Paper 131-2008

Tying your metadata to your other information

Frank Poppe, PW Consulting, the Netherlands

ABSTRACT

Managing SAS® v9 applications means using information of different nature and origin. The applications in this context are both ETL processes and the creation of OLAP cubes and Information Maps to surface the information for the end user. Examples of types of information are the following.

1. Requirements for an application, technical architecture, technical draft, implementation issues, etc. This is mainly textual information, which can be in any form.
2. The metadata as it is being produced through the different clients: SAS® Management Console, the SAS® Data Integration Studio, SAS® OLAP Cube Studio, etc. This is all stored in the SAS Metadata architecture.
3. Status information and logging of the scheduled processes. Standard this information is partly available in the scheduler (like LSF) and optionally as log files.

In this presentation a set of guidelines will be described together with some SAS macro code developed to be used in the job status options of DI Studio. Together with the right settings this will tie the information together.

For example, this will make it possible to jump from the properties of a job in DI Studio to the documentation on the process, or to the log file of the last run of that job. And when browsing the metadata one will see among the attributes of a job also the return code and the date time stamp of the last run.

INTRODUCTION

The development of the techniques described in this paper is the result of a certain frustration.

After developing and deploying up jobs for the first time using DI Studio (DIS) and scheduling them using the SAS Management Console (SMC) I started looking in the SMC for the information on the actual run. I expected to find items like the date and time of the last start of the job, the result code and the log file. But no such information was available.

I first will describe how I see the different kinds of information that together form a project. Developers and managers of running SAS applications in their daily work use different sources of information. Some are used almost continuously, others are accessed only when something out of the ordinary is at hand, e.g. a daily running ETL process has crashed. I recognize the following three clusters of information.

TEXTUAL INFORMATION

Some information is in text documents somewhere on the intranet, as a Word document or maybe as an HTML document.

METADATA

Other information has to be found in the SAS metadata, accessible through the various SAS Java clients like Data Integration Studio (DIS) and the SAS Management Console (SMC). The metadata can also be browsed or even edited using the Metabrowse command in an ‘old-fashioned’ SAS session or the Metadata Utility within the SMC. And there also is the web based Metadata Explorer.

STATUS INFORMATION

A third kind of data is that about the jobs scheduled to run on the SAS server. Scheduling (using LSF or other tools) can be done on a regular interval (each day, week, etc.) or on the basis of particular events (like the existence of a certain source file in a directory). Items of interest are items like the return code of the last run, the log file of the last run and the date and time of the last run.

Some of these items can be found in the LSF interface, but it is not available in the SAS metadata.

HOW THIS PAPER IS ORGANIZED

Because the basic idea behind this paper is to add information to the metadata I start with a short discussion on the position of the metadata and the metadata server in a production environment.

In the next paragraphs I will for each of those clusters of information describe ways to store and access that information, including links from items in one cluster to items in the other cluster. In the paragraphs on metadata I will also give some attention to the way metadata is stored.

Finally I describe the combination of SAS code and settings in the SMC that can be used to make status information available in the metadata.

THE METADATA SERVER AND THE PRODUCTION LEVEL

Writing up to date information into the metadata obviously requires a running metadata server, accessible from the running jobs. That a metadata server must be up and running in the SAS version 9 architecture may seem a truism, but this appears not always to be so for production systems.

When a development cycle has been set up at a site, with a development environment and possibly test and acceptance environments separated from the production environment the first levels always have a metadata server running at all times. This indeed is a truism. (Whether the different levels have been separated only logically or also physically doesn't matter here.)

But although in all publications SAS positions the metadata as the central point of control for everything and anything, one can also observe that all transformations in DI Studio generate code that does not rely on any communication with the metadata server when the code is run. All information needed to access external sources (host names and addresses, schema names, usernames, passwords, etc.) is collected from the metadata when the code is generated and written in the SAS code. This means that a deployed job containing only standard transformations from DI Studio can run without the need for a running metadata server.

This strongly relates to the issue how a job gets deployed in the production environment. It may very well be that there is no possibility of running DI Studio against the metadata server in production to deploy a job there. Jobs often have to be offered ready to run, i.e. as .SAS files with the right options, machine names, user names, etc., already in place. This leaves still open the possibility that there is a metadata server running, although it is not being used to deploy the jobs. However, there must be mechanism in place to ensure that the metadata on the production level has been synchronised with the metadata used to deploy the jobs.

