Paper 095-2009

Useful by Association: Adding Practical Value by Analyzing SAS® 9 Data Integration Metadata

Steve Morton, Applied System Knowledge Ltd, Henley-on-Thames, United Kingdom

ABSTRACT

SAS® 9 has an incredibly rich metadata model, but the documentation can be very daunting! This paper will explain the practical use of Associations and Attributes in the metadata to extract, analyze, and report this valuable resource using SAS® DATA step programming. Starting from simple techniques such as listing all libraries, examples will show how you can build a set of utility programs to report on how your ETL environment is structured

INTRODUCTION

When we create a SAS 9 metadata repository and start to use it in SAS® Data Integration Studio we invest significant time and effort in setting up metadata definitions for all the resources we will use – servers, SAS libraries, relational databases, tables, columns (the list goes on…).

Then we invest yet more time in building jobs with step-by-step logic using transformations, mapping columns and adding logic to create working jobs that process our data and load the data warehouse or data marts we are managing.

We then use that investment to generate code, creating jobs to run and schedule – but this only taps into part of the value we have created by defining all that metadata. We can get some more value by using the in-built analysis provided by Impact Analysis and Reverse Impact Analysis tools. These begin to show the value of the links that exist between the objects we have defined.

But, like an oil well that just pumped the easy stuff to begin with, there's still a lot of valuable 'crude' underground just waiting for us to bring along some new extraction methods to get at it. In the next few minutes I'll show some examples that I've found useful in recent work – but first, I'll show how to learn your way around the metadata jungle!

In this paper you'll see step-by-step how to build analyses of your ETL metadata using SAS DATA step programming and metadata API function calls. The finished analysis shown was used by the author on a recent project as an aid to release management and promotion of jobs.

MAIN FEATURES OF THE SAS 9 METADATA MODEL

The full description of the SAS 9 metadata model is given in the publication *SAS® 9.1.3 Open Metadata Interface: Reference* – but you'll find that this runs to four chapters and over 75 pages of text, tables and diagrams so it's not a quick coffee-break read! You'll also find that this manual gives most of its examples using Java and XML. Other supported languages are also covered by giving language-specific examples in C++ and Visual Basic – and there is a section that describes SAS DATA step functions to interact with metadata, but strangely these are not given a full set of examples like the other languages.

Now, I don't know about you all but I've been a SAS programmer for many years – and although I've learned some Java and Visual Basic along the way and can write either language when I have to, I still feel a lot more comfortable writing DATA step code. Anyway, on a SAS project I can always find a SAS programmer but I can't be sure of finding a Java or C++ programmer! So for the kinds of analysis I'm going to show here it's safer to stick with what we all know.

Incidentally, I would not advocate using DATA step functions to create or manipulate metadata objects – although the features exist to do so. The object oriented nature of the metadata makes constructing new objects and relationships a task most suited to object oriented languages; I recommend you program in Java if you need to do this.

So what are the main features of the SAS Open Metadata Architecture that you need to know if you just want to write DATA step programs to read metadata? Essentially just the primary components and how they are used:

- Objects
- Attributes

- Associations

Of these, probably the most valuable are the Associations. These are the cornerstone of data integration metadata.

Obvious examples of Objects are Servers, Tables, Columns, Jobs and Transformations; there are many more than these in the whole list!

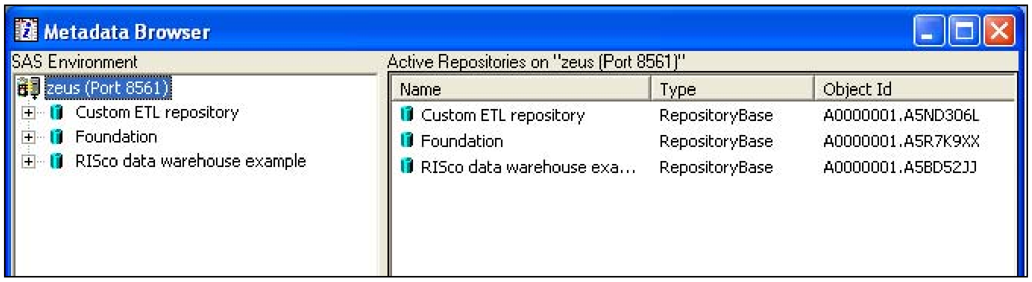
Each Object then has Attributes. For example, a Column will have Attributes that include its Type, Length, Name, Format & Informat. Attributes allow us to report on how our metadata Objects have been defined.

Most importantly, Associations then link Objects to other Objects. For example, each Column will have an Association to the Table it is in – and conversely the Table will have Associations to each of its Columns as well as to the Library in which it resides. Then whenever you drop a table onto a drop-zone for a transformation in an ETL job you create some more Associations. Associations provide most of the functionality exploited by SAS and they allow us to navigate around our metadata.

The other crucial feature to understand about the OMA is the naming of Objects, and how name searches can be performed through the API. Each object has a fully unique name or URI (Unique Resource Identifer) consisting of its type and metadata ID – for example a data library might have a URI of OMSOBJ:SASLibrary\A5R7K9XX.BH0000RT. This can also be retrieved by searching for it by type and name using a search value such as omsobj:SASLibrary?@Name='sastest'. I'll explain more of this by example when we look at some programs.

So it seems pretty simple, right? Well, fundamentally it is – but when there are so many different Object types and Associations, once they are put together in a given metadata repository there is much scope for the reader getting lost among all the details. Simply trying to take it all in from the documentation is a task that is likely to defeat all but the most determined of us!

For that reason I encourage you to move quickly from a brief reading of the overviews in *SAS Open Metadata Interface: Reference* to using a really handy tool that SAS provides in the desktop SAS environment – the Metadata Browser (see Figure. 1) accessed from the "Solutions=>Accessories=>Metadata Browser" pulldown menu.



