Paper CC-16

Which Job Sent *THAT* Error Message – How to Generate a Lookup List from Your Metadata

Robert Janka, Business Intelligence Professional, Apex, NC

ABSTRACT

Imagine you are a newly hired SAS® Platform Administrator and Data Integrator. It is trial by fire … jobs are failing! How would you quickly figure out which nightly jobs send which error messages? How would you match up those error texts to the appropriate jobs and create a lookup list that will help not only you, but your co-workers as well? The author found himself in just such a situation. He will present an example using DATA STEP metadata functions to automatically generate this lookup list.

INTRODUCTION

Using a combination of the METABROWSE command and Base SAS® programming, I will show how to explore your Business Intelligence metadata to generate a list of SAS® Data Integration jobs and the error messages they send. Readers should have intermediate skills in Base SAS programming, including some familiarity with SAS Macros. Readers interested in using the code described in this paper should have access to a SAS® Business Intelligence system which uses SAS® Data Integration Studio to build and run data flow jobs.

PROBLEM

As a recently hired SAS Application Engineer at a medium-sized technology company, I found myself in a position wearing many hats. The two relevant hats are Platform Administrator and Data Integrator. The previous incumbent had left the company several months earlier and no one remaining knew enough about SAS to handle any major issues while they were re-staffing this position.

During the first few weeks of work, some error messages appeared via e-mail. Here is an example:

```
Subject: ETL Process Error
22APR10:02:30:07 Errors loading Development Dimension table
```

This message comes from a nightly ETL (Extract, Transform, Load) job. Apparently, there were some errors severe enough to notify the administrator.

Now, what does a SAS Application Engineer do with this? She or he hunts around, finds the job logs, and looks for anything that might match “Development Dimension”. Eventually, she or he finds the issue and resolves it. In this particular case, I found that there was some contention for a file lock. A simple re-run of the job cleared the problem.

As an experienced IT professional, I took a step back. I realized that I needed a list of the regular ETL jobs at this work-site as a more general solution that would reduce the time to isolate future errors. SAS® Business Intelligence Architecture stores metadata with each job that can be used to connect jobs to their related error handling. By understanding and using this metadata, you can create a list (a look-up list, if you will) that benefits not only your administrator, but also any one else who might back-up your administrator when she or he is not available. What is the end result? Quicker resolutions, should a problem occur in your ETL processes.

OVERVIEW OF SOLUTIONS

This paper will examine the following three approaches to generating this lookup list:

1. Manual approach
2. String search approach
3. SAS programmatic approach

Although the first two options have a definite place as possible solutions, this paper will focus on the third approach. Utilizing the power and flexibility of the Base SAS programming language, we will use the DATA STEP metadata functions to walk the metadata objects and associations to extract the relevant items for the job and error message problem I described earlier. A key benefit I hope to impart is a sense of the power of the information SAS stores in its metadata. I encourage you to use this paper as a springboard to create other tools or reports to help you in your job. I sincerely hope that you enjoy the process as much as I did!

MANUAL APPROACH

At first glance, one might think it overkill to use SAS Data Integration Studio to manually open each job, one by one, and find out which ones generate error messages? However, there are least two benefits to this approach. First, you find the job names and the corresponding error messages for your lookup list. Second, and less obvious, you gain a hands-on awareness of what ETL jobs are active and what kinds of transformations each one uses. This approach is similar to reading the documentation for a new system: often tedious, but ultimately useful.

STRING SEARCH APPROACH

Anyone who has used “`grep`” on UNIX® knows the power of this approach. Use a simple string search tool to sift out the nuggets of information in the sea of log data.

The “grep” command, while powerful and available on many platforms, is a system-level approach which ties you to the specific operating systems that support the command and its options. I chose to use an application level programming approach instead, which is access on all SAS supported operating systems.

SAS PROGRAMMATIC APPROACH

Keep in mind, metadata is just a *different* kind of data. This data is something which we can manipulate with code. The SAS® Metadata Repository has application programming interfaces (APIs) which can access this metadata. Please refer to *SAS 9.2 Language Interfaces to Metadata* for a more complete listing of the available APIs.

Please note that the code referenced throughout the rest of this paper appears in full as an appendix.

The programmatic approach involves two paths, which we will alternate as we build up our `%jobs_actions_report()` SAS Macro. The first path is metadata exploration, using the METABROWSE command. The second path is DATA STEP programming, using the metadata functions.