One might also argue that, from a security point of view, it is not a good practice to have security sensitive information written out in the SAS code. Passwords of course should be encoded (in the `{sas001}KioqKioqKio` form), but also encoded passwords can be used for a lot of things from within SAS.

A better system might be to extract the necessary information at run time from the metadata server.

Also it is my observation that when the need arises to add user written code, or to develop user generated transforms, it often is necessary to use code that communicates with the metadata server, in order to avoid hard coding access information that might change later. (This might be avoided when one develops user transformation in Java code that would be able to communicate with the metadata server at deployment time, as many standard transformations do.)

In any case, the methods described in this paper rely on having a running metadata server in the production environment.

TEXTUAL INFORMATION

To incorporate textual information into a system of links the obvious and essential requirement is that the text is stored in a format that allows hyperlinks. Many word processing packages like Microsoft Word nowadays know how to handle hyperlinks. So the ideas presented here can be brought into practice on the basis of Word documents stored on disks shared by all users.

But a more flexible system is the use of an intranet to make the information available to everybody, and the use of a Wiki system to store and edit the information.

It then is very easy to create links between the textual information, to search through the documentation for particular items, and to link into it from another application.

In the remainder I will suppose that a Wiki is being used for all textual information, but, with slight modifications, everything will be valid for other way of storing the information as well.

METADATA

Before launching into a description of the different clients that can be used to access the metadata I will shortly describe the architecture of the metadata. If you are already familiar with that you can skip the next paragraph.

METADATA OBJECTS

All information in the metadata is fixed in metadata objects, the smallest atoms of information in the metadata world. These objects are related to each other through associations. All associations work either way: if an object *x* has an

association with object *y*, then object *y* will also show an association with object *x*. The metadata model is fixed, i.e. the different types of metadata objects are defined, and for each type the attributes are defined. Also it has been defined between which object types associations are possible.

Things that are shown in a client as one item usually consist of many metadata objects. E.g., a SAS table does exist as a PhysicalTable object but is not complete as a table without many other associated objects.
In the first place there may be an association with a SASLibrary (the libref). The SASLibrary in itself has different associations to make it a complete libref. One would not consider these 'secondary' associated objects as part of the table information. It should be noted however that within the context of the metadata there is no such border: there is an endless string of associations.
Then there are multiple associations with Column objects. Column objects often have other associations, apart from the association with the Table object. E.g., if a column is used in DI Studio as the value of a transformation option, there will be an association from that transformation to the column object, and, because of the two-sided way associations work, this is also visible in the metadata tree as an association from the column object to that transformation. The transformation is of course associated to the job, etc., etc.

For associations it is also defined whether it is an optional or a required association and for each side of the association whether multiple associations can exist. This definition regulates what happens when an item is being deleted: if a metadata object *x* has a required association with a metadata object *y*, then metadata object is always deleted with metadata object *y*.
Note that although an association always works either way, the definition of optional versus required may differ. E.g., when a table is being deleted all column objects are deleted with it, because for a column object an association with a table object is required. But of course a column object may be deleted without the table it belongs to being deleted.

Somewhere in the background the metadata objects also have their physical representations. In the end they are stored in observations in SAS tables in the directory associated with the repository. They might also be stored in an external DBMS. But these tables are better left alone.

CLIENTS TO ACCESS METADATA

Since the metadata objects are linked to each other through associations the metadata is a natural candidate for a hyperlinked representation. Several clients let you jump from one metadata object to the next, ad infinitum.
Some clients simply show selected metadata objects with all attributes and with their associations to other metadata objects. On the other hand clients that are designed for specific tasks only show the metadata that is deemed relevant for those tasks, and present them in a way deemed suitable for that task.

