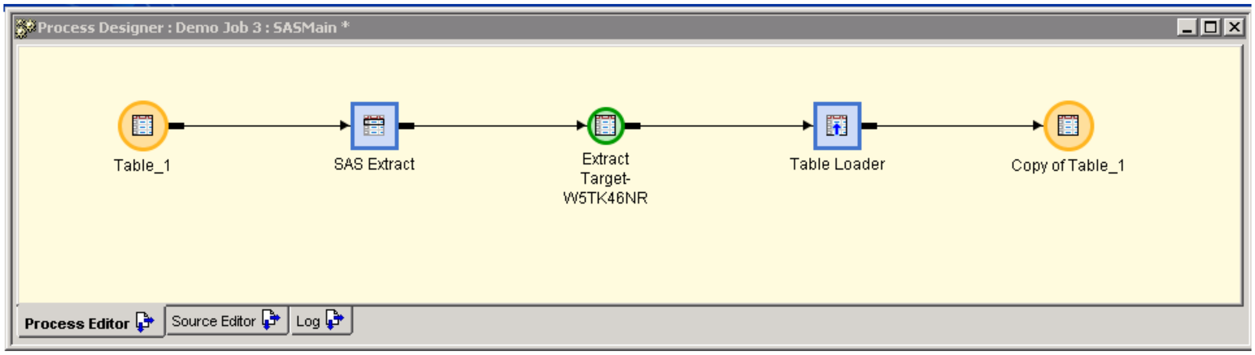
In base SAS it is a relatively common thing to perform operations in place on a table. However, this operation is not intuitively obvious in DI Studio circa 9.1.3. A job allows only one reference to a given table. Any attempt to link the same table as both input and output will result in a system error. If you would like to use the table for both input and output you must instead register the same table metadata twice.

SAS 9.1.3, DI VERSION 3.4



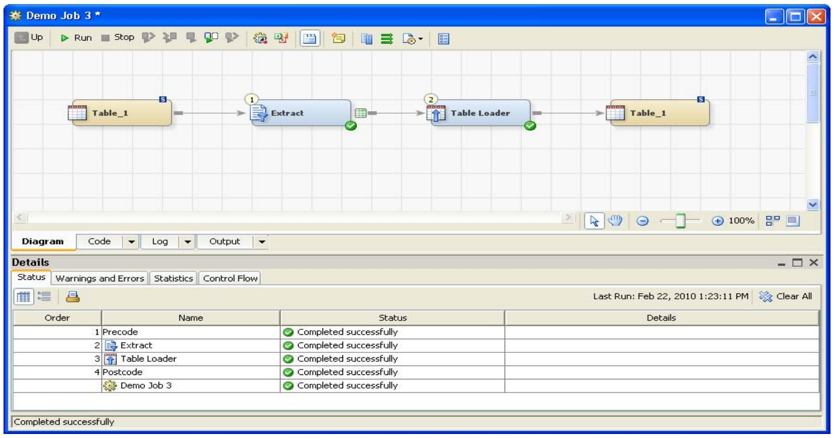
According to the DI Studio help file, the secret lies in library names. By creating two different libnames which each point to the same physical location, a single table can be defined in metadata twice. However, there is a much simpler solution: using copy and paste to duplicate a table’s definition in metadata.

Simply copy the table in question and paste it. Give the “new” table an arbitrary name (or leave it as “Copy of …”), preferably a name different than the name of the original table (there are many reasons for this, but the specifics are beyond the scope of this paper). Now, simply choose the new table as the target of the process. You will have two copies of metadata to maintain but the job will be simpler for it.



SAS 9.2, DI VERSION 4.2

In SAS 9.2, reading and writing to the same table is as simple as dragging the table object onto the job twice, as this picture indicates. Just route the data from one, into the other.

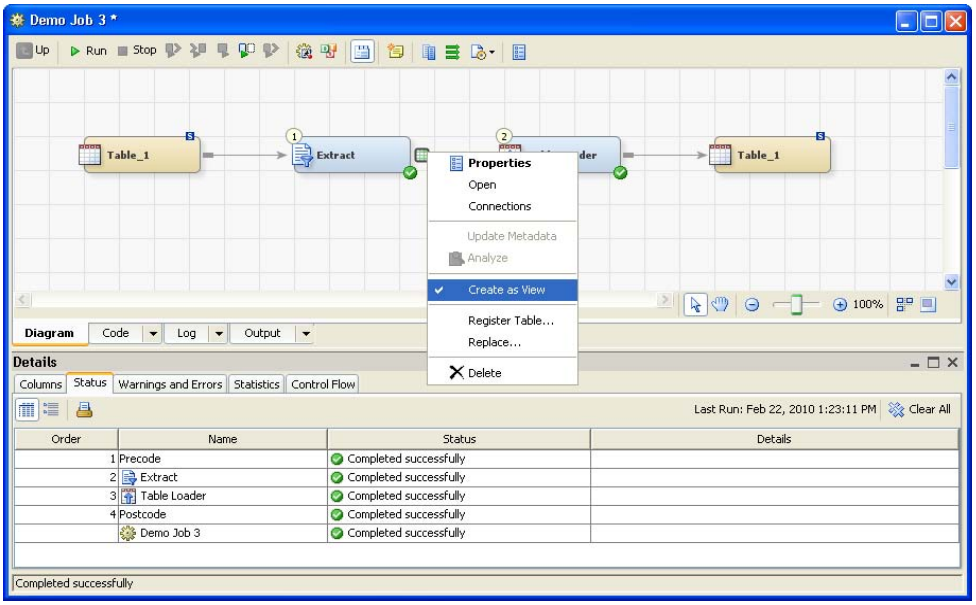


Last Run: Feb 22, 2010 1:23:11 PM

Order	Name	Status	Details
1	Precode	Completed successfully	
2	Extract	Completed successfully	
3	Table Loader	Completed successfully	
4	Postcode	Completed successfully	
	Demo Job 3	Completed successfully	

Completed successfully

Using these techniques may cause one minor problem that DI users might not be familiar with. Essentially, most DI transformations output as a View, which is fine, until one attempts to append that view to the table the view is based on. That being the case, unless planning a direct load, one must make one of the temporary views before the loading step into a table. Un-checking this option causes the temporary output to be written to a table, and avoids the problem.



Last Run: Feb 22, 2010 1:23:11 PM

Order	Name	Status	Details
1	Precode	Completed successfully	
2	Extract	Completed successfully	
3	Table Loader	Completed successfully	
4	Postcode	Completed successfully	
	Demo Job 3	Completed successfully	

Completed successfully

AVOIDING THE EXPRESSION LIMIT

THE SETUP

Consider the following Case statement:

```
Case
  when A = 1 then "Supercallafragalisticexpialladocious1"
  when A = 2 then "Supercallafragalisticexpialladocious2"
  when A = 3 then "Supercallafragalisticexpialladocious3"
  when A = 4 then "Supercallafragalisticexpialladocious4"
```

```
      when A = 5 then "Supercallafragalisticexpialladocious5"
      when A = 6 then "Supercallafragalisticexpialladocious6"
      when A = 7 then "Supercallafragalisticexpialladocious7"
      else "dociousallaexpiciousfragacallarupes"
    End
```

The code above is notable for 2 reasons. First, it is completely pointless. Second, it will not fit inside the expression box in the select tab of the SQL join in SAS 9.1.3, as it exceeds 254 characters.

SAS 9.1.3, DI VERSION 3.4

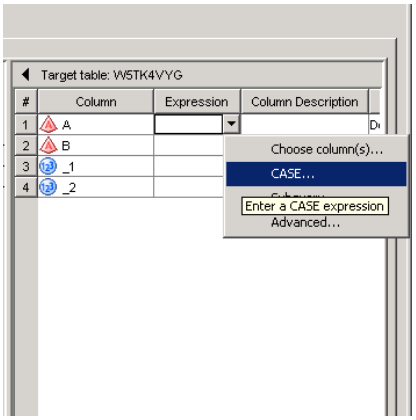
The error here is a peculiar one, especially since the extract node is not so encumbered and the underlying code written by both is an SQL step. The expression box on the mapping tab of the extract node can support a seemingly unlimited number of characters. Thus an obvious way of getting around the limitation of the SQL node is to follow it with an extract containing the expression needed. However, this is not always convenient, or efficient, and so we investigate the obvious alternatives:

- Case statement writer
- Locally defined formats
- Renaming tables, columns

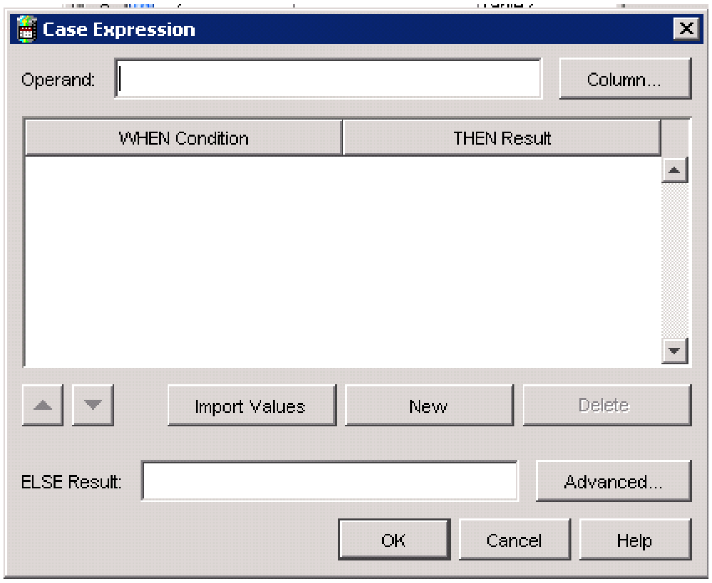
CASE STATEMENT WRITER

Generally, when expression statements get longer than 254 characters, it is because a case statement is being written. A case statement is effectively an “if case x then do y else if case z …” statement that is used with SQL. Throwing these into the expression node is a quick and easy way of conditionally setting values.

The expression box of an SQL join node has the option of using the CASE dialogue to write these statements.


Target table: W5TK4VYG

#	Column	Expression	Column Description
1	A		
2	B		
3	_1		
4	_2		



While we find the Case Expression box more tedious to use than simply writing the case expression ourselves, it does have its advantages. First, it allows the contents of the Case statement to be presented in an ordered fashion, increasing the reusability of the process. Second, the use of it allows one to bypass the 254 character limit. We have yet to find an upper limit to the number of when conditions we can create using the Case dialogue.

LOCALLY DEFINED FORMATS

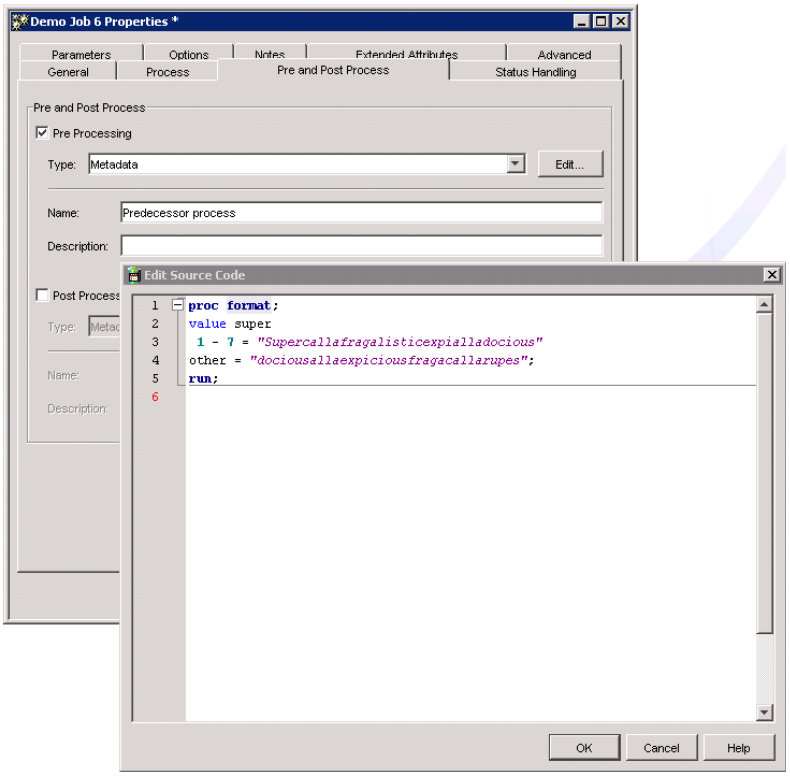
Another way of cheating the character limit is to type fewer characters, and allow for a substitution to take place. If the data is set up properly, one could simply create a custom format and use the put function to output the column that way.

For example, if one established a format like so:

```
proc format;
picture super
 1 - 7 = 9 (prefix="Supercallafragalisticexpialladocious")
 other = "dociousallaexpiciousfragacallarupes";
run;
```

One could simulate the case statement above simply placing “put(A,super.)” in the expression box.

Since formats designed solely for the sake of convenience are not likely to be widely useful, they should be defined locally, with a scope only as large as the current job. We suggest that they be added to the pre-process code section of the job.