Name	Type	Object Id
Custom ETL repository	RepositoryBase	A0000001.A5ND306L
Foundation	RepositoryBase	A0000001.A5R7K9XX
RISco data warehouse exa...	RepositoryBase	A0000001.A5BD52JJ

Figure 1 - Metadata Browser window

By using this expandable tree view to explore the metadata of a repository that you are already familiar with through SAS Data Integration Studio and SAS Management Console interfaces, you can rapidly understand how the whole mass of information is related together and start to see what you can do by reading this in a SAS program. Cross-check what you see in the browser with what the documentation tells you and it should start to quickly make sense. Note: if you don't have any Base SAS licenses for a desktop environment (either Windows or Unix) then you won't have this utility – there is no equivalent in SAS® Enterprise Guide®, for example.

FIRST STEPS WITH THE DATA STEP INTERFACE

The DATA step interface is implemented as a set of functions, all with names beginning METADATA_, and the most useful ones to us are the METADATA_GET… functions. There are also functions to perform 'SET' and 'NEW' actions (changing and creating objects respectively) and even to 'DEL' (delete) but since we are limiting ourselves to read-only use of the metadata I will not go into these at all.

The first thing we need to be able to do is access our metadata repository. This requires the SAS options metaserver, metaport, metaprotocol, metarepository, metauser and metapass with appropriate values. For example:

```
options       metaserver="MyServer"
              metaport=8561
              metaprotocol=bridge
              metauser="sasdemo"
              metapass="xxxxxxxxxx"
              metarepository="Foundation";
```

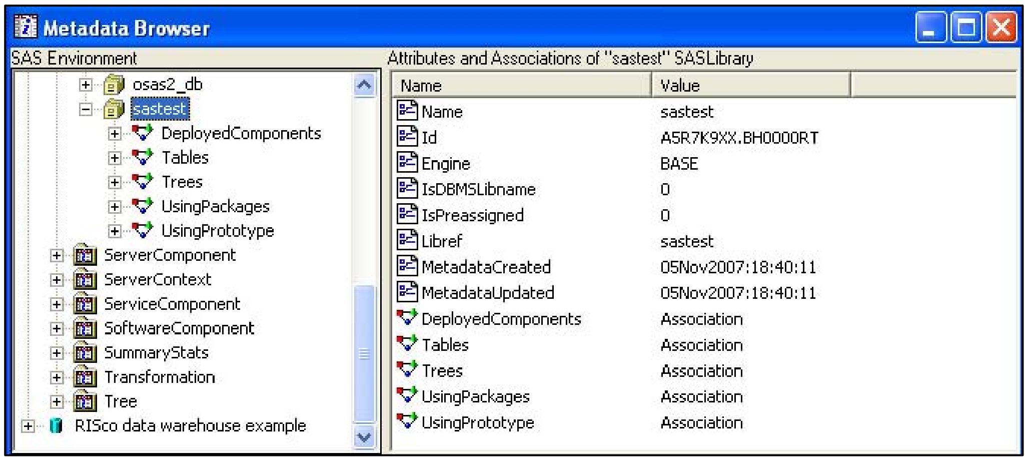
After this we can use the SAS program features that connect to the metadata server, including PROC METADATA and the DATA step API interface. So a simple program to retrieve one SAS Library by name could be as follows:

```
data _null_;
        a = 0; rc = 0; length attr value $256;
        Put 'Attributes of sastest library:';
        do while(rc >= 0);
                A + 1;
                rc = metadata_getnatr("omsobj:SASLibrary?@Name = 'sastest'", A,
                attr, value);
                if value ne '' then do;
                        put +5 attr ' = ' value;
                end;
        end;
stop;
run;
```

Notice that this uses just one metadata function: METADATA_GETNATR. The text string is a URI (Universal Resource Identifier) that specifies searching for a SASLibrary object with the name “sastest”. This call returns attribute values by number (in this example A is the attribute number). The output looks like this:

```
Attributes of sastest library:
     Name  = sastest
     MetadataUpdated  = 05Nov2007:18:40:11
     MetadataCreated  = 05Nov2007:18:40:11
     Libref  = sastest
     IsPreassigned  = 0
     IsDBMSLibname  = 0
     Engine  = BASE
     Id  = A5R7K9XX.BH0000RT
```

The same information displayed through Metadata Browser looks like Figure 2. Notice that Associations also exist but are not returned by this program:



Name	Value
Name	sastest
Id	A5R7K9XX.BH0000RT
Engine	BASE
IsDBMSLibname	0
IsPreassigned	0
Libref	sastest
MetadataCreated	05Nov2007:18:40:11
MetadataUpdated	05Nov2007:18:40:11
DeployedComponents	Association
Tables	Association
Trees	Association
UsingPackages	Association
UsingPrototype	Association

Figure 2 - metadata for SASLibrary 'sastest'

We can access specific attributes by their name using the function METADATA_GETATTR – a call as follows would simply retrieve the engine-name for the library:

```
rc = metadata_getattr("omsobj:SASLibrary?@Name = 'sastest'", ‘Engine’, value);
```

The programming can be made a little neater by always getting the explicit URI for the object we wish to process, using the METADATA_GETNOBJ function, which returns the exact URI value with the metadata ID. This is especially useful when we don’t know in advance the full object name – for example when searching for all objects of a given type (which we will see later). The call for this function looks like:

```
rc = metadata_getnobj("omsobj:SASLibrary?@Name = 'sastest'", 1, lib_uri);
```

It is interesting to note that the rc value returned for a successful call gives the number of objects that match the URI – which should be 1 if we have an explicit search as above, but would be >1 for a more general search. If you want to be sure that your ‘unique’ search really does only match one object it is good practice to test the return code value.