METADATA TERMINOLOGY & EXPLORATION

Before we can write any code, let’s examine what we know. We have jobs that are failing … and those jobs are sending e-mail when failures occur. In this section, we will use the METABROWSE command to learn how to find this metadata (let’s call it “target metadata”) and how this target metadata is arranged. We will be looking for jobs and a piece of metadata that indicates e-mail is being sent. The connection of the job with the metadata related to e-mail is one example of an *association*.

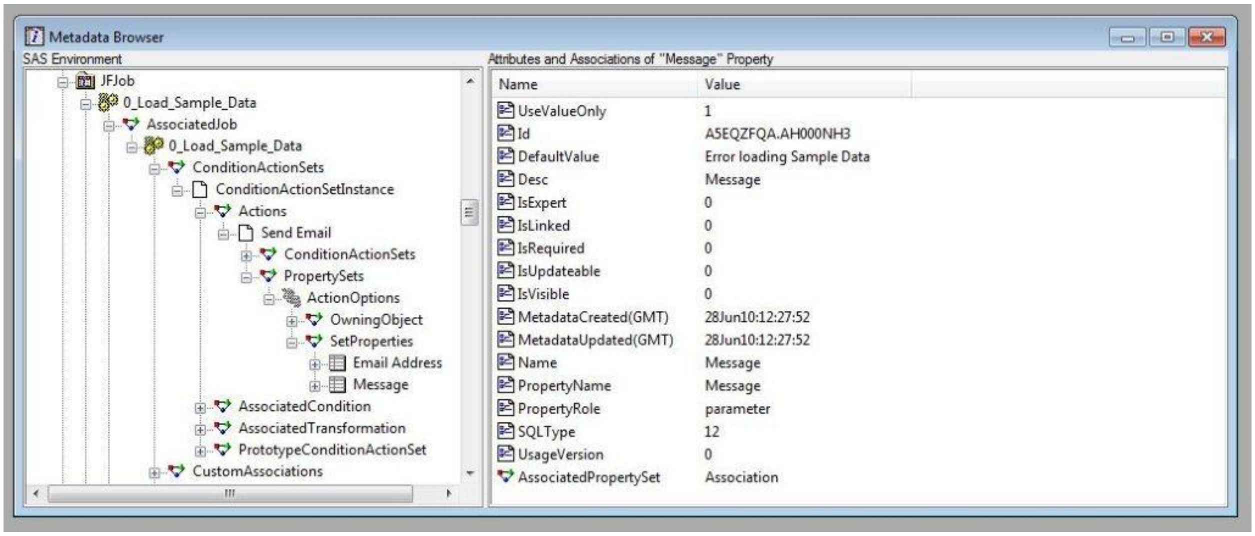
In addition to associations, each metadata object has a unique identifier known a Uniform Resource Identifier (URI). These URIs comprise two alpha-numeric strings separated by a period (‘.’). The first string uniquely identifies the metadata repository while the second uniquely identifies the individual metadata object. By printing the URI values with some of the metadata objects we encounter with our code, we can cross-reference them to the listing in the METABROWSE window to confirm that we have the desired objects.

In the SAS Business Intelligence metadata, there are a set of standard top-level object types. We will use one of these top-level object types, “JFJob”, to start our search for our target metadata.

Below is a preview of the different metadata objects we will encounter and their relationships. Items in *italics* are *metadata associations* while items in parentheses are job-specific names.

- JFJob
  - (Deployed Job Name)
    - *AssociatedJob*
      - (Original Job Name)
        - *ConditionActionSets*
          - ConditionActionSetInstance
            - *Actions*
              - Send Email
                - *PropertySets*
                  - ActionOptions
                    - *SetProperties*
                      - Email Address
                      - Message

Let’s look at an example of a job and some of its associated metadata:



Attributes and Associations of "Message" Property

Name	Value
UseValueOnly	1
Id	A5EQZFQA.AH000NH3
DefaultValue	Error loading Sample Data
Desc	Message
IsExpert	0
IsLinked	0
IsRequired	0
IsUpdateable	0
IsVisible	0
MetadataCreated(GMT)	28Jun10:12:27:52
MetadataUpdated(GMT)	28Jun10:12:27:52
Name	Message
PropertyName	Message
PropertyRole	parameter
SQLType	12
UsageVersion	0
AssociatedPropertySet	Association

Figure 1 Metadata Brower

Figure 1 shows one of the ETL jobs, “0_Load_Sample_Data”. We can walk from object to object, traversing the association links, until we reach our destination: the “Send Email” action with two key attributes: “Email Address” and “Message”. Please note that the association links look like little triangles, pointing downwards.