☑ Pre Processing

Type: Metadata

Name: Predecessor process

Description:

☐ Post Process

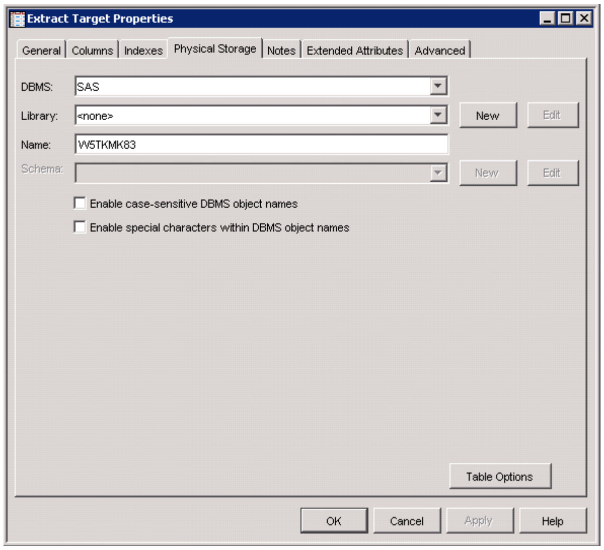
```
proc format;
value super
 1 - 7 = "Supercallafragalisticexpialladocious"
other = "dociousallaexpiciousfragacallarupes";
run;
```

RENAMING TABLES, COLUMNS

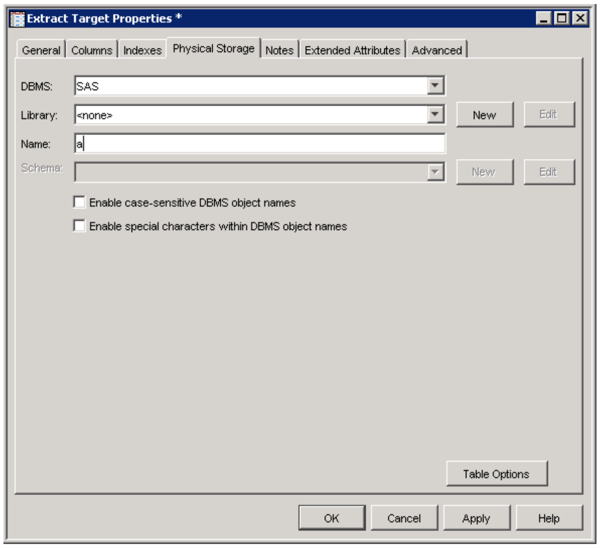
When the difference between an expression length and the allowed length is small, it is often possible to reduce the length of table and column names in order to get in under the limit. Given the way that DI studio works, it will often decorate columns referenced in an expression in the format “table_name.column_name”. Depending on the tables and columns involved, this can eat up a lot of space.

A considerable space savings can be garnered by giving the temporary table leading into the join a shorter name. However, this comes at the tradeoff of having less descriptive names (which gets even worse when one does the same thing with column names).

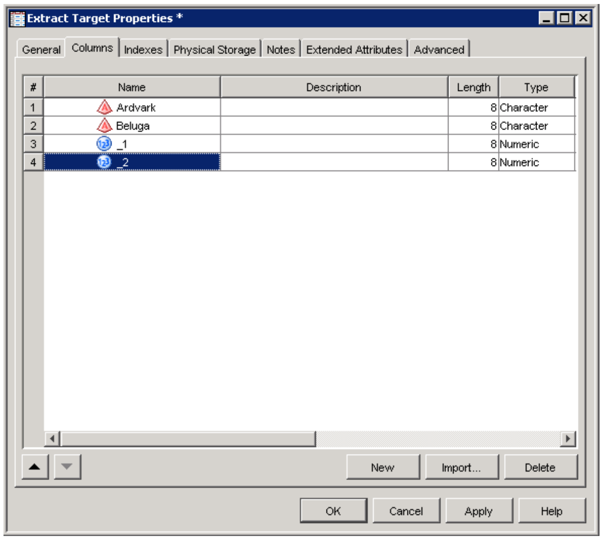
Changing table names is as simple as selecting the Properties of the temporary table that feeds the SQL join, switching to the Physical Storage tab, and replacing the randomly assigned moniker with a shorter one. One should take care to make sure that they do not replicate another table name while doing this. Similarly, column names can be changed in the columns tab of the same table.



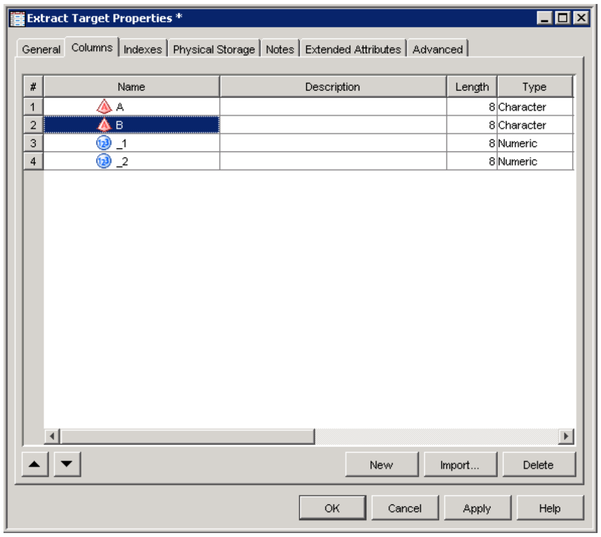
DBMS: SAS
Library: <none>
Name: W5TKMK83
Schema:
☐ Enable case-sensitive DBMS object names
☐ Enable special characters within DBMS object names



DBMS: SAS
Library: <none>
Name: a
Schema:
☐ Enable case-sensitive DBMS object names
☐ Enable special characters within DBMS object names



#	Name	Description	Length	Type
1	Ardvark		8	Character
2	Beluga		8	Character
3	_1		8	Numeric
4	_2		8	Numeric



#	Name	Description	Length	Type
1	A		8	Character
2	B		8	Character
3	_1		8	Numeric
4	_2		8	Numeric

SAS 9.2, DI VERSION 4.2

This SQL expression bug has been fixed in 9.2. It is no longer necessary to trick the system into shorter expressions. As such, add any expression that you would like, including extra long case statements.

EXTRACT NODE TRICKS

The extract node is one of the simplest tools that DI Studio 9.1.3 provides. The code it generates is generally just creating a view as a SQL select. However, it has a behavior in DI studio that allows one to perform some interesting tricks with it. Namely, the extract node can be dropped into the middle of an existing flow, and deleted, without breaking the flow.

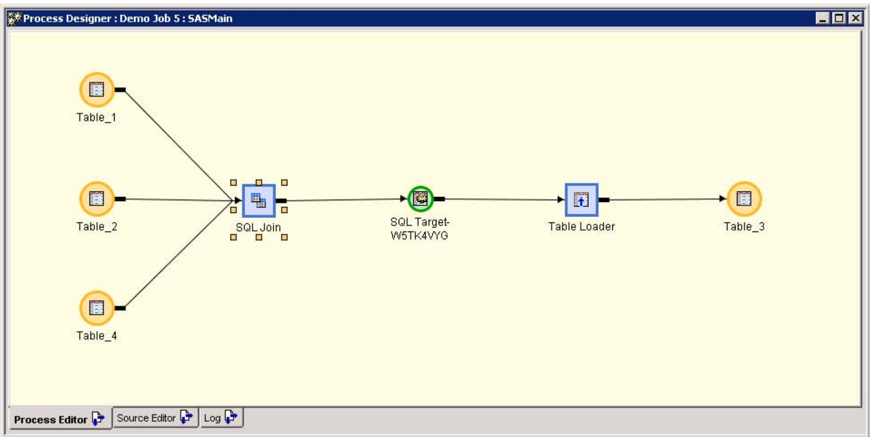
CHANGING JOIN ORDER

THE SETUP

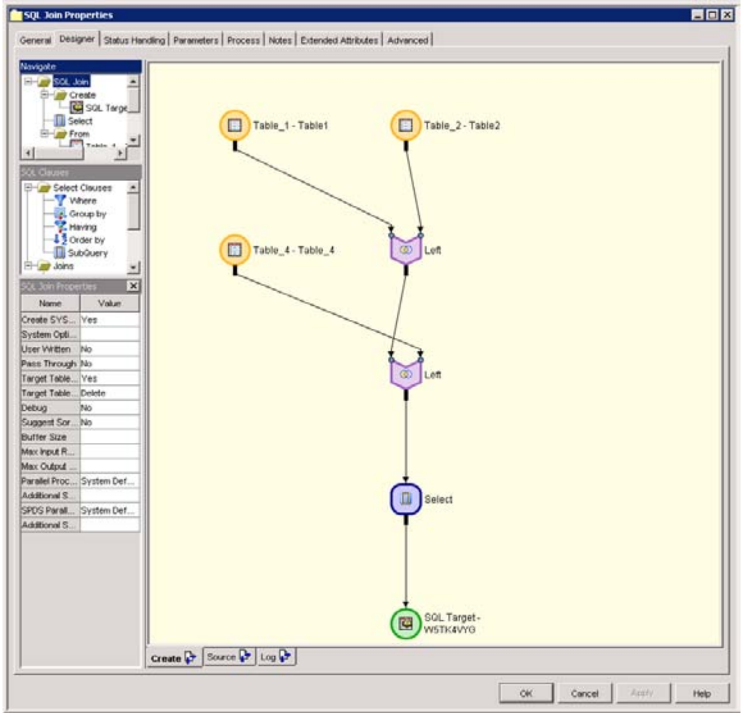
After completing a job containing a join of three or more tables you might find that it was necessary to change the order of the internal sub-joins in an attempt to minimize the behind the scenes sorts that SAS will use. This problem can be solved by splitting tables from the merge and carefully planning their reentry. So why should we bother with extract nodes? In this case, deleting linkages to the SQL node can have negative consequences - especially as jobs grow larger. Nodes can be lost and linkages can mysteriously disappear. Expressions can be literally destroyed… As well, we must take into consideration the time necessary to sort everything out. Breaking joins can have profound implications. What we would like is a way to reorder things without destroying the work that we have already done.

SAS 9.1.3, DI VERSION 3.4

Below is an example job in 9.1.3. Three tables are merged and then loaded into a fourth.

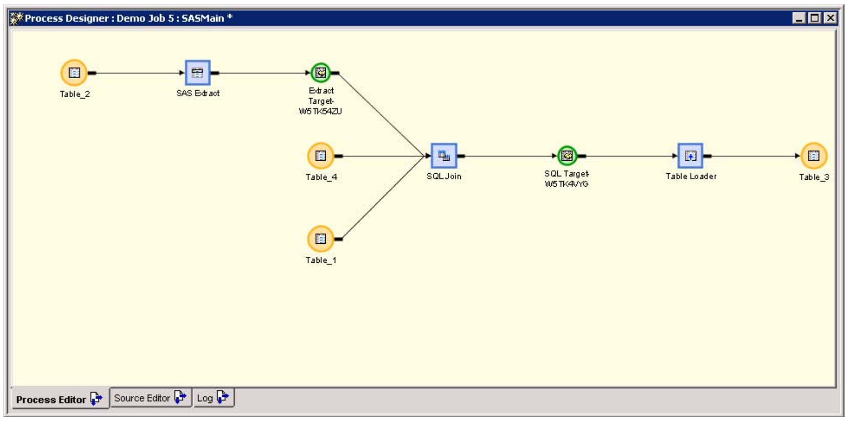


Double clicking on the SQL join node and then selecting the designer tab exposes the join specifics. As can be seen, Table_1 and Table_2 are joined, before the result is joined with Table_4. There is no way to manipulate the order of these joins from this screen. This is unfortunate, because the goal is to merge Table_4 with Table_1 first.

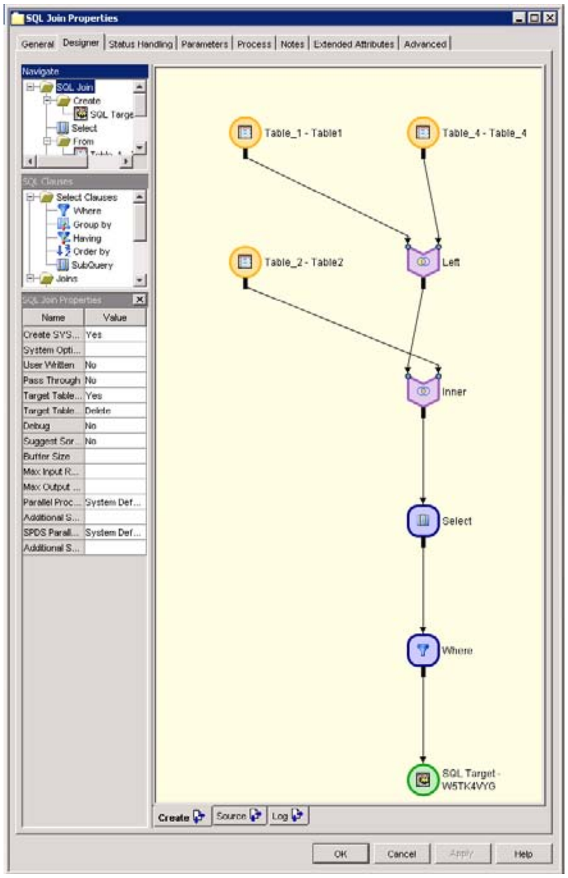


Name	Value
Create SYS...	Yes
System Opti...	
User Written	No
Pass Through	No
Target Table...	Yes
Target Table...	Delete
Debug	No
Suggest Sor...	No
Buffer Size	
Max Input R...	
Max Output ...	
Parallel Proc...	System Def...
Additional S...	
SPDS Parall...	System Def...
Additional S...	

Returning to the main menu, drop an extract node between Table_2 and the SQL node.