METADATA BROWSER

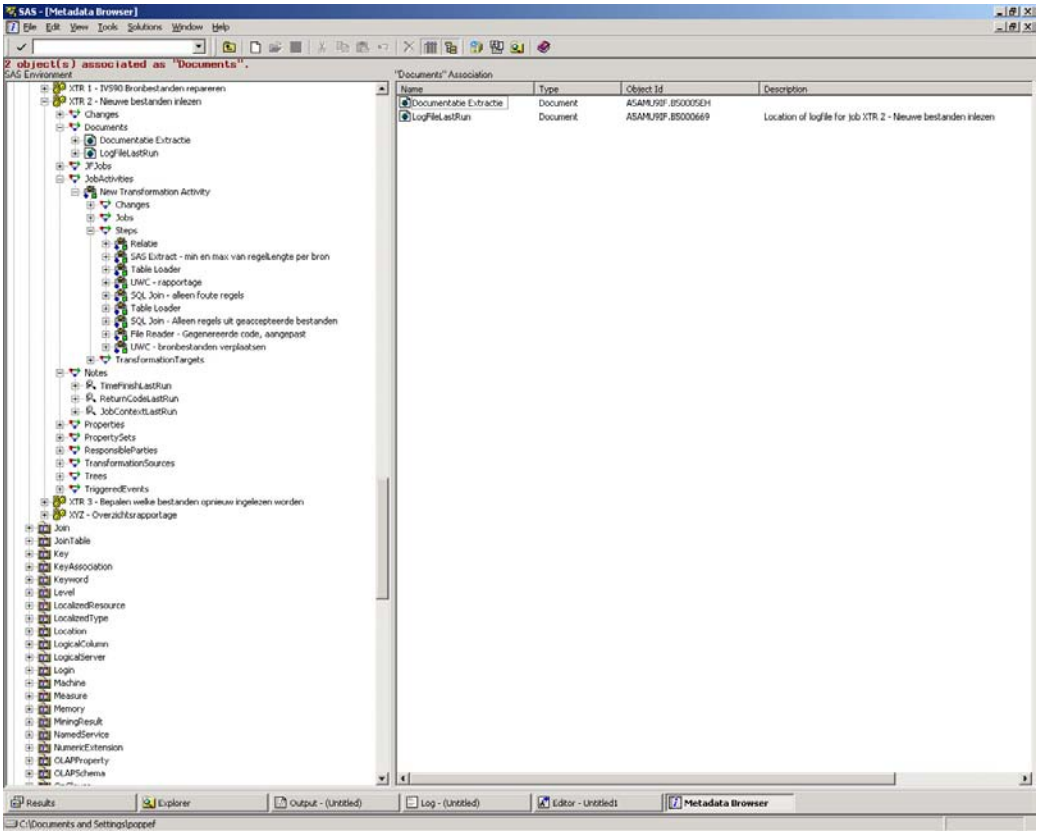
The Metadata Browser runs in an ordinary old-fashioned SAS session and can be launched with the METABROWSE command. In a dialog screen the information to connect to a metadata server can be entered (machine name, port, protocol, user name and password). It then shows all metadata objects, starting with the repository objects.
Each repository object can be folded out to show the different object types, and from there the objects are shown.
Then for each object the associations can be folded out and followed. This can be done ad infinitum.
By default the Metadata Browser shows only a subset of metadata types. The subset shown can be set through the Explorer Options dialog, which can be found in the following way:

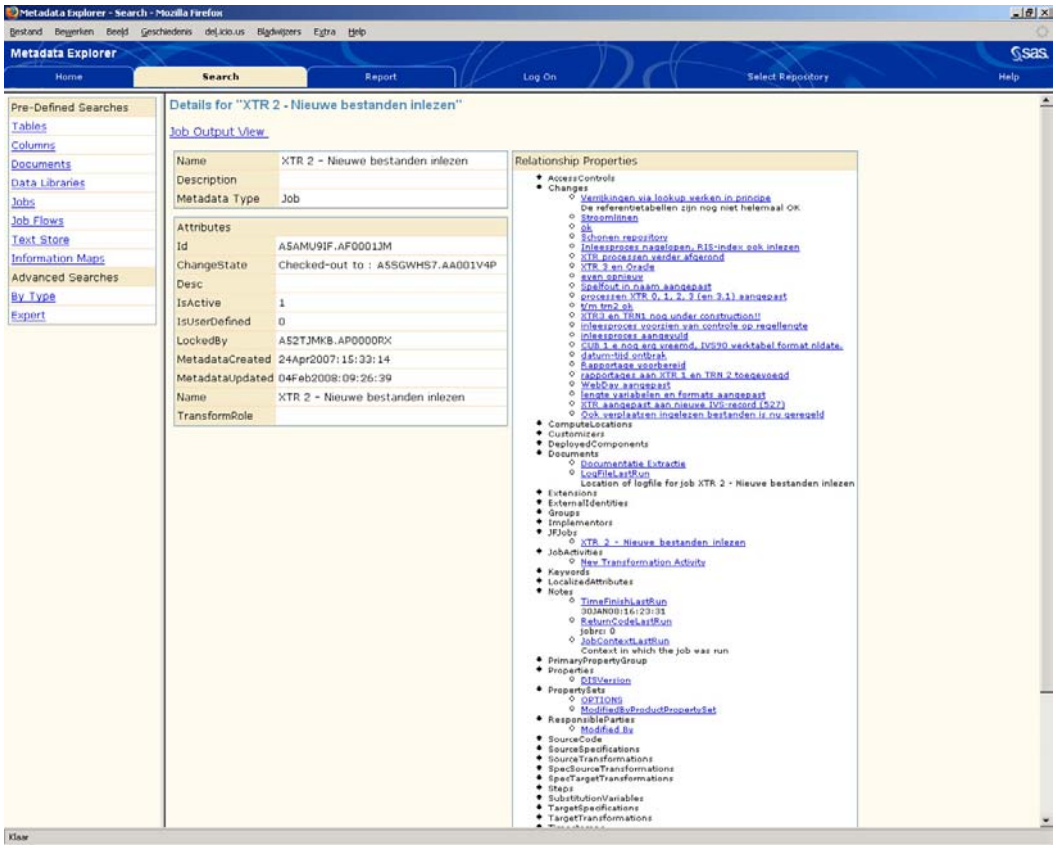
- select the menu *Tools*
- choose *Options >*
- select *Explorer…*
- click the tab Metadata.

There you can choose *Hide All* or *Unhide All*, or hide and unhide specific types selectively.
A screen shot of the results can be seen on top of the next page.

METADATA EXPLORER

The Metadata Explorer is a web application.
Under version 9.1 it is not in the standard installation but it can be obtained from SAS Support. To deploy it you obviously need a web server and the rights to administer it.
Also in the *Metadata Explorer* you connect to a metadata server, and then you select a repository.
From there you can look for different object types, and search for specific names or id's. Then each association is shown as a hyperlink that selects the next object. The screenshot (bottom of next page) shows the information on more or less the same point in the chain of associations as in the Metadata Browser above.


SAS - [Metadata Browser]
2 object(s) associated as "Documents".
"Documents" Association
Name
Type
Object Id
Description
Documentatie Extractie
Document
A5AMU9IF.B50005EH
LogFileLastRun
Document
A5AMU9IF.B5000669
Location of logfile for job XTR 2 - Nieuwe bestanden inlezen
Results
Explorer
Output - (Untitled)
Log - (Untitled)
Editor - Untitled1
Metadata Browser


Metadata Explorer - Search - Mozilla Firefox
Metadata Explorer
Home
Search
Report
Log On
Select Repository
Help
Pre-Defined Searches
Tables
Columns
Documents
Data Libraries
Jobs
Job Flows
Text Store
Information Maps
Advanced Searches
By Type
Expert
Details for "XTR 2 - Nieuwe bestanden inlezen"
Job Output View
Name
XTR 2 - Nieuwe bestanden inlezen
Description
Metadata Type
Job
Attributes
Id
A5AMU9IF.AF0001JM
ChangeState
Checked-out to : A5SGWHS7.AA001V4P
Desc
IsActive
1
IsUserDefined
0
LockedBy
A52TJMKB.AP0000RX
MetadataCreated
24Apr2007:15:33:14
MetadataUpdated
04Feb2008:09:26:39
Name
XTR 2 - Nieuwe bestanden inlezen
TransformRole
Relationship Properties
AccessControls
Changes
Documents
Documentatie Extractie
LogFileLastRun
Location of logfile for job XTR 2 - Nieuwe bestanden inlezen
Extensions
ExternalIdentities
Groups
Implementors
JFJobs
XTR 2 - Nieuwe bestanden inlezen
JobActivities
New Transformation Activity
Keywords
LocalizedAttributes
Notes
TimeFinishLastRun
ReturnCodeLastRun
JobContextLastRun
Context in which the job was run
PrimaryPropertyGroup
Properties
PropertySets
ResponsibleParties
SourceCode
SourceSpecifications
SourceTransformations
SpecSourceTransformations
SpecTargetTransformations
Steps
SubstitutionVariables
TargetSpecifications
TargetTransformations

DATA INTEGRATION STUDIO

DI Studio of course is a client that is not designed to show metadata in general, but to perform certain specific actions. It therefore shows only the metadata that is relevant to those actions. However, it is possible to associate the information we are interested in now in such a way that DI Studio shows.