As well as Attributes we are of course interested in Associations; these are in fact the most powerful part of the metadata model. To access the Associations from an object we need two more calls: the METADATA_GETNASL to get each named association that could exist for the object, and the METADATA_GETNASN function to return the nth association of a given name. For example, to get all Associations of our test library, we would code:

```
data _null_;
       length id $20 type attr prop assoc value lib_uri auri $256;
    nobj = metadata_getnobj("omsobj:SASLibrary?@Name = 'sastest'", 1, lib_uri);
       if nobj gt 0 then do;
           put lib_uri=;
              n = 0; rc = 0;
              Put 'Associations of sastest library:';
              do while(rc >= 0);
                     n + 1;
                     rc = metadata_getnasl(lib_uri, n, assoc);
                     if assoc ne '' then do;
                            i = 0; arc = 0;
                            do while(arc >= 0);
                                   i + 1;
                                   arc = metadata_getnasn(lib_uri, assoc, i,
                                          auri);
                                   if arc >= 0 then do;
                                          put +5 assoc i ': ' auri=;
                                   end; *if arc…;
                            end; *do while…;
                     end; * if assoc…;
              end; *do while…;
       end; *if nobj…;
stop;
run;
```

This give output similar to:

```
lib_uri=OMSOBJ:SASLibrary\A5R7K9XX.BH0000RT
Associations of sastest library:
     DeployedComponents 1 : auri=OMSOBJ:ServerContext\A5R7K9XX.AT000001
     Tables 1 : auri=OMSOBJ:PhysicalTable\A5R7K9XX.BJ000006
     Trees 1 : auri=OMSOBJ:Tree\A5R7K9XX.AK0002BE
     UsingPackages 1 : auri=OMSOBJ:Directory\A5R7K9XX.B00000RT
     UsingPrototype 1 : auri=OMSOBJ:Prototype\A5R7K9XX.AA00003R
```

That may look incomprehensible at first glance, but it tells us that the library is in a single ServerContext (actually SASMain, but we would need another call to return the name itself), it has one Table and it is associated with a Package which defines a Directory (so it's probably a SAS data library). Like other objects of its type it has a place in a Tree (a folder) and is based on a Prototype that defines how such an object is made up.

By making further METADATA_GETATTR calls using the associated objects' URIs we can access more of the information about this Library, such as the physical path of the Directory it references or the name of the Table.

REPORTING ALL OBJECTS OF A TYPE

The initial examples have introduced the basic concepts, but some of the most useful metadata reports start not by knowing the name of the object we are searching for, but by getting each object of a specific type one-by-one.

In these cases we use a more general search URI – something like the following, which will match every SASLibrary object defined:

```
rc = metadata_getnobj("omsobj:SASLibrary?@Name ? ''", n, uri_lib);
```

So to get a list of all the directory paths of all the SAS Libraries, we could code:

```
data _null_;
       length name uri_lib uri_path path $256;
       do until (lrc lt 0); * iterate until no matching objects;
              n+1;   * get the next SASLibrary definition with non-blank name;
              lrc = metadata_getnobj("omsobj:SASLibrary?@Name ? ''", n,
                     uri_lib);
              if lrc > 0 then do;
```

```
                              path = ''; * clear any existing value for the path;
                              * now get attribute and association details;
                              rc = metadata_getattr(uri_lib, 'Name', name);
                              rc = metadata_getnasn(uri_lib, 'UsingPackages', 1,
                                    uri_path);
                              rc = metadata_getattr(uri_path, 'DirectoryName', path);
                              put / 'Library name= ' name / +8 path= ;
                        end; * if lrc...;
                  end; * do until...;
                  stop;
            run;
```

This produces log output like:

```
      Library name= groc
              path=Data\DWAD\Grocery

      Library name= sastest
              path=Data\sastest

      Library name= osas2_db
              path=
```

A couple of important points to notice here. One is that the third library (named osas2_db) does not have a Path – this is because it is a DBMS library definition and is associated not with a Directory but with a Database Schema. As a result it is also necessary – as well as good programming practice – to assign path=' ' before the metadata function call; if that is omitted then the value from the previous library will be wrongly put to the log for the osas2_db library, because the call made to METADATA_GETATTR looking for DirectoryName would fail – and a fail does not return any values other than the return code!

To have the program also handle DBMS schema references, replace the simple call and put with a conditional alternative so that when no Path is found we attempt to get the Schema details:

```
      rc = metadata_getattr(uri_path, 'DirectoryName', path);
      if rc=0 then put / 'Library name= ' name / +8 path= ;
      else do;
            rc = metadata_getattr(uri_path, 'SchemaName', schema);
            put / 'Library name= ' name / +8 schema= ;
      end;
```

For Library osas2_db this would then correctly output:

```
      Library name= osas2_db
              schema=sas2_db
```

MORE INTERESTING REPORTS

By now you should see the basic principles of metadata analysis and reporting – get the series of URIs for the Objects we want to analyze, follow the Associations to trace relationships we are interested in and read the Attributes stored for the related Objects.

Let's examine one final report to see how we can use this to get a list of all the Jobs in a repository together with the Input Tables used by each Job and the Output Tables loaded/updated.

First, examine the metadata for a known job. Figure 3 shows the example Job in the SAS Data Integration Studio Process Designer window, while Figure 4 shows the expanded metadata tree for the Job. Notice that there is one Step defined ("SCD Type 2 Loader") – but that we have to follow a JobActivities association through a New Transformation Activity object and then a Transformation association before we reach the Step. Then within the Step are the ClassifierSources and ClassifierTargets associations that will finally lead us to the input and output Tables! This shows why I recommend browsing the metadata while writing programs to access it.