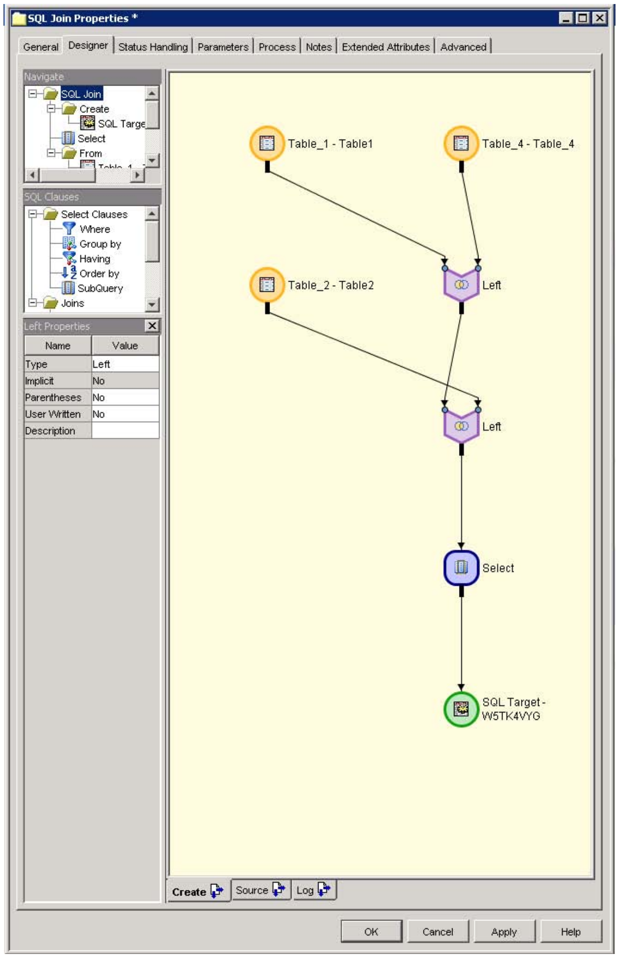


Now, delete the extract node and return to the designer page of the SQL join to see what has happened. Note that now Table_2 is being joined in second, but that it has gone from being a left join to an inner join. Consequently, a where clause node has also been added.



Name	Value
Create SYS...	Yes
System Opti...	
User Written	No
Pass Through	No
Target Table...	Yes
Target Table...	Delete
Debug	No
Suggest Sor...	No
Buffer Size	
Max Input R...	
Max Output ...	
Parallel Proc...	System Def...
Additional S...	
SPDS Parall...	System Def...
Additional S...	

Fixing the join structure leaves the state as so, with Table_2 and Table_4 swapped. Unfortunately, this destroys any kind of special join conditions that might have been set up, but it does not break the flow. You must redo the join logic and remove the where statement node by hand.

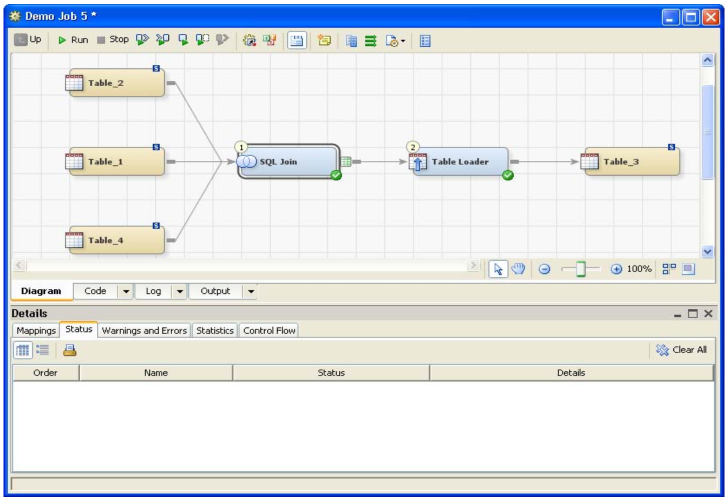


Name	Value
Type	Left
Implicit	No
Parentheses	No
User Written	No
Description	

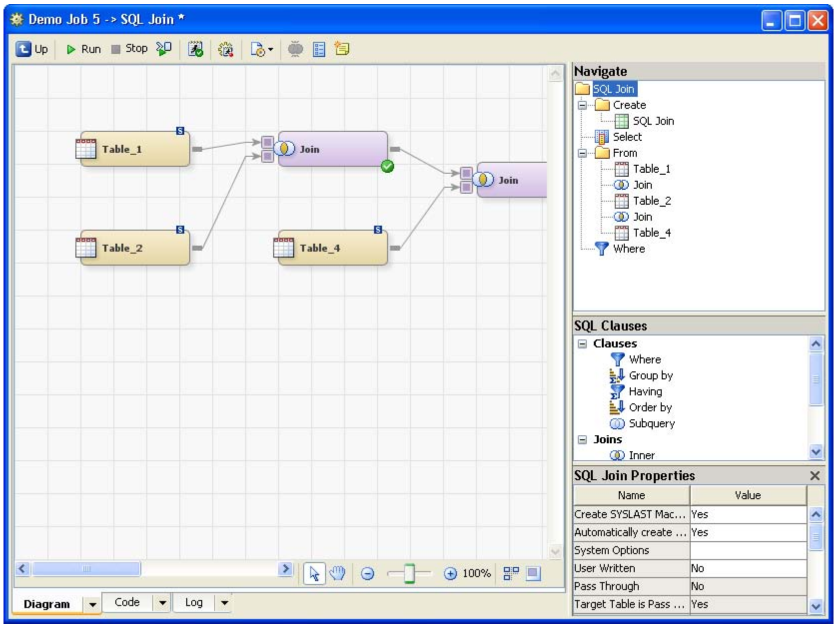
This illuminates a broader principle. The last table to enter a SQL node will be the last one joined in. This means that with careful planning, SQL joins can be designed correctly the first time. Alternatively, if something changes, this behavior can be triggered by placing and deleting extract nodes.

SAS 9.2, DI VERSION 4.2

This is the same job as before, but in SAS 9.2.

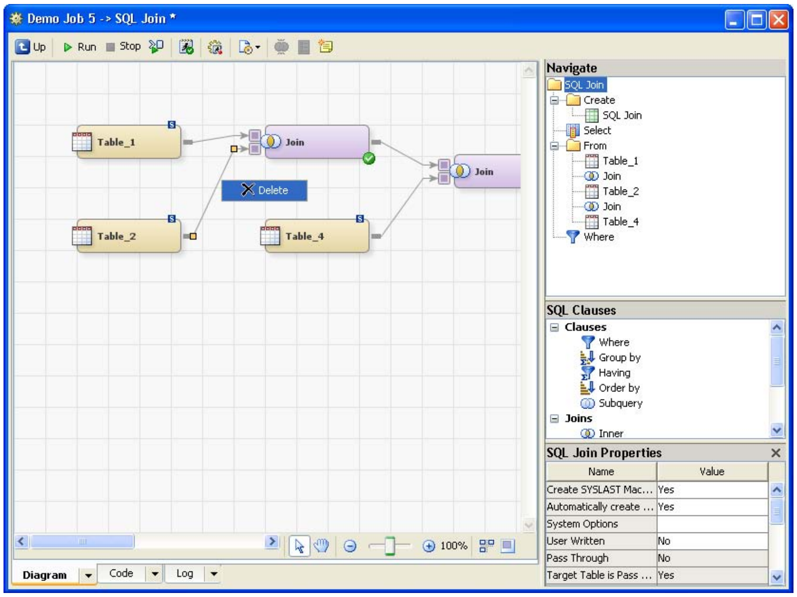


In SAS 9.2, configuring the order of the joins is done very simply. Double clicking the SQL node exposes the specifics of the join.



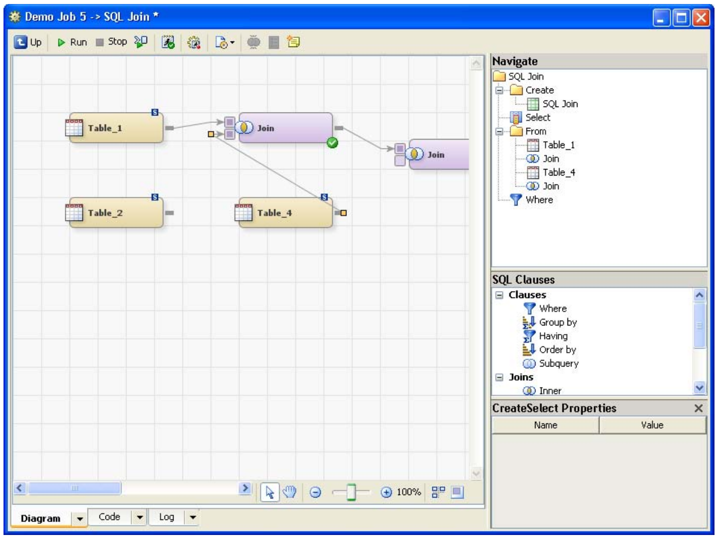
Name	Value
Create SYSLAST Mac…	Yes
Automatically create …	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass …	Yes

By right clicking on the line joining Table_2 to the Join node and choosing delete, one can remove the linkage.

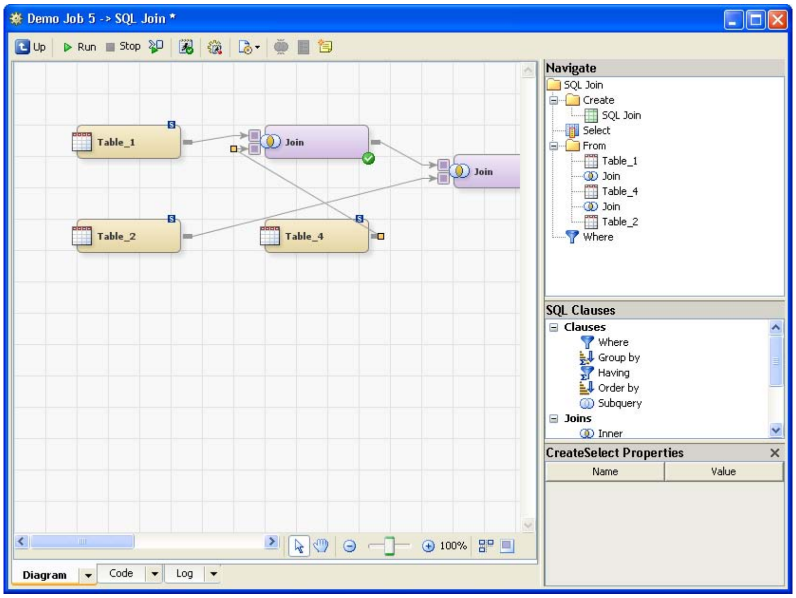


Name	Value
Create SYSLAST Mac…	Yes
Automatically create …	Yes
System Options	
User Written	No
Pass Through	No
Target Table is Pass …	Yes

Positioning the mouse near the node causes the cursor to transform into a 4 sided arrow. This tool allows one to move linkages.



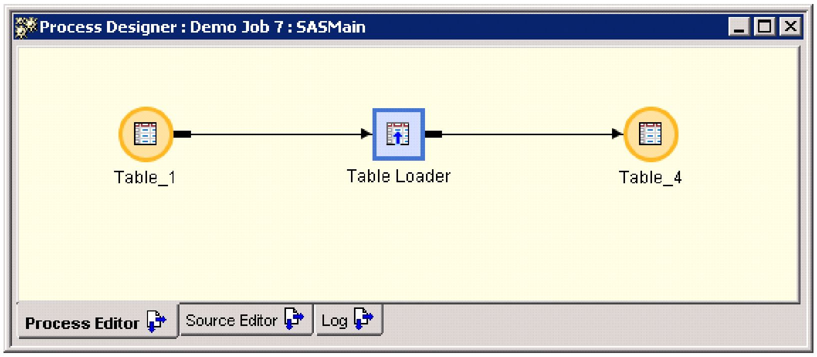
After pulling the linkage from Table_4 to the first join node, move the cursor towards the right edge of Table_2, where it becomes a pencil. Using the pencil tool draw the linkage to the second join node. Unfortunately, in the case of left joins, the where-clause statements will not be preserved.



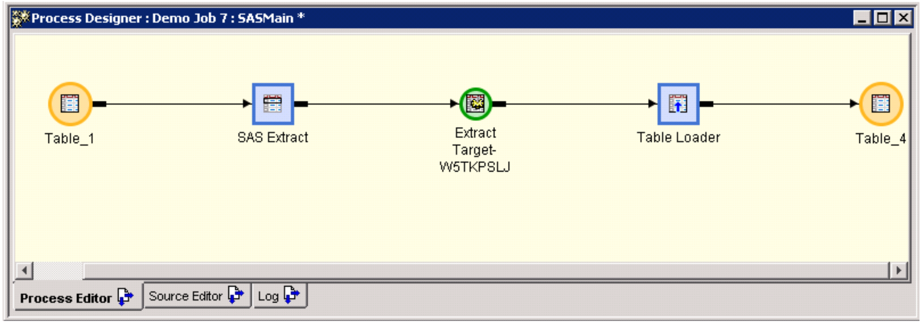
BUFFERING SELECTIONS AND MAPPINGS

A well placed extract node can operate as a buffer, allowing the mappings of the node after it to survive, even if structure of the job behind it were to change. This can be useful in preserving work if you combine it with the extract node’s ability to be dropped into, and taken out of most flows.