If the information is stored in a certain way in specific metadata objects it will be visible in the job *Properties* dialog, in the *Notes* tab.

An example can be seen in the screen shot on top of the next page.

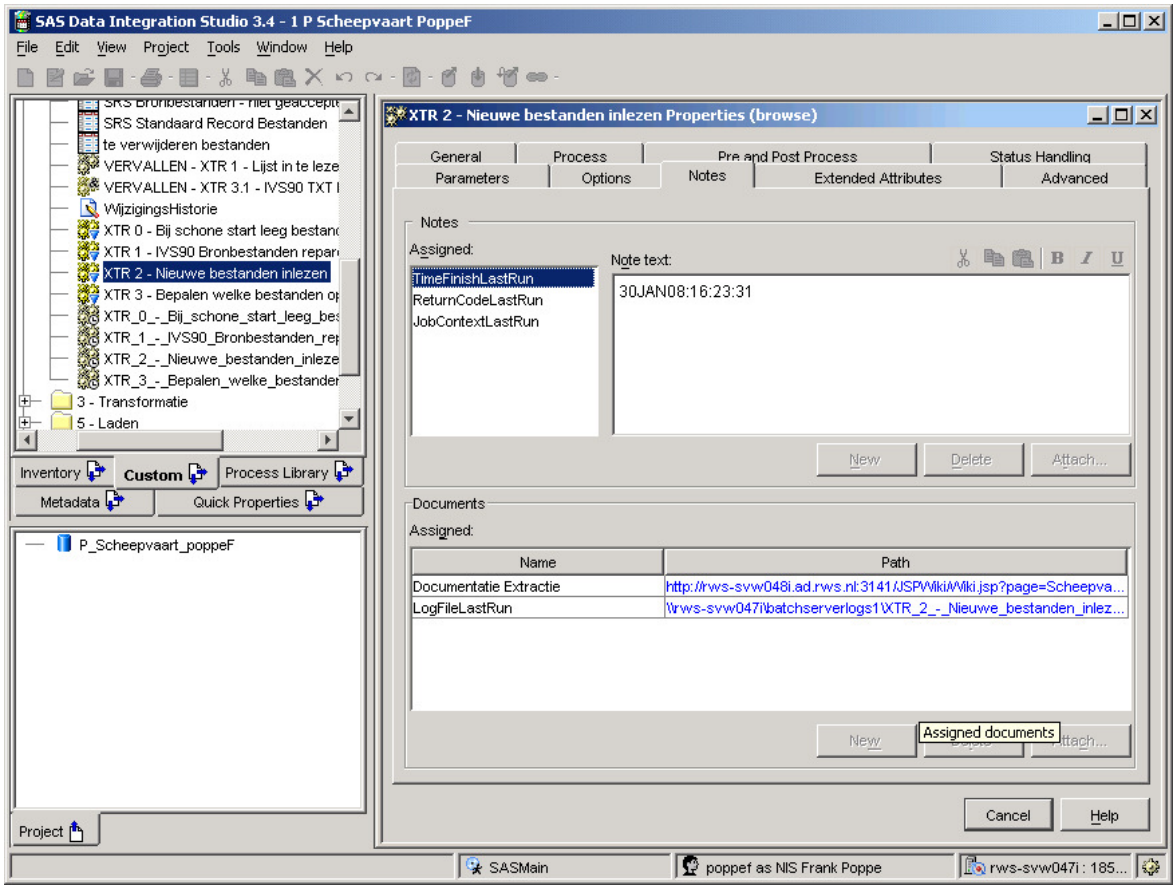
The information in the *Notes* tab will also be included as comment in the generated code (between `/*` and `*/`). This may be a bit confusing when the code is being deployed (i.e. converted into a .SAS file to be scheduled). That is because the specific information that is written into the file reflects the situation at the moment of deployment. That file the code can be used to run the code many times in the future. If all goes as intended, the information in the metadata will always reflect the situation after the last run, but the information written in the source file will still reflect the situation at the time of deployment.