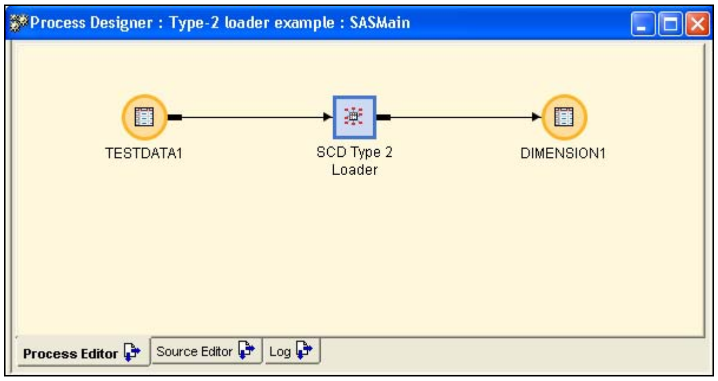
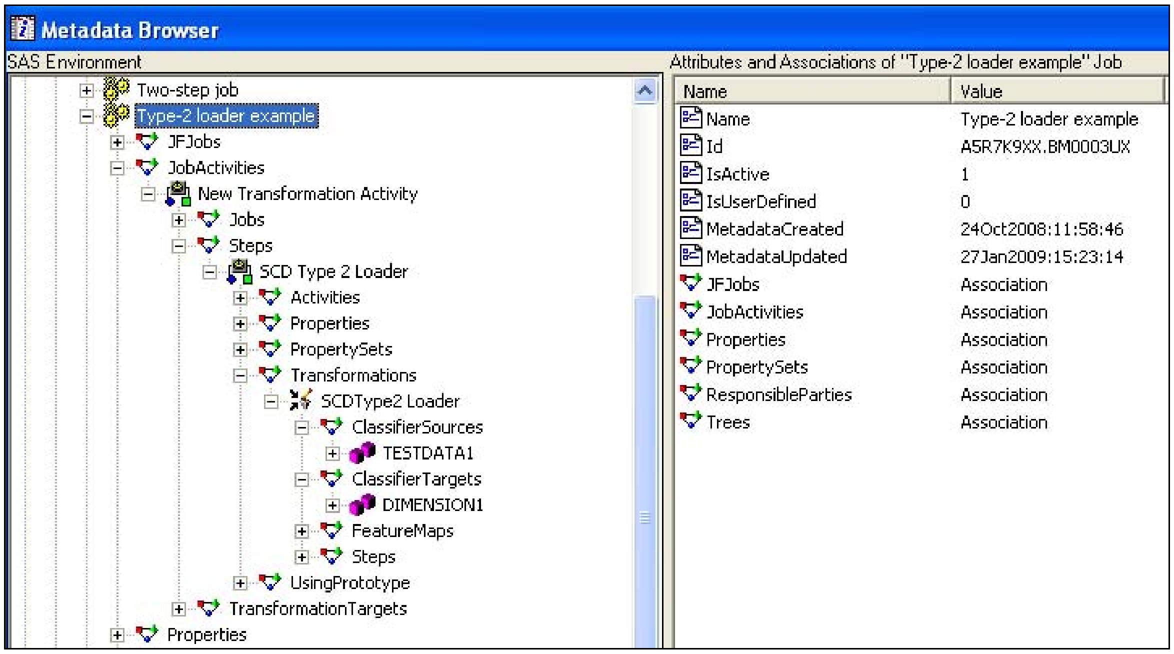


Figure 3 - simple Job in Process Designer



Metadata Browser

SAS Environment

```
Two-step job
Type-2 loader example
    JFJobs
    JobActivities
        New Transformation Activity
            Jobs
            Steps
                SCD Type 2 Loader
                    Activities
                    Properties
                    PropertySets
                    Transformations
                        SCDType2 Loader
                            ClassifierSources
                                TESTDATA1
                            ClassifierTargets
                                DIMENSION1
                            FeatureMaps
                            Steps
                    UsingPrototype
            TransformationTargets
    Properties
```

Attributes and Associations of "Type-2 loader example" Job

Name	Value
Name	Type-2 loader example
Id	A5R7K9XX.BM0003UX
IsActive	1
IsUserDefined	0
MetadataCreated	24Oct2008:11:58:46
MetadataUpdated	27Jan2009:15:23:14
JFJobs	Association
JobActivities	Association
Properties	Association
PropertySets	Association
ResponsibleParties	Association
Trees	Association

Figure 4 - Metadata for the simple Job

The code to return all metadata structured like this will need the following sections:

A. Get the URI for each Job in the repository, and execute a do-loop for the Job URI
B. For each Job URI, get the Name attribute and follow the associations to find out how many Steps exist
C. For each Step, get the Name attribute and follow Transformations association to get the Transformation URI
D. Use the Transformation URI to access ClassifierSources and ClassifierTargets associations, which will link to the Tables used as inputs and outputs respectively
E. For each object found as an input (Source) or output (Target), read the Table attributes and associations to determine the details to be reported, such as Library and Physical Table Name
F. Output the values required for reporting

I'll show the complete code at the end of the paper – there are still a couple of wrinkles to consider!

Figure 5 shows the result for our simple job of running the first version of this program. For each job we report the step_name of each step, the usage (Source or Target) for each table with its library and table name.

```
job_name=Type-2 loader example

                                      library_    physical_
    step_name            usage          name      table_name

SCD Type 2 Loader       Source        sas_db      TESTDATA1
SCD Type 2 Loader       Target        sas_db      DIMENSION1
```

Figure 5 - report output from simple job example

However, very few jobs are as simple as this so a couple of improvements will be needed to the finished analysis. Examining a two-step job we can see what a bit more complexity reveals (see figures 6 and 7).