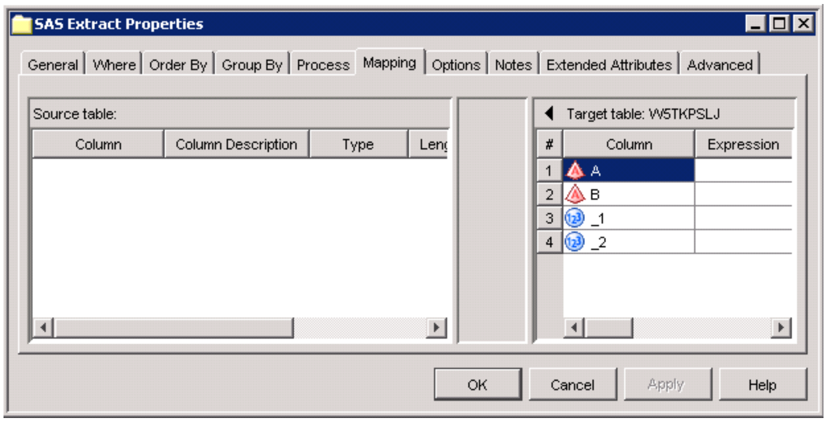
SAS 9.1.3, DI VERSION 3.4



In the simple job above, Table_1 is being directly loaded into Table_4. Table_1 and Table_4 do not share any column names, so all the mappings were manually defined. However, something is about to change in the job. Another table (which also does not share any mappings with Table_1) will be replacing Table_1. This could mean that all of those mapping would need redoing, and depending on the complexity of the expressions involved, this could be quite tedious. Alternatively, an extract node could be used.



The table loader's mappings are unperturbed, and if Table_1 is removed, all that happens to the extract is as follows:



Target table: W5TKPSLJ

#	Column	Expression
1	A	
2	B	
3	_1	
4	_2	

You can place anything you want feeding into the extract, map it to the values on the right, and the mappings would flow through to the table loader intact. Alternatively, if one were creating a larger structure, the end result of which directly mapped to the extract node's target, they could then erase the extract node, and things would continue to flow through.

SAS 9.2, DI VERSION 4.2

Since the flow from one node to another in SAS 9.2 is now user defined, things have changed. Essentially, if column names aren't shared between the two tables, as soon as the linkage between Table_1 and the Table Loader node is disrupted, the manually produced mappings are lost. One cannot use extract nodes to buffer in the same way that one could in 9.1.3.

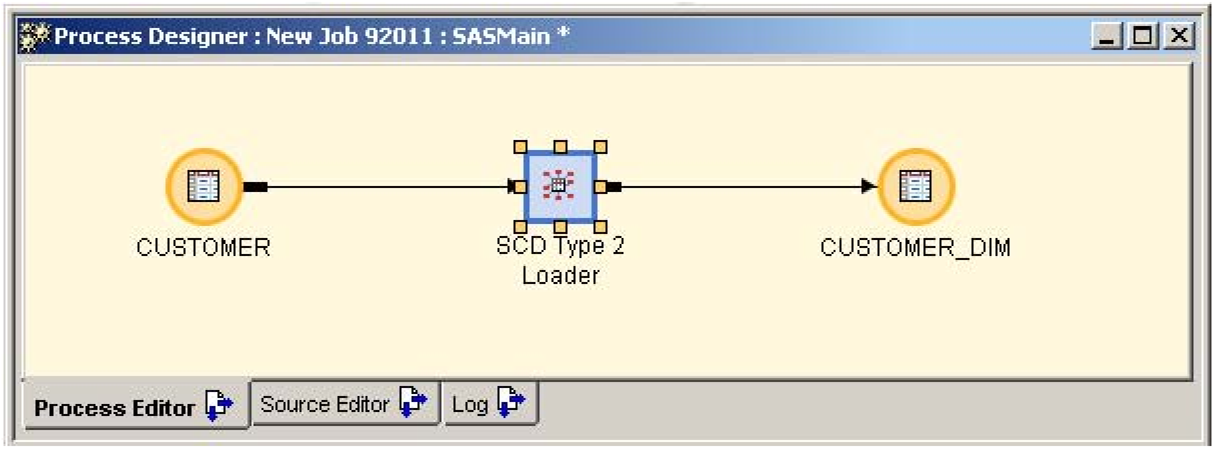
CUSTOM DATE ASSIGNMENT IN THE SCD TRANSFORM

THE SETUP

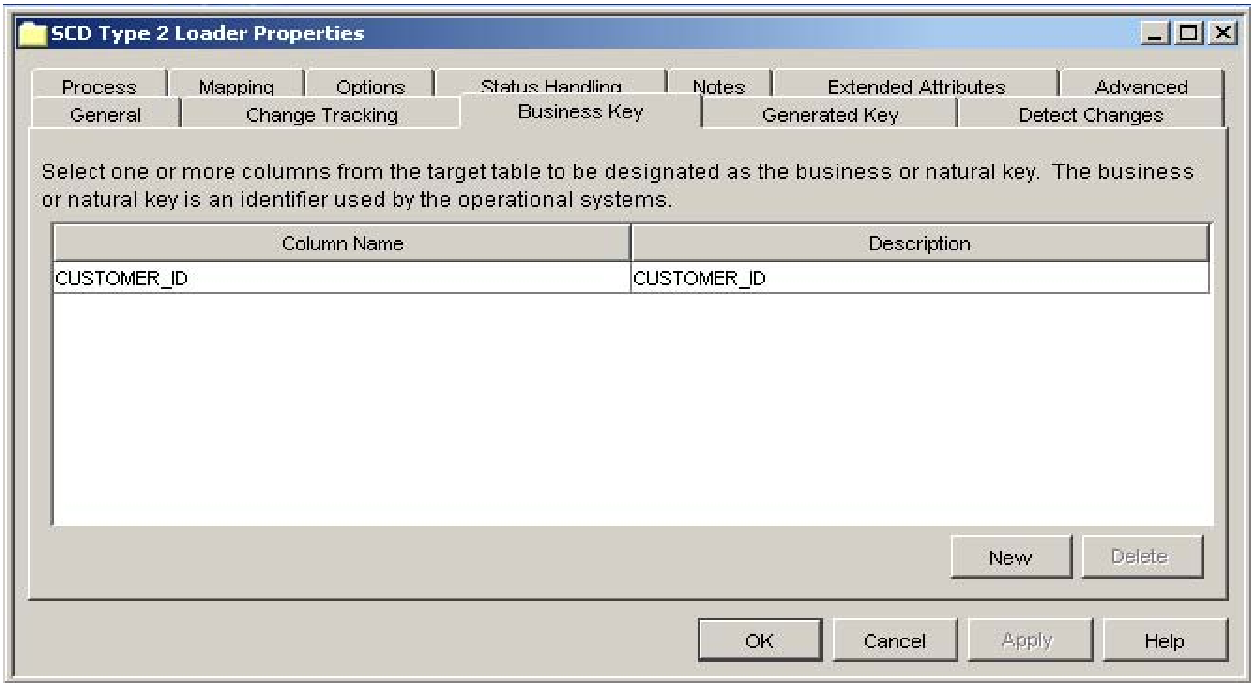
You would like to use the SCD transform to track effective dates, but you would like to assign a different start date for the first record. Often, this is result of including existing data into the slowly changing dimension. This data already has a predefined start time window. Using the default datetime() call does not allow setting the proper start date and time.

SAS 9.1.3, DI VERSION 3.4

The SCD transform has some issues. There are times when the code that it generates is not the most efficient or works the way you would like. In this case, we would like to assign different dates to the start date field tracking SCD's. The following is an example of a simple job that tracks slowly changing dimensions:

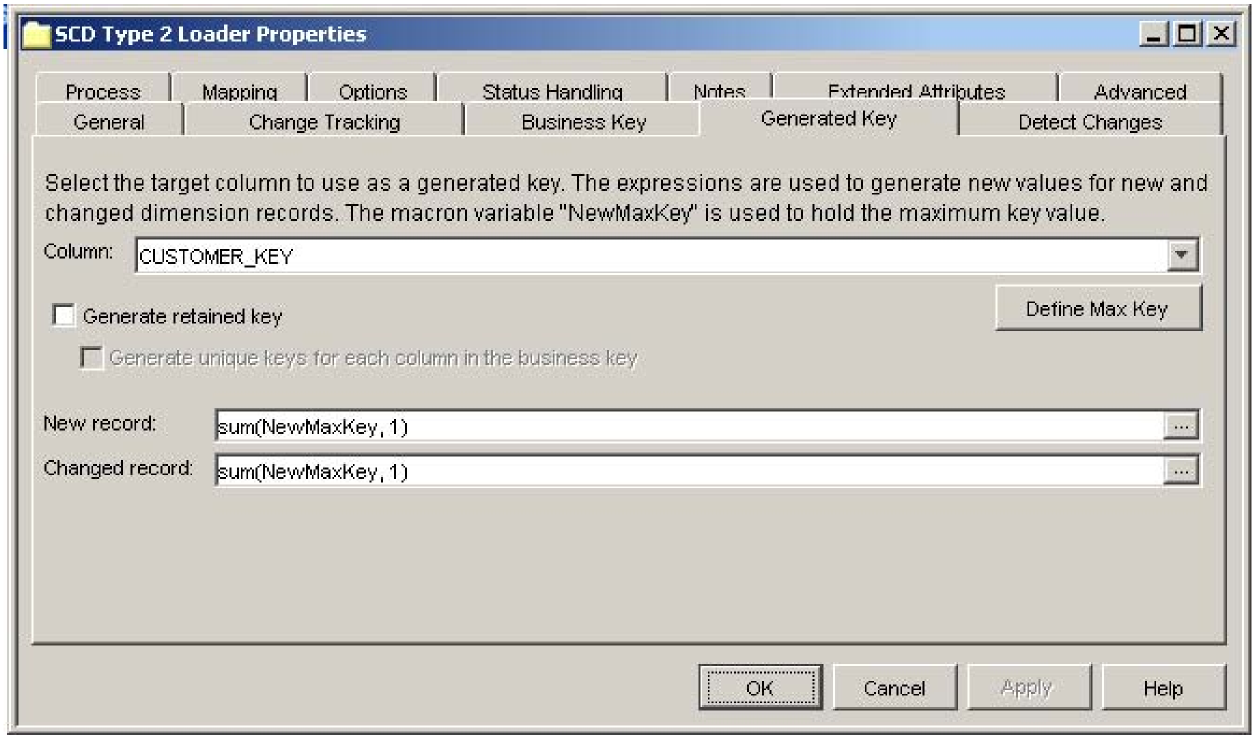


After setting the appropriate values we have the following for the options necessary to track an SCD:



Select one or more columns from the target table to be designated as the business or natural key. The business or natural key is an identifier used by the operational systems.

Column Name	Description
CUSTOMER_ID	CUSTOMER_ID



Select the target column to use as a generated key. The expressions are used to generate new values for new and changed dimension records. The macron variable "NewMaxKey" is used to hold the maximum key value.

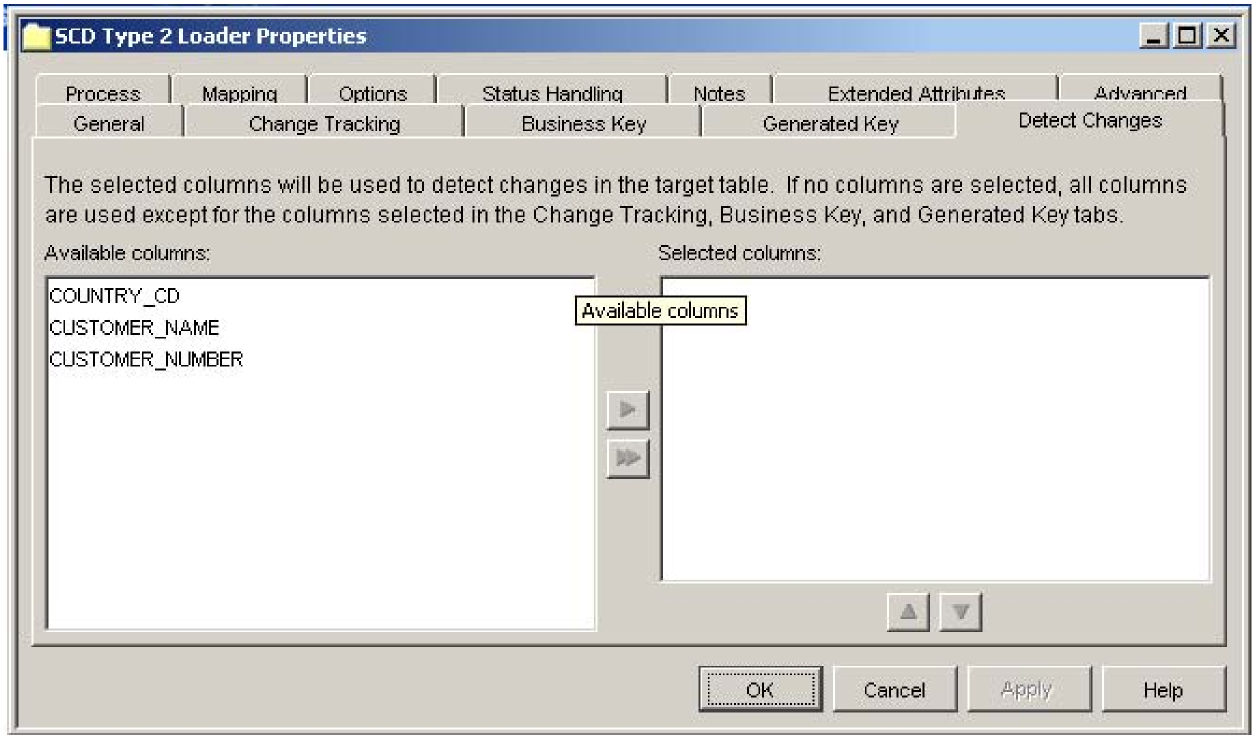
Column: CUSTOMER_KEY

☐ Generate retained key

☐ Generate unique keys for each column in the business key

New record: sum(NewMaxKey, 1)