DATA STEP PROGRAMMING

Let’s start programming our SAS Macro by finding one which does something similar. In *SAS 9.2 Language Interfaces to Metadata*, Chapter 3, titled “Examples: Using Metadata Language Elements”, provides several examples of using language interfaces to manipulate metadata. We will use the fourth one: “Example: Creating a Report with the DATA STEP” as our starting point.

GET THE EXAMPLE WORKING

The first step is to copy the code from the PDF file, modify the metadata parameters to fit your site, and run the program.

TIP: Remember to check your single quotes when you cut and paste code examples from PDF files!

When I first copied the example, the code did not work cleanly. The problem was that the PDF file uses textual single quotes, which are slanted left and right, while many programming languages use vertical single quotes. A global *search and replace* command in the editor will resolve this issue if you encounter it.

There are two sections we need to modify: the metadata connections and possibly the host-specific filename and libname assignments. Here are the original sections from the sample code. Find these and modify them to fit your environment:

```
/* first section – metadata server connection */
/* Adapt to fit *YOUR* environment!! */

options metaserver="&metaserver"
metarepository="Foundation"
metaport=&metaport
metauser="&usr"
metapass="&pw";

/* second section – host-specific filename */
/* either change the default in the macro header */
/* OR specify it in the macro invocation */

htmlloc=c:\reports\myservers.htm
```

See Chapter 3 in *SAS 9.2 Language Interfaces to Metadata* for an example of the generated report. Take the time to make sure you have everything working correctly. Save a copy as a reference.

FIGURE OUT THE METADATA PATH

Here, we go back to the METABROWSE command to focus in on the desired metadata elements: “Email Address” and “Message” for each job. First, we have to decide which set of jobs we want to examine. For this paper, I decided to pick ALL deployed jobs. These are the “JFJob” objects which I mentioned earlier.

We now use the first of three key metadata functions: `metadata_getnobj()`. The other key functions are `metadata_getattr()` and `metadata_getnasn()`. See *SAS 9.2 Language Interfaces to Metadata* for more details on these functions.

Modifying the sample macro, `%server_report()`, we substitute the metadata object type in the search string for `metadata_getnobj()`. This function “Returns the nth object that matches the specified URI.”

The metadata functions which use URI’s have a very powerful option: Specifying a search string. Let’s examine this one used in our macro:

```
   nobj=metadata_getnobj( “omsobj:JFJob?Id contains ‘.’”, i, uri );
```

The first argument is the input URI. It has 4 parts: a leading “omsobj” to tell it to find a metadata object, a metadata object type, “JFJob”, a search indicator “?” and argument telling it to look at the “Id” attribute of our desired object type, and the string to match “ contains ‘.’”. This last part is subtle: every metadata object has an ID with two UUID strings, each 8 characters long, and separated by a “.” This is the same “.” which our search string is seeking! Remember the Universal Resource Identifiers (URIs) I mentioned earlier? The METABROWSE window shows them as the “ID” value. In figure 1, the ID for the Message property is this URI string: “A5EQZFQA.AH000NH3”.

What does this function return? The first metadata object of type “JFJob” (aka DeployedJob) which has a valid ID string. The second argument is set to 1 just before this call, telling it to get the 1st object. The third argument is the URI for the first object. The return value is the number of objects which match this specification. If the return value is 0 or less, then there was an error. Here is the first bump we might encounter.

Why does our program call this function twice? The first time gets the number of top-level objects which match our search condition. Then, we loop from that number of object (from 1 to nobj), and the second call gets each successive top-level object in the metadata as we index it with the loop iteration count.

For our target metadata objects, we need to walk a string of associated objects. For each one, we need to get and keep its URI. The next section of the macro resets these variables for each deployed job. We also clear some of our value variables as not all jobs have all of the intermediate values. We do that at the top of the loop to set the stage for each job we encounter. To help in debugging, the code prints outs a separator line and the name of each DeployedJob to the log. Then, we get our first set of variables: the Name and the ID of the job.

Next, we traverse our first association to get the AssociatedJob using the “metadata_getnasn()” function. This function takes a source URI, the association type, “AssociatedJob”, a count of 1 (we want the first and only one), and a target URI. It returns a count of associated objects. If that result is positive (1 or greater), then we know that the target URI is valid and we can proceed to retrieve the next variable, the jobname of the associated job.