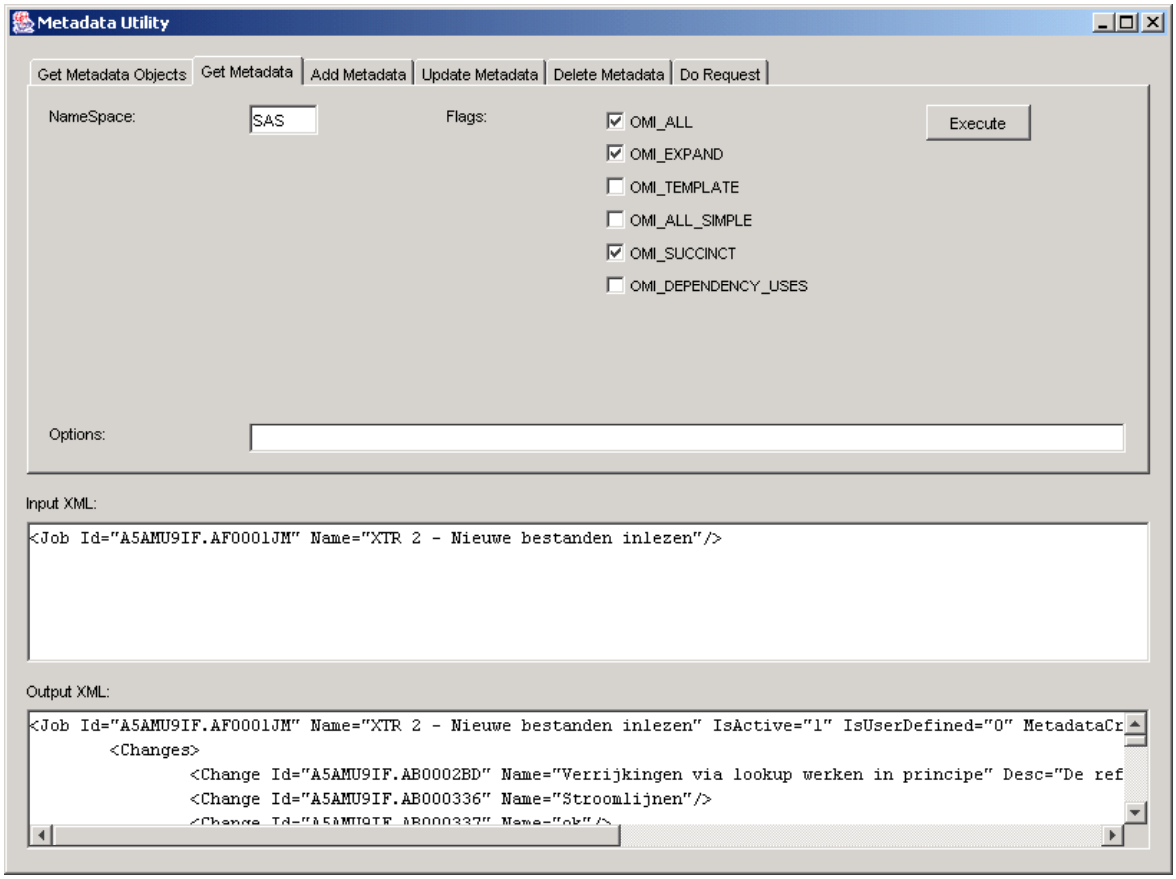
SAS MANAGEMENT CONSOLE

The SAS Management Console in itself does not show any added metadata information. However, there is the so-called Metadata Utility under the *Tools* menu. This is a stand-alone Java application with a rather crude user interface to access the metadata. It can also be used to change and delete metadata, which is rather dangerous way to do that.