Changed record: sum(NewMaxKey, 1)



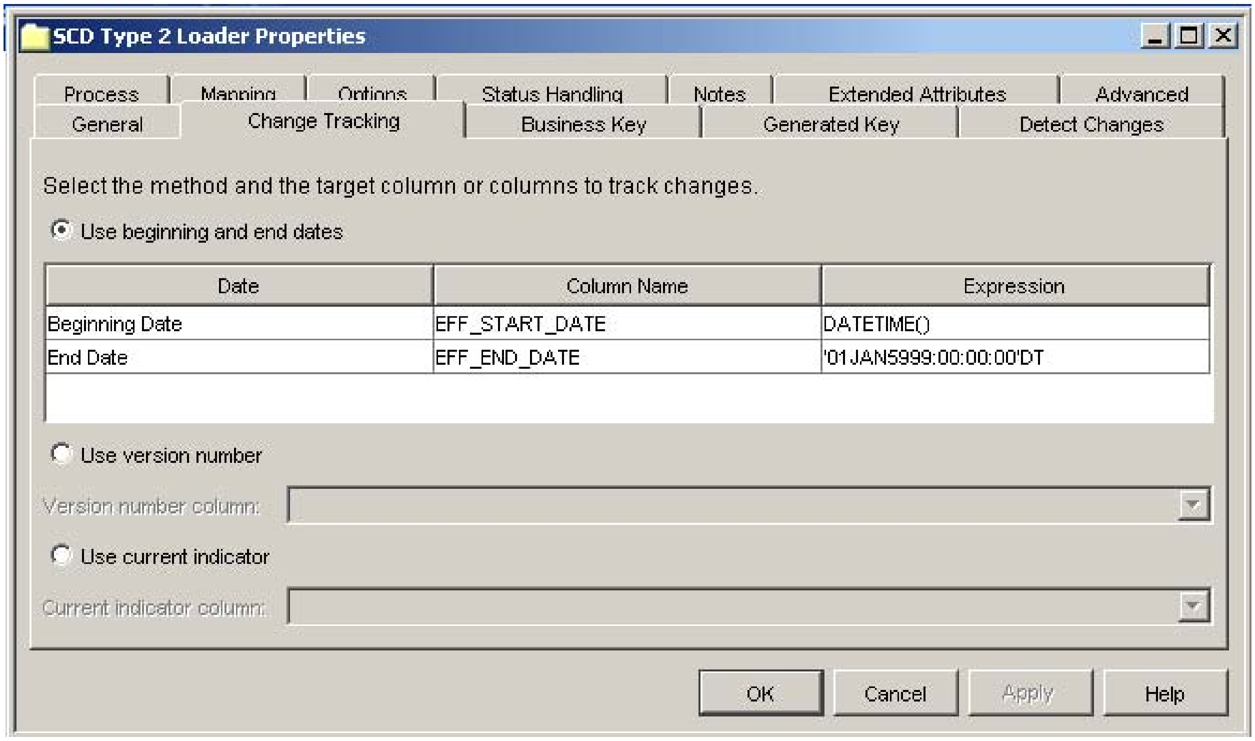
The selected columns will be used to detect changes in the target table. If no columns are selected, all columns are used except for the columns selected in the Change Tracking, Business Key, and Generated Key tabs.

Available columns:

COUNTRY_CD
CUSTOMER_NAME
CUSTOMER_NUMBER

Selected columns:

The issue at hand is the information on the Change Tracking panel. This panel gives the ability to assign the columns used in tracking the effective dates. It also allows you to override the logic used in assigning the values. The expression field on the right defaults to DATETIME(). This means that the effective start date for any record in the table will be set to the current time when the job is first run. In the case where we would like to set the start date to some date in the past for the first record this approach does not work.



Select the method and the target column or columns to track changes.

● Use beginning and end dates

Date	Column Name	Expression
Beginning Date	EFF_START_DATE	DATETIME()
End Date	EFF_END_DATE	'01JAN5999:00:00:00'DT

○ Use version number

Version number column:

○ Use current indicator

Current indicator column:

What we would like to do is replace the DATETIME() with an expression that allows us to switch on whether or not the record is the first record to be added, per brand new Business Key. To do this, we need to know when a record is brand new versus an update of an existing record. After doing some code spelunking, it turns out that you can take advantage of some generated code variables to let you know if the record is brand new or already exists. Note, this trick is what one might call a hack, but it works for us. Remove the comments before you assign to the expression:

```
/* Add this to the Pre and Post Options tab */
/* Needed to silence the DROP warnings       */
Options dkrOCond=noWarning;

/* replace DATETIME() with the following (or something similar) */
%str(

  /* if inSort and inXref do not exist then assign missing */
  ifn( missing(inSort) and missing(inXref), .,

    /* else, if inSort and not inXref then new record */
    ifn( inSort and not inXref,

      /* value for a new record */
      '01JAN1900:00:00:00'dt,

      /* value for an existing record */
      DATETIME()
    )
  );

  /* drop these created variables */
  /* needed for the 1st dataset, not the 2nd */
  drop inSort inXref;
);
```

SAS 9.2, DI VERSION 4.2

The code that is generated for an SCD is very similar to the 9.1.3 code. This tweak will work for 9.2 as well.

LEVERAGING EXTERNAL OPTIONS FILES TO CUSTOMIZE JOBS

THE SETUP

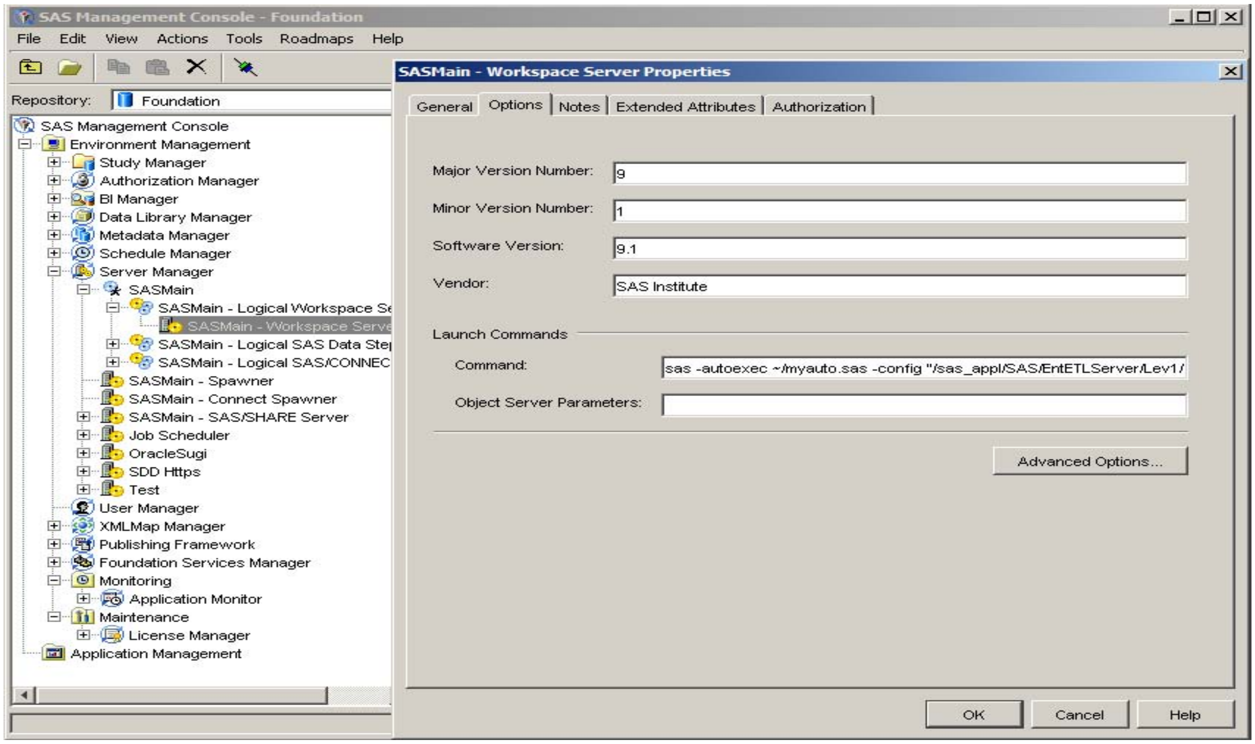
A DI job is mostly hard coded. That is, unless you use the built in parameterization options via an input data set, you are basically stuck with whatever hard coded values you have set in your job. Oftentimes however, we want to run a job with different setups – perhaps with user configurable date ranges – and we want to be able to changes inputs easily via a text editor. What we really want is something more akin to the –autoexec option. In this scenario I just pass in a file that has configuration information and the system does what is right.

SAS 9.1.3, DI VERSION 3.4

In Version 3.4, you can leverage auto-created macro variables in the job to switch on the repository you are running from. If you look at the generated code for a job you will see that the following macro variables are created:

```
/* Create metadata macro variables */
%let IOMServer      = %nrquote(SASMain);
%let metaPort       = %nrquote(8561);
%let metaServer     = %nrquote(localhost);
%let metaRepository = %nrquote(sasdemo);
```

The last macro variable is the repository that you are running the job under. If you are running from a Project repository then you will have the name of a project repository. If I then write some code that leverages this macro variable in a path to an options file I can load a set of macro variables that I can use to configure my job. Note, this is somewhat similar to modifying the SAS launch command in the SAS® Management Console. From the SMC, I can declare the SAS launch string to include a relative path autoexec. For instance,



Major Version Number: 9
Minor Version Number: 1
Software Version: 9.1
Vendor: SAS Institute
Command: sas -autoexec ~/myauto.sas -config "/sas_appl/SAS/EntETLServer/Lev1/
Object Server Parameters:

This would allow a per-person inclusion of the autoexec in the logged in users home directory. However, it requires that jobs run from a non-project repository setup to have their own Unix or Windows logins. For jobs run from the Foundation repository you would need a special account to run in production mode versus project mode.

Another solution is to use the &metaRepository macro variable that is defined for us. If you have access to the Workspace server, a Samba mapped directory (for Unix installations), or a mapped drive (for Windows installations) then you can write a macro that reads an external options file and auto-loads macro variables based on the repository we are running from. For instance, if you add the following to the “Pre and Post Process” tab for a job:

```
%eoptions(file=adw.opt,repo=&metaRepository)
```

And you had options files created in the following locations:

```
/samba_mount/configs/Foundation/adw.opt
/samba_mount/configs/ProjectRepos1/adw.opt
/samba_mount/configs/ProjectRepos2/adw.opt
```

Then you could create macro variables using by reading the directives of the following form:

```
# Sample config file. All values here will be created as macro vars in the
# referenced SAS session

# controls the initial pull of data and number of months
OPT_RANGE_START  = 2009/10/01
OPT_RANGE_MONTHS = 3

# Start and end date for Period dimension
OPT_PERIOD_START = '01JAN1900'd
OPT_PERIOD_END   = '31DEC2040'd

# Start and End for Age Sex dimension
OPT_AGE_INCS  = 0.5 1.5
OPT_AGE_START = 0
OPT_AGE_END   = 130
```

These macro variables can be used anywhere in the job. The macro itself must be stored in a location that is part of the macro auto-call concatenation. The easiest place to store the macro is in the following location. This location is by default part of auto-call path concatenation for a workspace server

```
SAS/EntBIServer/Lev1/SASApp/SASEnvironment/SASMacro
```

See the appendix for the eoptions() macro.

SAS 9.2, DI VERSION 4.2

In DI Studio 4.2, the metaphor is changed. Project repositories do not exist in the same form as they did in 9.1.3. Instead, you get macro variables generated in the following format:

```
/* General macro variables  */
%let jobID = %quote(A5M82C5K.A7000003);
%let etls_jobName = %nrquote(Test Job);
%let etls_userID = %nrquote(colinger@d-wise);
```

Here, the setup is very much like our –autoexec example. That is, you only have the user account to switch on. In this case, you could use either the –autoexec trick or change the eoptions() macro to read from a path generated from a user account name. Either way, a flat file editable from a text editor is a much easier parameter import source than a parameterized job.