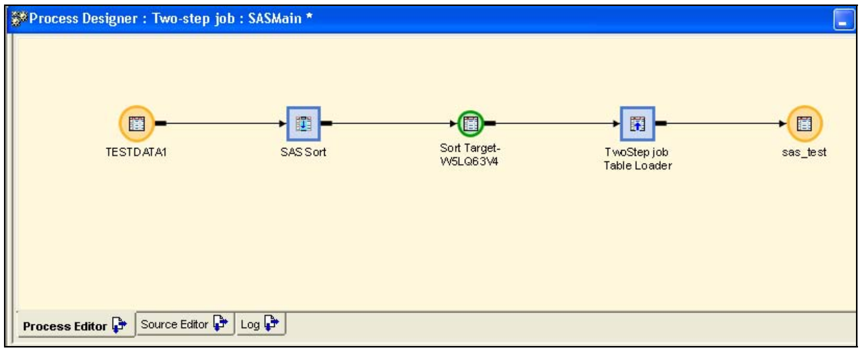
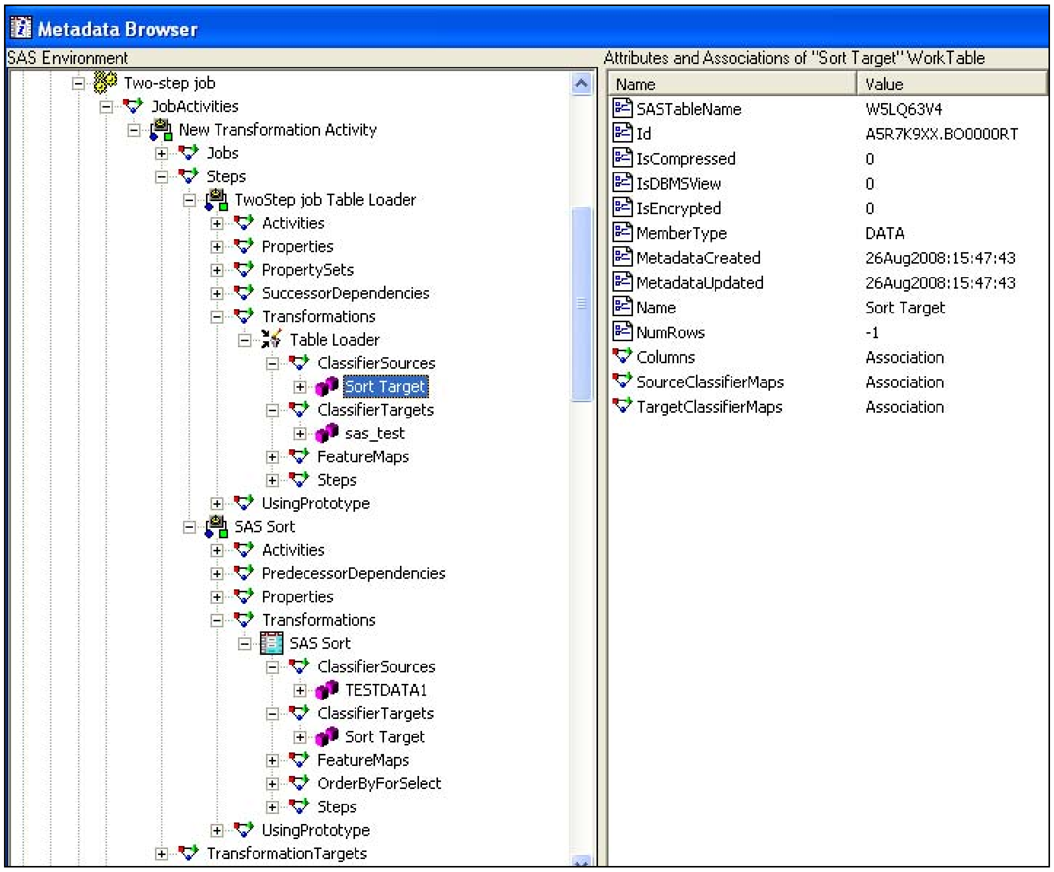


Figure 6 - Job with two steps



Metadata Browser

SAS Environment

```
Two-step job
  JobActivities
    New Transformation Activity
      Jobs
      Steps
        TwoStep job Table Loader
          Activities
          Properties
          PropertySets
          SuccessorDependencies
          Transformations
            Table Loader
              ClassifierSources
                Sort Target
              ClassifierTargets
                sas_test
              FeatureMaps
              Steps
          UsingPrototype
        SAS Sort
          Activities
          PredecessorDependencies
          Properties
          Transformations
            SAS Sort
              ClassifierSources
                TESTDATA1
              ClassifierTargets
                Sort Target
              FeatureMaps
              OrderByForSelect
              Steps
          UsingPrototype
      TransformationTargets
```

Attributes and Associations of "Sort Target" WorkTable

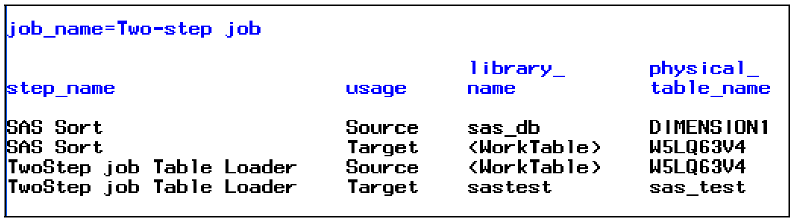
Name	Value
SASTableName	W5LQ63V4
Id	A5R7K9XX.BO0000RT
IsCompressed	0
IsDBMSView	0
IsEncrypted	0
MemberType	DATA
MetadataCreated	26Aug2008:15:47:43
MetadataUpdated	26Aug2008:15:47:43
Name	Sort Target
NumRows	-1
Columns	Association
SourceClassifierMaps	Association
TargetClassifierMaps	Association

Figure 7 - metadata for two-step job with intermediate work table

In this we see that each step has its own inputs and outputs, and between the Sort and the Loader is a WorkTable. When we examine the properties of the work table we find it has no library definition – when code is generated it will appear as work.W5LQ63V4 but the "work." is implied by its object type. To deal with this gracefully section E of the logic above must be extended with logic thus:

When the object type is WorkTable assign the library_name the value '<WorkTable>'

Finally, it must be remembered that not all steps have inputs and outputs – especially if a job is incomplete – so this situation also needs to be handled gracefully in section E logic. With these additions the code reports the two-step job as in figure 8.



```
job_name=Two-step job

                                              library_        physical_
step_name                  usage              name            table_name

SAS Sort                   Source             sas_db          DIMENSION1
SAS Sort                   Target             <WorkTable>     W5LQ63V4
TwoStep job Table Loader   Source             <WorkTable>     W5LQ63V4
TwoStep job Table Loader   Target             sastest         sas_test
```

Figure 8 - two-step job with work tables reported

TOWARDS A FINISHED RESULT

As the number of steps in a job becomes larger the presence of work tables becomes a distraction, so a further refinement is to ignore work tables altogether and exclude them from analysis. This final example uses a complex job from a demo data warehouse presented as part of the ‘Warehouse Architecture and Design Principles’ class which I present for SAS Institute in the UK. I’m using this because of the difficulty in getting permission to use a real customer’s ETL metadata – that is far more complex than the programming!