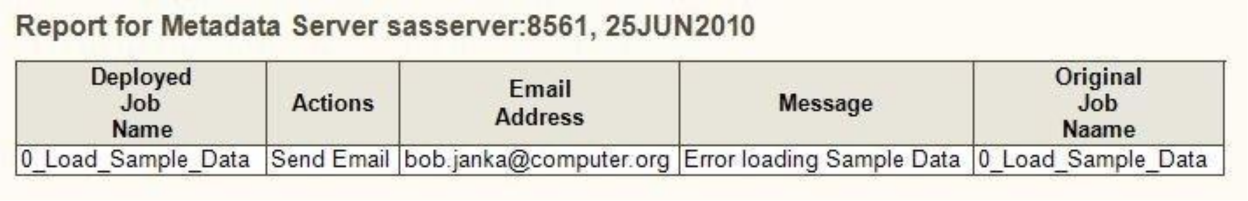
We repeat this process as work through the metadata associations tree.

When we get to the SetProperties, we have two metadata objects containing our desired fields: “EmailAddress” and “Message.” If we had a lot of these to extract, such as the server options in the original example code, we could extract these into a separate dataset, sort both datasets, transpose the options dataset, and merge the two datasets. In our case, we only have two values. We can loop through the properties, check for the desired properties via their “Desc” attribute and save the corresponding value via their “DefaultValue” attribute into an appropriately named dataset variable.

Save the persistent variables and repeat for each remaining DeployedJob. At the end, we have a dataset with one row per DeployedJob containing these variables:

Name
Id
JobName
Action
EmailAddress
Message

At this point, most readers will have their own preferences on how to display the dataset. I chose to modify the code provided in the original example for PROC REPORT and successfully generated a listing similar to that shown in figure 2:



Report for Metadata Server sasserver:8561, 25JUN2010

Deployed Job Name	Actions	Email Address	Message	Original Job Naame
0_Load_Sample_Data	Send Email	bob.janka@computer.org	Error loading Sample Data	0_Load_Sample_Data

Figure 2 Report

USING THE MACRO

The user which submits this macro will need a userid which has authorization to view the desired job metadata. The macro has 5 arguments. The first four provide the connection details to the metadata server. The fifth and final argument tells the macro where to put the HTML results from Proc REPORT.

FUTURE THOUGHTS

What do we do next? One obvious step is to extend the code to dive into the individual jobs and find any “Send Email” actions on individual transformations. Another step is to explore code changes to handle some of the other actions.

CONCLUSION

This paper describes how to explore metadata and use those findings as a guide to writing programs which report on that metadata. We learned a little about metadata objects and associations, explored the metadata using the METABROWSE command to find the path to our target metadata, and wrote a macro with an embedded DATA step to programmatically walk that metadata and report on the information.

I sincerely hope that the general methodology presented will help you write your own metadata report macro.

REFERENCES

SAS Institute Inc. 2009, *SAS® 9.2 Language Interfaces to Metadata*. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author thanks his client for the opportunity to tackle this problem with their SAS® Enterprise Business Intelligence environment. The author is also indebted to many of his former colleagues at SAS Institute, Inc. and gratefully recognizes their direct and indirect contributions to his knowledge and skills with SAS® Enterprise Business Intelligence.

CONTACT INFORMATION

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

Robert Janka
Business Intelligence Professional
Apex, NC
E-mail: Bob.Janka@computer.org