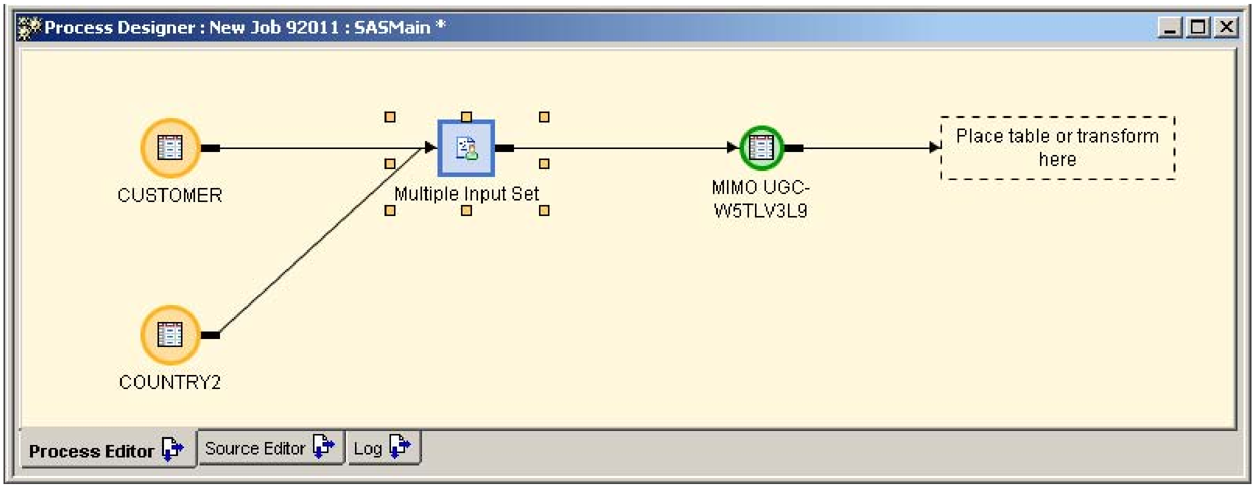
USING EMPTY TRANSFORMS TO SET MACRO VARIABLES

THE SETUP

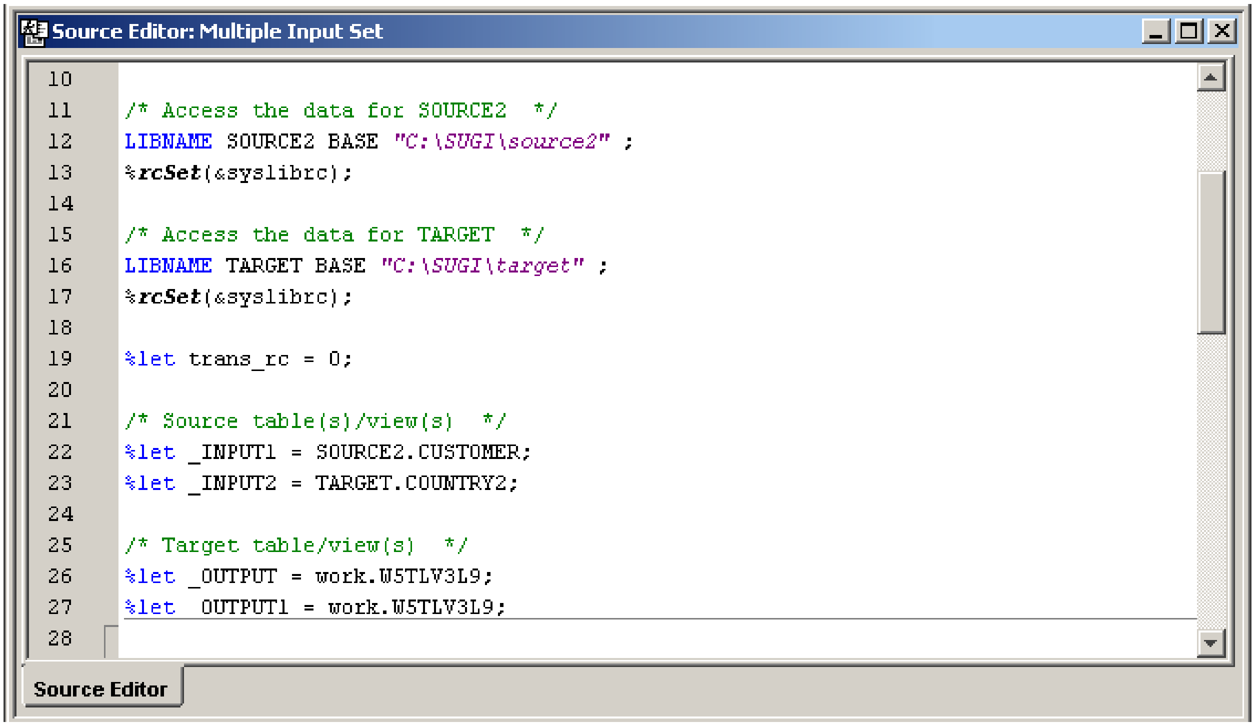
Suppose you create a multi-input custom transform. You decide it is going to take two inputs, and will be populated with custom code to append the two datasets together with a multi-set data step. How do you accomplish this using the auto-created macro variables for _INPUT1-_INPUT2?

SAS 9.1.3, DI VERSION 3.4

In DI 3.4 this scenario is broken. If you create a custom transformation and add your own code then your code must create the input macro variables. The code that auto-generates the macro variables is not run for multiple inputs. For instance:

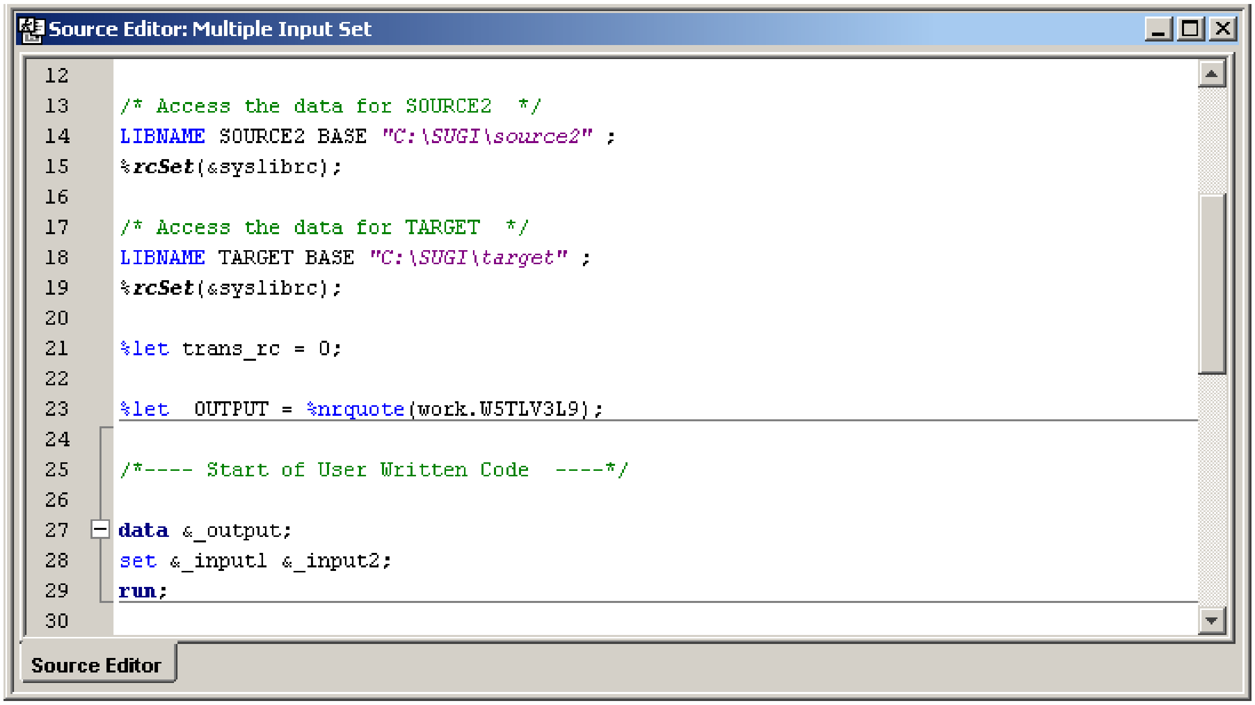


If the Process for the transform is set to “Automatically Create Source Code” then you get the following:



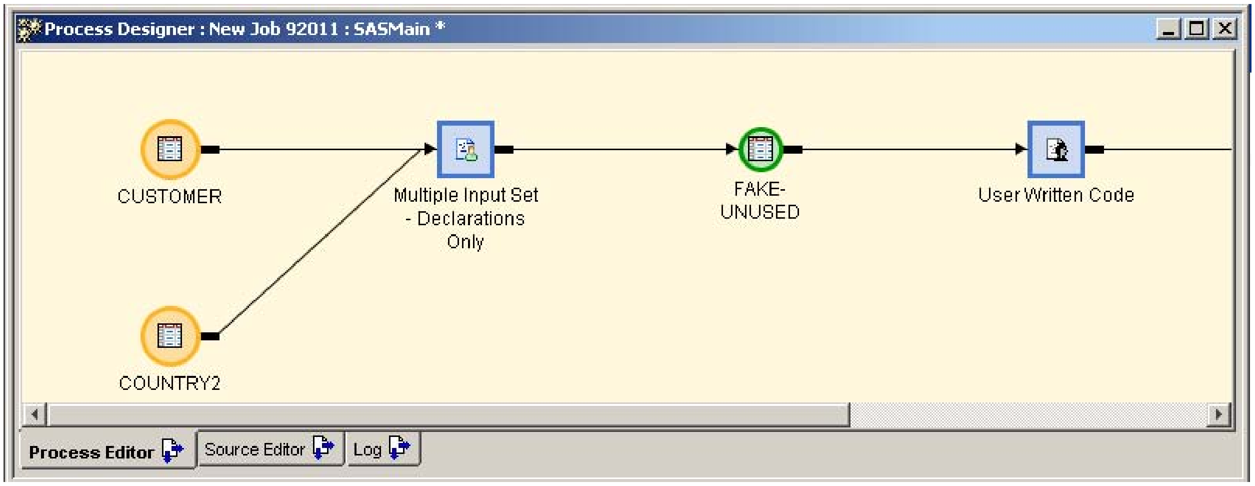
```

/* Access the data for SOURCE2  */
LIBNAME SOURCE2 BASE "C:\SUGI\source2" ;
%rcSet(&syslibrc);

/* Access the data for TARGET  */
LIBNAME TARGET BASE "C:\SUGI\target" ;
%rcSet(&syslibrc);

%let trans_rc = 0;

/* Source table(s)/view(s)  */
%let _INPUT1 = SOURCE2.CUSTOMER;
%let _INPUT2 = TARGET.COUNTRY2;

/* Target table/view(s)  */
%let _OUTPUT = work.W5TLV3L9;
%let _OUTPUT1 = work.W5TLV3L9;

```

Notice that both Input and Output macros are created. If, however, you choose to add your own code to the transform via the “User Written” option then you get the following:

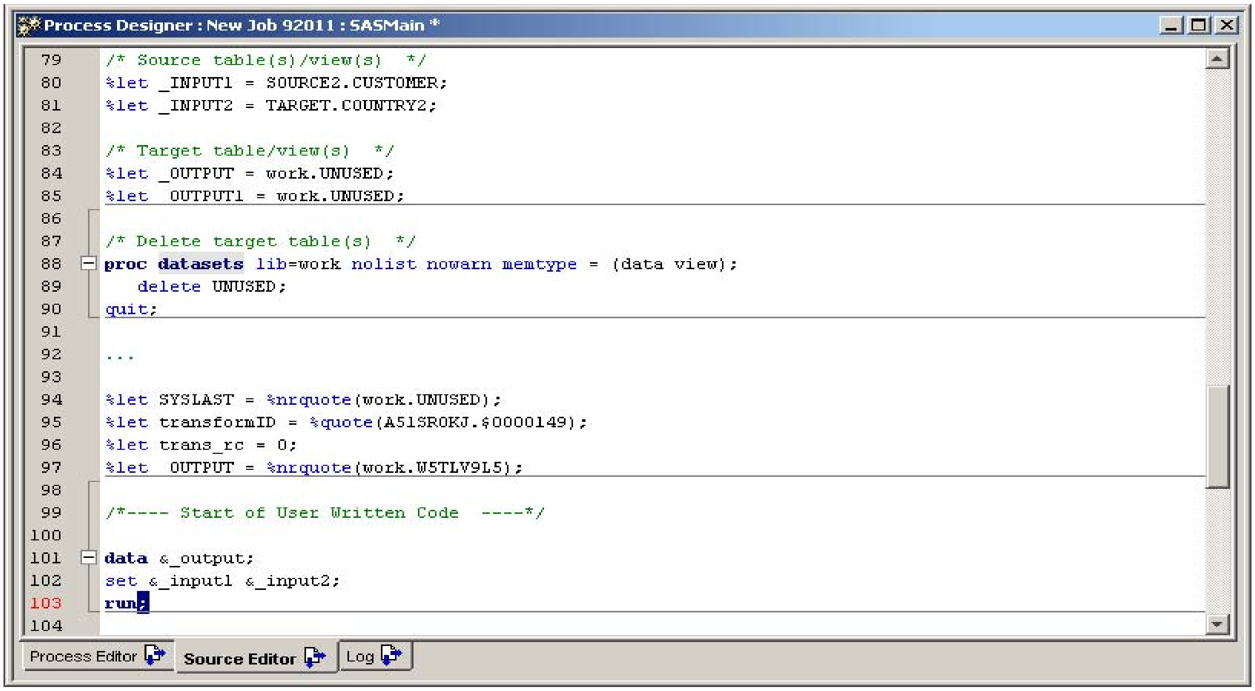


```
/* Access the data for SOURCE2  */
LIBNAME SOURCE2 BASE "C:\SUGI\source2" ;
%rcSet(&syslibrc);

/* Access the data for TARGET  */
LIBNAME TARGET BASE "C:\SUGI\target" ;
%rcSet(&syslibrc);

%let trans_rc = 0;

%let  OUTPUT = %nrquote(work.W5TLV3L9);

/*---- Start of User Written Code  ----*/

data &_output;
set &_input1 &_input2;
run;
```