There too one can walk through the associations in the metadata. What is shown is the XML version of the metadata. Instead of clicking hyperlinks one has to cut the id of the metadata object one is interested in from the output field and paste it into the input field. The screenshot (bottom of next page) shows the interface. Below also part of the XML is shown. Some parts of the resulting XML are not shown.

```
<Job Id="A5AMU9IF.AF0001JM" Name="XTR 2 - Nieuwe bestanden inlezen" IsActive="1"
IsUserDefined="0" MetadataCreated="24Apr2007:15:33:14"
MetadataUpdated="04Feb2008:09:56:50">
    <Changes> ... not shown ... </Changes>
    <Documents>
          <Document Id="A5AMU9IF.B50005EH" Name="Documentatie Extractie"/>
          <Document Id="A5AMU9IF.B5000669" Name="LogFileLastRun" Desc="Location of
logfile for job XTR 2 - Nieuwe bestanden inlezen"/></Documents>
    <JFJobs><JFJob Id="A5AMU9IF.A70000RU" Name="XTR_2_-
_Nieuwe_bestanden_inlezen"/></JFJobs>
    <JobActivities>
          <TransformationActivity Id="A5AMU9IF.A80002BE" Name="New Transformation
Activity"/></JobActivities>
    <Notes>
          <TextStore Id="A5AMU9IF.A900144P" Name="TimeFinishLastRun"
Desc="30JAN08:16:23:31"/>
          <TextStore Id="A5AMU9IF.A900144Q" Name="ReturnCodeLastRun" Desc="jobrc:
0"/>
          <TextStore Id="A5AMU9IF.A900144R" Name="JobContextLastRun" Desc="Context
in which the job was run"/></Notes>
    <Properties><Property Id="A5AMU9IF.AT0001OU" Name="DISVersion"/></Properties>
    <PropertySets> ... not shown ... </PropertySets>
    <ResponsibleParties><ResponsibleParty Id="A5AMU9IF.AQ00099D" Name="Modified
By"/></ResponsibleParties>
    <TransformationSources><Transformation Id="A5AMU9IF.B0000N5D"
Name="PreProcess"/></TransformationSources>
    <Trees><Tree Id="A5AMU9IF.A60000RT" Name="1 - Extractie"/></Trees>
    <TriggeredEvents><Event Id="A5AMU9IF.AW0007PW" Name="Job Status"
Desc="CodeCondition"/></TriggeredEvents>
</Job>
```


SAS Data Integration Studio 3.4 - 1 P Scheepvaart PoppeF
File Edit View Project Tools Window Help
SRS Standaard Record Bestanden
te verwijderen bestanden
VERVALLEN - XTR 1 - Lijst in te leze
VERVALLEN - XTR 3.1 - IVS90 TXT I
WijzigingsHistorie
XTR 0 - Bij schone start leeg bestanc
XTR 1 - IVS90 Bronbestanden repar
XTR 2 - Nieuwe bestanden inlezen
XTR 3 - Bepalen welke bestanden op
XTR_0_-_Bij_schone_start_leeg_bes
XTR_1_-_IVS90_Bronbestanden_rep
XTR_2_-_Nieuwe_bestanden_inleze
XTR_3_-_Bepalen_welke_bestander
3 - Transformatie
5 - Laden
Inventory
Custom
Process Library
Metadata
Quick Properties
P_Scheepvaart_poppeF
Project
XTR 2 - Nieuwe bestanden inlezen Properties (browse)
General
Process
Pre and Post Process
Status Handling
Parameters
Options
Notes
Extended Attributes
Advanced
Notes
Assigned:
TimeFinishLastRun
ReturnCodeLastRun
JobContextLastRun
Note text:
30JAN08:16:23:31
New
Delete
Attach...
Documents
Assigned:
Name
Path
Documentatie Extractie
http://rws-svw048i.ad.rws.nl:3141/JSPWiki/Wiki.jsp?page=Scheepva...
LogFileLastRun
\\rws-svw047i\batchserverlogs1\XTR_2_-_Nieuwe_bestanden_inlez...
New
Assigned documents
Attach...
Cancel
Help
SASMain
poppef as NIS Frank Poppe
rws-svw047i : 185...


Metadata Utility
Get Metadata Objects
Get Metadata
Add Metadata
Update Metadata
Delete Metadata
Do Request
NameSpace:
SAS
Flags:
OMI_ALL
OMI_EXPAND
OMI_TEMPLATE
OMI_ALL_SIMPLE
OMI_SUCCINCT
OMI_DEPENDENCY_USES
Execute
Options:
Input XML:
<Job Id="A5AMU9IF.AF0001JM" Name="XTR 2 - Nieuwe bestanden inlezen"/>
Output XML:
<Job Id="A5AMU9IF.AF0001JM" Name="XTR 2 - Nieuwe bestanden inlezen" IsActive="1" IsUserDefined="0" MetadataCr
<Changes>
<Change Id="A5AMU9IF.AB0002BD" Name="Verrijkingen via lookup werken in principe" Desc="De ref
<Change Id="A5AMU9IF.AB000336" Name="Stroomlijnen"/>

STATUS INFORMATION

From the SMC the LSF scheduler can be fed with all the requirements for the jobs:

- when to start a flow,
- how jobs within a flow are dependent on each other,
- etcetera.