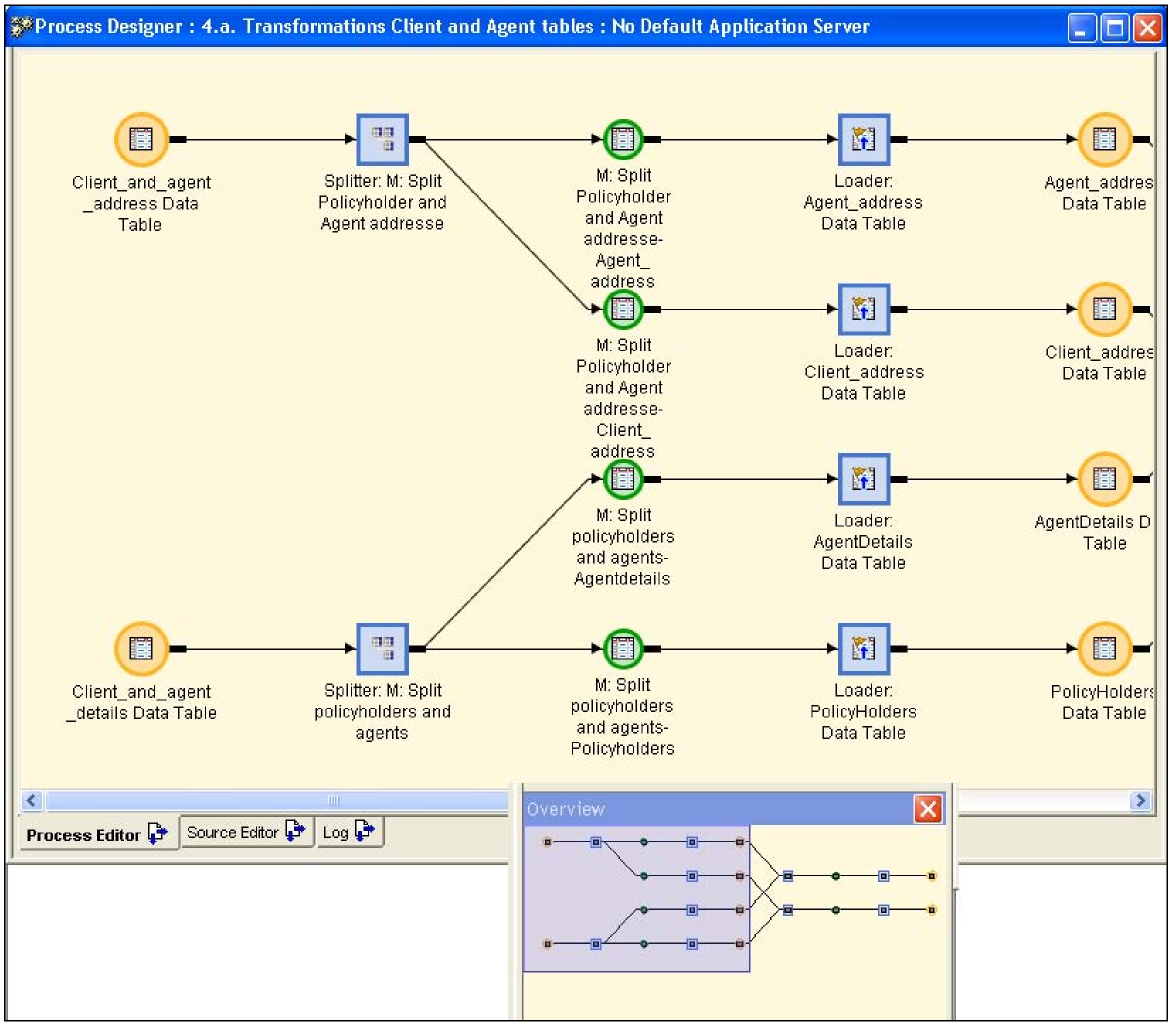
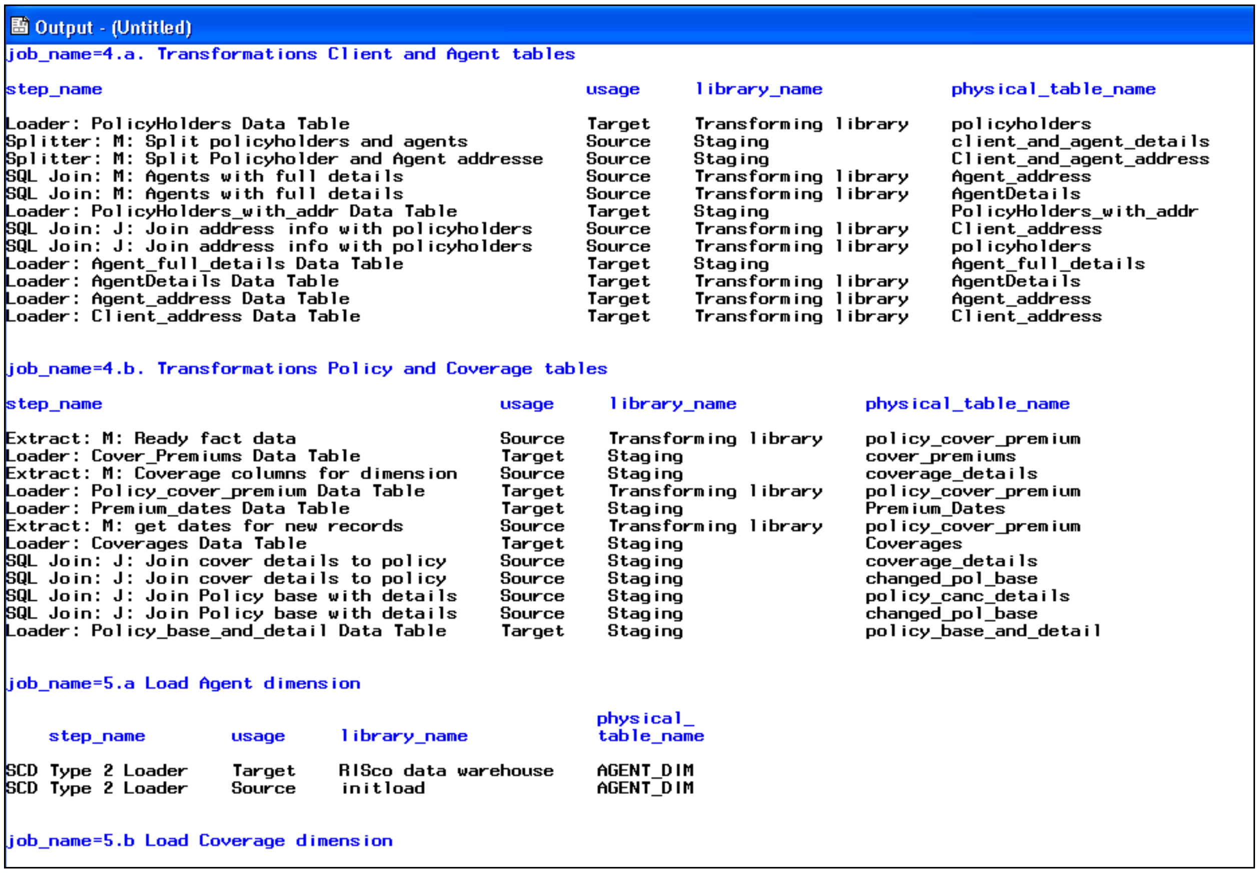