And the Input macro variables have disappeared. You can get around this by using the Multi-Input transform just to declare the macro variables and then following that node with a User Written code node. For instance,


Process Designer : New Job 92011 : SASMain *
CUSTOMER
COUNTRY2
Multiple Input Set - Declarations Only
FAKE-UNUSED
User Written Code
Process Editor
Source Editor
Log

The generated code would then look like this:

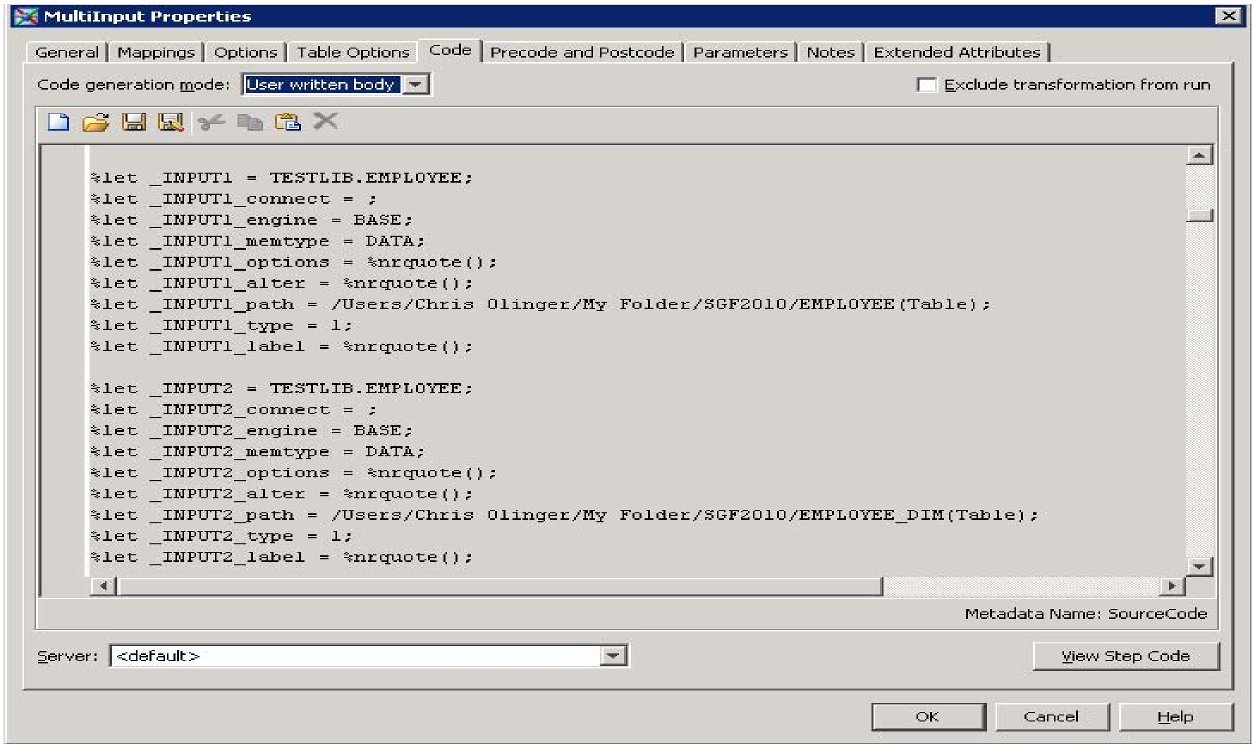


```
/* Source table(s)/view(s)   */
%let _INPUT1 = SOURCE2.CUSTOMER;
%let _INPUT2 = TARGET.COUNTRY2;

/* Target table/view(s)   */
%let _OUTPUT = work.UNUSED;
%let _OUTPUT1 = work.UNUSED;

/* Delete target table(s)   */
proc datasets lib=work nolist nowarn memtype = (data view);
   delete UNUSED;
quit;


...

%let SYSLAST = %nrquote(work.UNUSED);
%let transformID = %quote(A51SR0KJ.$0000149);
%let trans_rc = 0;
%let _OUTPUT = %nrquote(work.W5TLV9L5);

/*---- Start of User Written Code  ----*/

data &_output;
set &_input1 &_input2;
run;

```

Basically, you have used two nodes to fake DI Studio into doing what it should have done to start with.

SAS 9.2, DI VERSION 4.2

There is good news on this front. DI Studio 4.2 fixes all of these code generation issues. In fact, you can tell DI Studio if just the body of the code should be user written, or all of it is. The generated code also contains much more information in macro variables than in DI Studio 3.4. You have options for variables that will be kept, engine information on the table, and other header information like labels and paths. As well, macro variables are created for every column in the mapping!



Code generation mode: User written body

☐ Exclude transformation from run

```

%let _INPUT1 = TESTLIB.EMPLOYEE;
%let _INPUT1_connect = ;
%let _INPUT1_engine = BASE;
%let _INPUT1_memtype = DATA;
%let _INPUT1_options = %nrquote();
%let _INPUT1_alter = %nrquote();
%let _INPUT1_path = /Users/Chris Olinger/My Folder/SGF2010/EMPLOYEE(Table);
%let _INPUT1_type = 1;
%let _INPUT1_label = %nrquote();

%let _INPUT2 = TESTLIB.EMPLOYEE;
%let _INPUT2_connect = ;
%let _INPUT2_engine = BASE;
%let _INPUT2_memtype = DATA;
%let _INPUT2_options = %nrquote();
%let _INPUT2_alter = %nrquote();
%let _INPUT2_path = /Users/Chris Olinger/My Folder/SGF2010/EMPLOYEE_DIM(Table);
%let _INPUT2_type = 1;
%let _INPUT2_label = %nrquote();
```

Metadata Name: SourceCode

Server: <default>

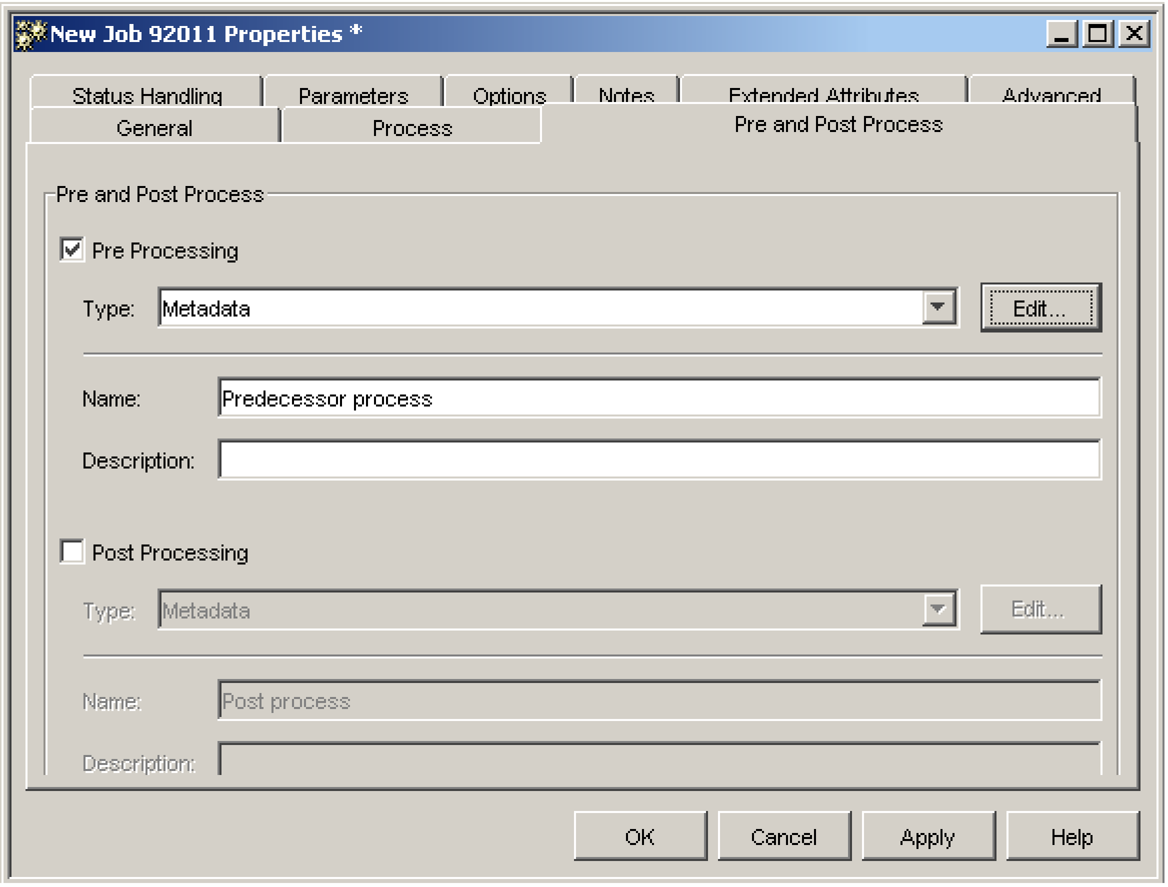
MANAGING MACROS AND FORMATS

THE SETUP

You want to add macros or formats to the system for an individual job or for all jobs.

SAS 9.1.3, DI VERSION 3.4

On an individual job basis, add any macros or formats to the ”Pre and Post Process” area of the job:



New Job 92011 Properties *

Pre and Post Process

☑ Pre Processing

Type: Metadata

Name: Predecessor process

Description:

☐ Post Processing

Type: Metadata

Name: Post process

Description:

For a set of global macros or formats, your best bet is adding them to the standard install directories for the Workspace server:

```
SAS/EntBIServer/Lev1/SASApp/SASEnvironment/SASMacro
SAS/EntBIServer/Lev1/SASApp/SASEnvironment/SASFormats
```

You will need an Administrator that has access to these locations. If you are using the –autoexec trick (~/autoexec.sas) described earlier then you can add a personal directory under your home location that can store your global macro variables (if you have login authority to the server).

For formats, you can build your formats to the standard format library (in the SASFormats directory) by specifying:

```
proc format lib=library;
…
run;
```

In order to make this work, you need to run as the SAS installer account. This may not be possible unless you are friends with the SAS administrator. However, once the formats are part of the standard library the system will work much better.

SAS 9.2, DI VERSION 4.2

There is no change in macro support for DI Studio 4.2. The same rules as above apply.

MANAGING METADATA USING AN EXTERNAL SOURCE

THE SETUP

You would like to use MS Excel or some other tool to manage metadata outside of DI Studio. You would like to be able to create an initial table set and load the metadata into the system.

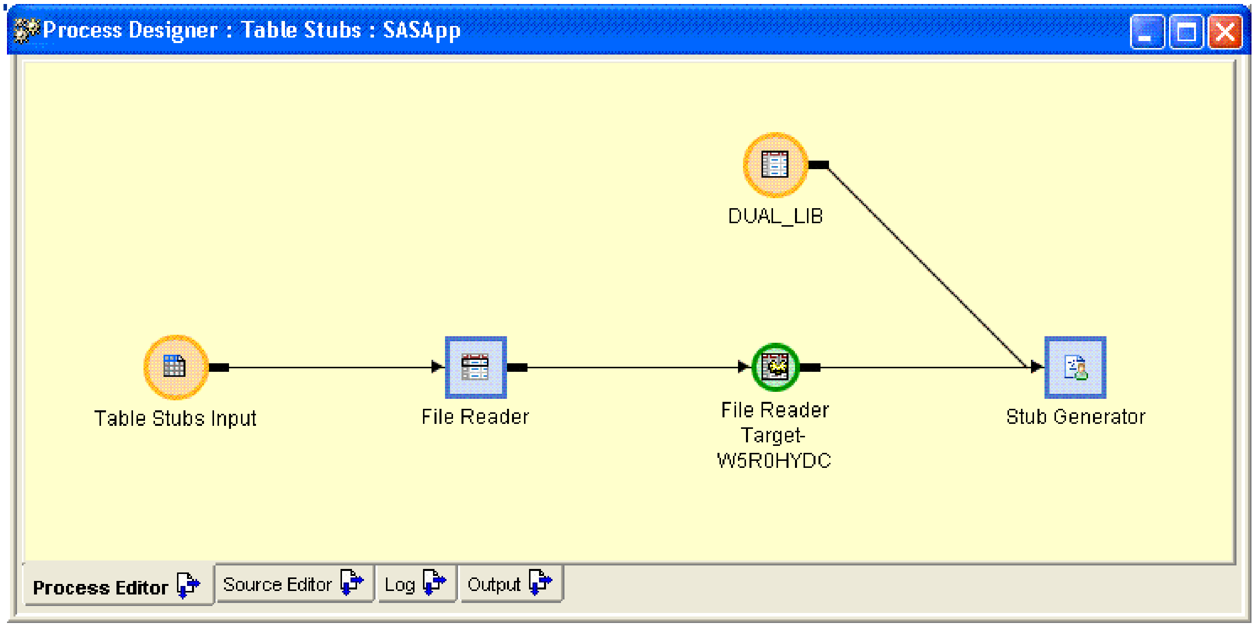
Note, the solution to this problem is a non-trivial task. As such, there are incremental ways of looking at this, each with increasing difficulty:

1) You would like to create initial table stubs (0 record tables) that then you can load metadata from using the standard Source Designer
2) After initial creation, you would like to download existing metadata to an external tool, make changes, and then re-upload the changed metadata
3) You would like have a DI Studio plugin that manages this process for you

We are going to talk about option 1, as options 2 and 3 are outside the scope of this paper. We have built and deployed solutions for options 2 and 3 using both Java and SAS to manage the metadata. These efforts are time consuming but they can be done, and are worth it if you want to invest the resources. If you are interested in more details then please contact us.