APPENDIX – CODE FOR %jobs_actions_report()

```
/* Name:      %jobs_actions_report() */
/* Author:    Bob Janka - Bob.Janka@computer.org */
/* Version:   v1.1 */

/* Adapted from %server_report()  */
/*   Chapter 3, Example 4  in  _SAS 9.2 Language Interfaces to Metadata_ */

/* MACRO DEFINITION */

%macro jobs_actions_report (
    metaserver=myserver,
    metaport=8561,
    usr=sasadm@saspw,
    pw=Password1,
    htmlloc=c:\reports\my_jobs_actions.htm
  );

  options
    metaserver="&metaserver"
    metarepository="Foundation"
    metaport=&metaport
    metauser="&usr"
    metapass="&pw";

  data _null_;
    length ver $20;
    ver=left(put(metadata_version(),8.));
    put ver=;
    call symput('METAVER',ver);
  run;

  %if %eval(&metaver>0) %then
    %do; /* connected to metadata server */
      data
        etl_jobs(keep=id name jobname condactionset action
          propset set_props property propvalue email_addr message)
        ;

        length uri job_uri set_uri action_uri prop_uri setprops_uri $500
        id $17 name jobname action propset set_props $50
        condactionset property propvalue email_addr message $200;

        nobj=1;
        n=1;

        /* Get list of jobs (JFJob ~= Deployed Job) */

        nobj=metadata_getnobj( "omsobj:JFJob?@Id contains '.'", n, uri );
        do i=1 to nobj;

          /* reset variables for each job */

          job_uri = "";         set_uri = "";            action_uri = "";
          prop_uri = "";        setprops_uri = "";

          jobname = "";         condactionset = "";      action = "";
          propset = "";         set_props = "";          property = "";
          propvalue = "";       email_addr = "";         message = "";

          /* Start examining this job ('i'th of 'nobj') */

          nobj=metadata_getnobj( "omsobj:JFJob?@Id contains '.'", i, uri );

          rc=metadata_getattr(uri,"Name",Name);
          rc=metadata_getattr(uri,"id",id);
```

```
put '-----------------------------------';        /* debugging */
put name=;                                         /* debugging */

/* For each deployed job, get its original job */

numjob=metadata_getnasn(uri,
  "AssociatedJob",
  1,
  job_uri);
if (numjob > 0) then
  do;
    rc=metadata_getattr(job_uri, "name", jobname);
  end;

/* OK, get the ConditionActionSets for the original job */

numset=metadata_getnasn(job_uri,
  "ConditionActionSets",
  1,
  set_uri);
if (numset > 0) then
  do;
    /* debugging */
    rc=metadata_getattr(set_uri, "name", condactionset);
  end;

/* Next, get the Actions for the job */

numactions=metadata_getnasn(set_uri,
  "Actions",
  1,
  action_uri);
if (numactions > 0) then
  do;
    rc=metadata_getattr(action_uri, "Name", action);
  end;

/* Then, walk down to the "PropertySets" association */

numprop=metadata_getnasn(action_uri,
  "PropertySets",
  1,
  prop_uri);

if (numprop > 0) then
  do;
    rc=metadata_getattr(prop_uri, "Name", propset);   /* debugging */
  end;

/* Now for the fun part of extracting the EmailAddress & Message   */
/*   These are found via the "Set_Properties" associations */

numsetprops=metadata_getnasn(prop_uri,
  "SetProperties",
  1,
  setprops_uri);

do nsp = 1 to numsetprops;
  numsetprops=metadata_getnasn(prop_uri,
    "SetProperties",
    nsp,
    setprops_uri);

  rc1=metadata_getattr(setprops_uri, "Desc", property);
  rc2=metadata_getattr(setprops_uri, "DefaultValue", propvalue);
  if (rc1>=0 AND rc2>=0) then
```

```
              do;
                if (property eq "EmailAddress") then
                  do;
                    email_addr = propvalue;
                    put email_addr=;                        /* debugging */
                  end;
                else if (property eq "Message") then
                  do;
                    message = propvalue;
                    put message=;                           /* debugging */
                  end;
              end;

          end;    /* do nsp 1..numsetprops */

          /* Save the key information about each job */

          output etl_jobs;

        end;    /* do i 1..nobj (aka all deployed jobs) */

      run;    /* DATA step to gather job metadata */

      /* Use PROC REPORT to pretty print the jobs and error messages */

      ods listing close;
      ods html body="&htmlloc";
        title "Report for Metadata Server &metaserver:&metaport, &sysdate9";
        footnote ;

      proc report data=etl_jobs
        nowindows headline headskip split='*' nocenter;

        column name action email_addr message jobname   ;
        define name / group flow missing "Deployed*Job*Name";
        define action / group flow missing "Actions";
        define email_addr / group flow missing "Email*Address";
        define message / group flow missing "Message";
        define jobname / group flow missing "Original*Job*Naame";

        break after name / style=[BACKGROUND=CCC];

      run;

      ods html close;
      ods listing;

    %end; /* connected to metadata server */

  %else
    %do;    /* could not connect to metadata server */
      %put ERROR: could not connect to &metaserver &metaport. ;
      %put ERROR: check connection details, userid and password.;
    %end;   /* could not connect to metadata server */

%mend; /* jobs_actions_report */

/* MACRO INVOCATION */

%jobs_actions_report (
  metaserver=myserver,
  metaport=8561,
  usr=Userid,
  pw=Password,
  htmlloc=c:\reports\myservers.htm
);
```