Process Designer : 4.a. Transformations Client and Agent tables : No Default Application Server
Client_and_agent _address Data Table
Splitter: M: Split Policyholder and Agent addresse
M: Split Policyholder and Agent addresse-Agent_address
Loader: Agent_address Data Table
Agent_addres Data Table
M: Split Policyholder and Agent addresse-Client_address
Loader: Client_address Data Table
Client_addres Data Table
M: Split policyholders and agents-Agentdetails
Loader: AgentDetails Data Table
AgentDetails D Table
Client_and_agent _details Data Table
Splitter: M: Split policyholders and agents
M: Split policyholders and agents-Policyholders
Loader: PolicyHolders Data Table
PolicyHolders Data Table
Overview
Process Editor
Source Editor
Log

Figure 9 - complex multi-step job with many inputs and outputs


Output - (Untitled)

```
job_name=4.a. Transformations Client and Agent tables

step_name                                          usage     library_name           physical_table_name

Loader: PolicyHolders Data Table                   Target    Transforming library   policyholders
Splitter: M: Split policyholders and agents        Source    Staging                client_and_agent_details
Splitter: M: Split Policyholder and Agent addresse Source    Staging                Client_and_agent_address
SQL Join: M: Agents with full details              Source    Transforming library   Agent_address
SQL Join: M: Agents with full details              Source    Transforming library   AgentDetails
Loader: PolicyHolders_with_addr Data Table         Target    Staging                PolicyHolders_with_addr
SQL Join: J: Join address info with policyholders  Source    Transforming library   Client_address
SQL Join: J: Join address info with policyholders  Source    Transforming library   policyholders
Loader: Agent_full_details Data Table              Target    Staging                Agent_full_details
Loader: AgentDetails Data Table                    Target    Transforming library   AgentDetails
Loader: Agent_address Data Table                   Target    Transforming library   Agent_address
Loader: Client_address Data Table                  Target    Transforming library   Client_address


job_name=4.b. Transformations Policy and Coverage tables

step_name                                  usage     library_name           physical_table_name

Extract: M: Ready fact data                Source    Transforming library   policy_cover_premium
Loader: Cover_Premiums Data Table          Target    Staging                cover_premiums
Extract: M: Coverage columns for dimension Source    Staging                coverage_details
Loader: Policy_cover_premium Data Table    Target    Transforming library   policy_cover_premium
Loader: Premium_dates Data Table           Target    Staging                Premium_Dates
Extract: M: get dates for new records      Source    Transforming library   policy_cover_premium
Loader: Coverages Data Table               Target    Staging                Coverages
SQL Join: J: Join cover details to policy  Source    Staging                coverage_details
SQL Join: J: Join cover details to policy  Source    Staging                changed_pol_base
SQL Join: J: Join Policy base with details Source    Staging                policy_canc_details
SQL Join: J: Join Policy base with details Source    Staging                changed_pol_base
Loader: Policy_base_and_detail Data Table  Target    Staging                policy_base_and_detail


job_name=5.a Load Agent dimension

                                                          physical_
    step_name         usage      library_name             table_name

SCD Type 2 Loader     Target     RISco data warehouse     AGENT_DIM
SCD Type 2 Loader     Source     initload                 AGENT_DIM


job_name=5.b Load Coverage dimension
```

Figure 10 - output from repository report

This is now the finished report. You may notice that steps don't always appear in the left-to-right order of the process editor diagram – this could be resolved by using predecessor/successor Associations and re-ordering the items using this information, but that's going too far for one paper and wasn't needed to achieve the purpose of this utility.