SAS 9.1.3, DI VERSION 3.4

The best way to tackle this problem is to use the transform generator to create a 'Stubs' generator. This transform will take a spreadsheet as input along with an input table that you can use to reference the appropriate libname to write to. For instance:



Notice that there are no outputs from the Stub Generator transform. The idea is that the SAS library of the input table DUAL_LIB will be used to build all tables that are described in the ‘Table Stubs Input’ source. To get the library in the Stubs code, simply parse the input _INPUT2 variable:

```
%let lib=%scan(&_INPUT2,1,.);
%let mem=%scan(&_INPUT2,2,.);
```

Using this trick means that you can add a table reference to the job stream only for the reference to the libname – which, by the way, causes the libname to be generated in the actual code. Simply ignore any columns in any mapping that came from the dummy table.

You can now use &lib to create a set of tables that are written to the DUAL_LIB location. The input source (Excel in this case) would possibly look like:

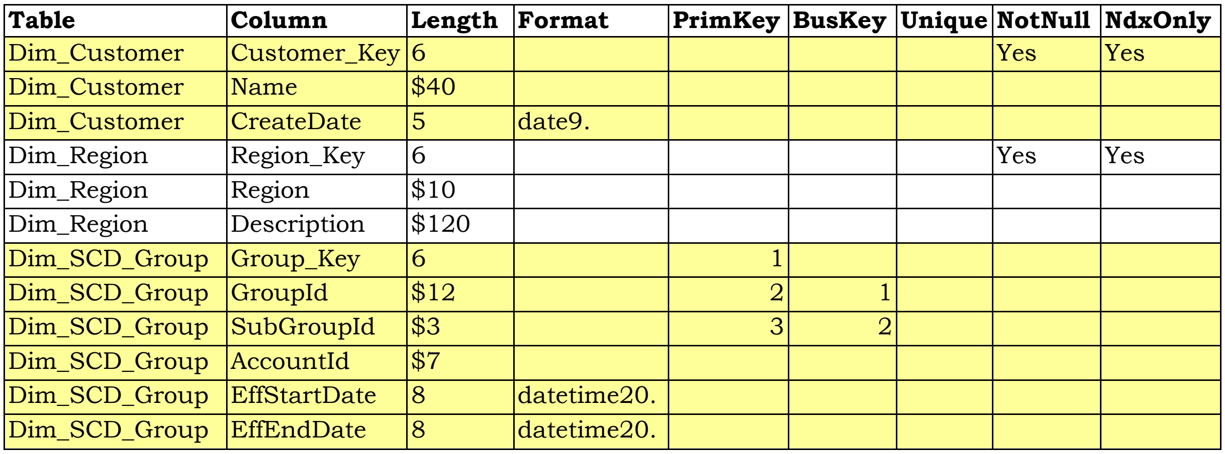


Table	Column	Length	Format	PrimKey	BusKey	Unique	NotNull	NdxOnly
Dim_Customer	Customer_Key	6					Yes	Yes
Dim_Customer	Name	$40						
Dim_Customer	CreateDate	5	date9.					
Dim_Region	Region_Key	6					Yes	Yes
Dim_Region	Region	$10						
Dim_Region	Description	$120						
Dim_SCD_Group	Group_Key	6		1				
Dim_SCD_Group	GroupId	$12		2	1			
Dim_SCD_Group	SubGroupId	$3		3	2			
Dim_SCD_Group	AccountId	$7						
Dim_SCD_Group	EffStartDate	8	datetime20.					
Dim_SCD_Group	EffEndDate	8	datetime20.					

The stub generator will read through these records and create table stubs using CALL EXCECUTE commands. After the tables have been created, use the Source Generator to create the metadata for the tables. Going forward, any metadata can be hand managed or you can recreate the stubs (wiping out any tables) and use ‘Update Table Metadata’ to refresh the table metadata.

See the appendix for the code for the sub generator.

SAS 9.2, DI VERSION 4.2

In DI 4.2, the Source Generator is replaced by the “Register Table” action off of the File menu. Other than that, this technique should still work.

CONCLUSION

SAS® Data Integration Studio is a powerful tool for building warehouses and processing data. However, a certain amount of clever work-arounds are necessary to smooth the edges. We encourage all of you to upgrade to Version 4.2 as in our experience it has proven to be superior is most cases.

We hope these tips and tricks prove as useful to you as they have to us.

ACKNOWLEDGMENTS

Hats off to the DI Studio team for Version 4.2. A much needed improvement.

CONTACT INFORMATION

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

Name: Chris Olinger
Enterprise: d-Wise Technologies, Inc.
Address: 4020 Westchase Blvd, Suite 527
City, State ZIP: Raleigh, NC, 27607
Work Phone: 919-600-6235
E-mail: colinger@d-wise.com
Web: http://www.d-wise.com

Name: David Kratz
Enterprise: d-Wise Technologies, Inc.
Address: 4020 Westchase Blvd, Suite 527
City, State ZIP: Raleigh, NC, 27607
Work Phone: 919-600-6233
E-mail: dkratz@d-wise.com
Web: http://www.d-wise.com

APPENDIX

EOPTIONS MACRO

```
/* macro to read an external file and create macro variables */

%macro eoptions(file=,
  base=/samba_mount_point/Configs,
  repo=,
  delim=/,
  extopts=EXTOPTS,
  prefix=OPT_
);

%if %length(%quote(&file)) %then %do;

  %local f;

  %let f=&base;
  %if %length(%quote(&repo)) %then %do;
    %if %length(%quote(&f)) %then
      %let f=&f&delim&repo;
    %else
      %let f=&repo;
  %end;
  %if %length(%quote(&f)) %then
     %let f=&f&delim&file;
  %else
     %let f=&file;

  %if %sysfunc(fileexist(&f)) %then %do;
     filename opts "&f";
     data &extopts;
       length buf $1032 option $32 value $1000;
       infile opts lrecl=2000;
       input;
       if _N_ = 1 then do;
         retain re;
         re = prxparse('/^([A-Za-z_][A-Za-z0-9_]*)\s*=\s*(.*?)\s*$/');
         if missing(re) then do;
            putlog "ERROR: Invalid regexp: parseOptions";
            stop;
         end;
       end;
       buf=strip(_infile_);
       if prxmatch(re, buf) then do;
         call prxposn(re, 1, pos, len);
         option = substr(buf, pos, len);
         option = upcase(option);
         if lengthn(option) < %length(%quote(&prefix))
         or substr(option,1,%length(%quote(&prefix))) ne upcase("&prefix") then
           option = upcase("&prefix")||left(option);
         call prxposn(re, 2, pos, len);
         value = substr(buf, pos, len);
         drop buf re pos len;
         output;
       end;
     run;
     filename opts;
  %end;
  %else %do;
```

```
     %put NOTE: External option file does not exist: &f;
  %end;

%end;

  * create the macro vars;
%if %sysfunc(exist(&extopts)) %then %do;
   /* possibly delete macro vars here to flush old ones */
   data _null_;
     set &extopts;
     call symputx(option,value,'G');
   run;
%end;
%else %do;
   %put NOTE: Options data set does not exist. No options loaded.;
%end;

%mend;
```

TABLE STUBS MACRO

```
/* Macro vars, table options referenced
   Table Name:              _SGTABLE
   Table Filter:            _SGFILTER
   Column Name:             _SGCOLNAME
   Column Length:           _SGCOLLEN
   Column Format:           _SGCOLFMT
   Column Informat:         _SGCOLNFMT
   Primary Key:             _SGPK
   Index Only:              _SGIO
   Business Key:            _SGBK
   Unique:                  _SGUN
   Not Null:                _SGNN
*/

%macro generateStubs;

 %local _SGTLIB;

 %* libname is the first arg of the dummy input;
 %let _SGTLIB=%SCAN(&_INPUT2,1,.);

 proc sort data=&_INPUT1 out=_tbl;
   by &_SGTABLE
   ;
 run;

 /* first delete any existing table */
 data _null_;
   set _tbl end=eof;
   by &_SGTABLE;
   %if (%quote(&_SGFILTER) ne) %then %do;
   where &_SGTABLE in (%unquote(&_SGFILTER));
   %end;
   if _n_=1 then do;
     call execute("proc datasets nolist lib=&_SGTLIB.;");
   end;

   if first.&_SGTABLE then do;
     call execute("delete "||strip(&_SGTABLE)||';');
   end;

   if eof then do;
     call execute("run; quit;");
   end;
 run;
```

```
/* create the tables */
data _null_;
   set _tbl;
   by &_SGTABLE;
   %if (%quote(&_SGFILTER) ne) %then %do;
   where &_SGTABLE in (%unquote(&_SGFILTER));
   %end;
   if first.&_SGTABLE then do;
     call execute("data &_SGTLIB.."||strip(&_SGTABLE)||';');
   end;

   /* process the columns */
   call execute('length '||strip(&_SGCOLNAME)||' '||strip(&_SGCOLLEN)||';');

   %if  (%quote(&_SGCOLFMT) ne) %then %do;
   if &_SGCOLFMT ne ' ' then do;
      call execute('format '||strip(&_SGCOLNAME)||' '||strip(&_SGCOLFMT)||';');
   end;
   %end;
   %if  (%quote(&_SGCOLNFMT) ne) %then %do;
   if &_SGCOLNFMT ne ' ' then do;
      call execute('informat '||strip(&_SGCOLNFMT)||' '||strip(&_SGCOLNFMT)||';');
   end;
   %end;

   if last.&_SGTABLE then do;
     call execute("stop; run;");
   end;
 run;

 %* primary keys;
 %if %length(%quote(&_SGPK)) %then %do;

   proc sql noprint;
     create table _pk as
       select * from _tbl
        where %unquote(&_SGPK) is not null
        %if (%quote(&_SGFILTER) ne) %then %do;
          and &_SGTABLE in (%unquote(&_SGFILTER))
        %end;
        order by %unquote(&_SGTABLE), %unquote(&_SGPK)
       ;
   quit;

   data _null_;
     set _pk end=eof;
     by &_SGTABLE;

     if _n_ = 1 then do;
       call execute("proc datasets lib=&_SGTLIB. nolist;");
     end;

     if first.&_SGTABLE then do;
       call execute("modify "||strip(&_SGTABLE)||"; ic create PRIMARY KEY (");
     end;

     call execute(strip(&_SGCOLNAME)||' ');

     if last.&_SGTABLE then do;
       call execute("); run;");
     end;

     if eof then do;
       call execute("quit;");
     end;
   run;

   proc datasets nolist lib=work; delete _pk; run; quit;

 %end;
```

```
%* not null;
%if %length(%quote(&_SGNN)) %then %do;
  data _null_;
    set _tbl end=eof;
    by &_SGTABLE;
    where upcase(&_SGNN) = 'YES'
    %if (%quote(&_SGFILTER) ne) %then %do;
      and &_SGTABLE in (%unquote(&_SGFILTER));
    %end;
    ;
    if _n_ = 1 then do;
      call execute("proc datasets lib=&_SGTLIB. nolist;");
    end;

    if first.&_SGTABLE then do;
      call execute("modify "||strip(&_SGTABLE)||";");
    end;

    call execute("ic create NOT NULL ("||strip(&_SGCOLNAME)||');');

    if last.&_SGTABLE then do;
      call execute("run;");
    end;

    if eof then do;
      call execute("quit;");
    end;
  run;
%end;

%* unique;
%if %length(%quote(&_SGUN)) %then %do;
  data _null_;
    set _tbl end=eof;
    by &_SGTABLE;
    where upcase(&_SGUN) = 'YES'
    %if (%quote(&_SGFILTER) ne) %then %do;
      and &_SGTABLE in (%unquote(&_SGFILTER));
    %end;
    ;
    if _n_ = 1 then do;
      call execute("proc datasets lib=&_SGTLIB. nolist;");
    end;

    if first.&_SGTABLE then do;
      call execute("modify "||strip(&_SGTABLE)||";");
    end;

    call execute("ic create UNIQUE ("||strip(&_SGCOLNAME)||');');

    if last.&_SGTABLE then do;
      call execute("run;");
    end;

    if eof then do;
      call execute("quit;");
    end;
  run;
%end;

%* index only;
%if %length(%quote(&_SGIO)) %then %do;
  data _null_;
    set _tbl end=eof;
    by &_SGTABLE;
    where upcase(&_SGIO) = 'YES'
    %if (%quote(&_SGFILTER) ne) %then %do;
      and &_SGTABLE in (%unquote(&_SGFILTER));
    %end;
    ;
    if _n_ = 1 then do;
```

```
        call execute("proc datasets lib=&_SGTLIB. nolist;");
      end;

      if first.&_SGTABLE then do;
        call execute("modify "||strip(&_SGTABLE)||";");
      end;

      call execute("index create "||strip(&_SGCOLNAME)||';');

      if last.&_SGTABLE then do;
        call execute("run;");
      end;

      if eof then do;
        call execute("quit;");
      end;
    run;
  %end;

  proc datasets nolist lib=work; delete _tbl; run; quit;

  %RCSET(&syserr);

%mend;
```