THE CODE

Note, the section sub-headings are not part of the code but refer to the logic steps mentioned above. The finished version of this has actually been turned into a macro for easier re-use. If you'd like to try it but prefer not to type it in, the example can be downloaded from my website www.appliedsystem.co.uk

Section A:

```
* Steps to read source & target for each step in every job;
data job_info;
    length     uri $256 next_uri $256 job_uri $256 step_uri $256 trans_uri $256
                      attr $256 value $256 job_name $256 step_name $256 trans_name $256
                      id $17 table_name $40 object_type $15 usage $10 library_name $40
                      physical_table_name $40;
    nobj = 1;
    n = 0;
    keep job_name step_name trans_name table_name physical_table_name library_name usage
       object_type step_order;

    /* Determine how many jobs are on this repository and identify inputs & outputs. */
    do while(nobj >= 0);
            n = n+1;
        nobj = metadata_getnobj("omsobj:Job?@Name contains ''", n, job_uri);
        if n = 1 then do;
              call symput('nobj', trim(left(nobj)));    /* Number of Job objects found. */
              if nobj lt 1 then do;
                      put 'Note: No Job objects found in this repository.';
                      stop;
              end;
        end;
```

Section B:

```
                if nobj gt 0 then do;
                        /* get the job name */
                        rc = metadata_getattr(job_uri, 'Name', job_name);
                        /* get the JobActivities association */
                        rc = metadata_getnasn(job_uri, 'JobActivities', 1, step_uri);
                        /*find out how many steps are associated */
                        n_steps = metadata_getnasn(step_uri, 'Steps', 1, next_uri);
```

Section C:

```
                        do s = 1 to n_steps;
                                /* get details of each step*/
                                rc = metadata_getnasn(step_uri, 'Steps', s, next_uri);
                                rc = metadata_getattr(next_uri, 'Name', step_name);
                                /* for each Step, get the Transformations assoc. */
                                uri = next_uri;
                                rc = metadata_getnasn(uri, 'Transformations', 1, next_uri);
                                rc = metadata_getattr(next_uri, 'Name', trans_name);
```

Section D:

```
                        /* get step inputs & outputs */
                                trans_uri = next_uri;
                                do type = 'ClassifierSources', 'ClassifierTargets';
                                        * iterate over the source/target associations;
                                        item = 0; number_of_tables = .;
                                        do until (item = number_of_tables);
                                                item = item + 1;
                                                number_of_tables = metadata_getnasn(trans_uri, type,
                                                item, next_uri);
                                                if number_of_tables gt 0 then do;
                                                        rc = metadata_getattr(next_uri, 'Name',
                                                        table_name);
                                                        rc = metadata_getattr(next_uri,
                                                        'SASTableName', physical_table_name);
                                                        object_type = scan(next_uri, 2, ':/\');
                                                        usage = substr(type, 11, 6);
                                * does the object have a TablePackage association? if so that will
                                identify the Library for persistent tables. Mote: this is not
                                present for WorkTable objects (transient intermediates)
                                so these will be assigned accordingly.;
                                                        uri = next_uri;
                                                        rc = metadata_getnasn(uri, 'TablePackage', 1,
                                                        next_uri);
                                                        if rc gt 0 then do;
                                                                rc = metadata_getattr(next_uri,
                                                                'Name', library_name);
                                                        end;
                                                        else do;
                                                                if object_type='WorkTable' then do;
                                                                        library_name = '<WorkTable>';
                                                                end;
                                                                else do;
                                                                * for any others, just leave blank;
                                                                        library_name = '';
                                                                end;
                                                        end;
                                                end; * if number_of_tables gt 0;
                                                else do;
                                                * when no inputs/outputs exist, provide some fixed
                                                text and prevent further looping;
                                                        library_name = '';
                                                        table_name = '<<no table specified>>';
                                                        physical_table_name =
                                                                '<<no table specified>>';
                                                        object_type = '';
                                                        usage = '?';
                                                        item = number_of_tables;
                                                end; *else [number_of_tables le 0];
```

Section F:

```
                                                * generate sequence number in reverse order (API
                                                provides steps backwards);
                                                step_order+-1;
                                                * Output results;
                                                output;
                                        end; * do until...;
                                end;
                        end;
                end;
        end;
run;
```

This code is then followed by a PROC SORT and PROC PRINT to generate the output seen.

CONCLUSION

From these examples you should be able to see that analyzing your metadata is well within the abilities of any competent SAS programmer – the only real challenge being to understand the structure of that metadata. By exploring the metadata in your repository using the Metadata Browser its mysteries are revealed, and the rest is up to your imagination!

REFERENCES

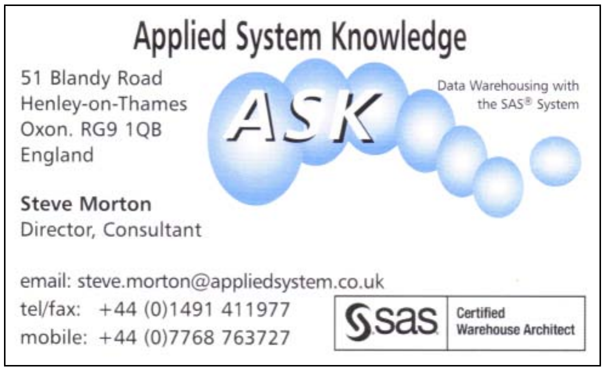
Metadata examples using ETL jobs from the SAS 'Warehouse Architecture and Design Principles' training course are used by permission of SAS Institute. Copyright © 2004-2007, SAS Institute Inc.

RECOMMENDED READING

SAS® Open Metadata Interface: Reference, Second Edition. Copyright © 2002-2007, SAS Institute Inc.

CONTACT INFORMATION

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



Applied System Knowledge

ASK

Data Warehousing with the SAS® System

51 Blandy Road
Henley-on-Thames
Oxon. RG9 1QB
England

Steve Morton
Director, Consultant

email: steve.morton@appliedsystem.co.uk
tel/fax: +44 (0)1491 411977
mobile: +44 (0)7768 763727

sas Certified Warehouse Architect