But status information on those jobs is not available in the SMC. For that one has either to use the LSF interface on the SAS server (and then you need to be able to log on directly to the server), or one has to look for and search through the log files on the SAS server.
The fact that these items of information are not available from the SMC or any other SAS client was felt as a serious drawback.

As a solution some SAS code was developed that has to run at the end of each job. The SAS code determines the resulting status code of the job, the location of the log file of the running job, etc., and stores this information in attributes of the job (or more precise: the metadata object for that job).
The location of the log file preferably should be specified in such a way that the value of the attribute is clickable in most metadata clients. How this can be accomplished will be described in the paragraph on SMC settings.

STATUS HANDLING

The code is stored in a macro. No fancy macro tricks are being used; storing it in a macro just makes it easy to get the code to run after each job. A convenient way to do that is to use the *Status Handling* option. This can be set in DI Studio on the *Status Handling* tab of the job Properties dialog. There you can attach an *Action* to certain *Code Conditions.* There are several predefined actions, but there is also the *Custom* action. This custom action takes a parameter, which is inserted without further inspection by DI Studio into the code at the end of the job. So any valid SAS code is allowed there. To insert more elaborate code the most convenient way is to turn that into a macro, so that you can call the macro from the parameter field for the custom action.
Don't forget to put the macro into the *Autocall* library, which will typically be a location like `D:\SASEnvironment\-Lev1\SASMain\SASEnvironment\SASMacro`.

The Custom action should be attached to the *Job Status* Code Condition (there are several other Code Conditions like *Successful, Warnings, Errors, Table Truncated,* etc.). Before a DI Studio user can use a combination of Code Condition and Action this combination has to be entered using the *Configure Status Handling* dialog (under the Menu *Tools*). To do this you need Write Access in the Custom Repository.
It would have been convenient if the *Status Handling* options had made it possible to define a new default setting for each new job. Currently a developer has to add this setting for each new job.
Also it would have been nice if the parameter for the *Custom* action (the macro call `%js2md`) could have been specified as the default.

The code itself is given later in this paper, together with some short explanatory text. The current version of the code can be downloaded from http://frank-poppe.xs4all.nl/SAS.

ADDING INFORMATION TO THE METADATA

The information is added to the metadata in the form of new metadata objects that are associated with the *Job* object. An alternative would have been to use the *Extended Attributes* possibility that makes it possible to add any type of information to a metadata object.
Both options have there are advantages and disadvantages. I have chosen for the option of new metadata objects because that way the information can be given a certain role so that e.g. file locations can be used as hyperlinks. The value of an extended attribute is always just a string, with cannot be given a special meaning.
The disadvantage of separate objects is that the only possible association between these objects and the job do not cause the objects to be deleted when the job is being deleted. So these special purpose objects remain 'orphaned' in the metadata when a job is being deleted.
Incidentally, an extended attribute is, although not shown that way, in reality also a separate object, of type *Extension*, with an association of type *Extensions*. But these objects are deleted automatically when the associated object is being deleted.
I understand that version 9.2 will know the concept of *public* and *private* notes. A private note is visible only in relation to the object is has been created for, and will be deleted together with that object. This would remedy the current problem.

SETTINGS WITHIN THE SAS MANAGEMENT CONSOLE

When the file with the SAS code is being created also the command to run that SAS file in batch is begin generated. To do this a template is being used. That template can be changed using the SAS Management Console. It can be found in the following place.

- In the *Repository* field select the *Foundation,*
- fold out the *Server Manager,*
- select and fold out the *server context* (e.g. SASMain)
- fold out the *Logical SAS Data Step Batch Server*
- select the (physical) *SAS Data Step Batch Server*
- right click and select *Properties*,
- click the *Options* tab.

The *Command Line* field usually specifies a `sasbatch.bat` script to be executed ((under Windows, under other operating systems an analogous script will be specified). Through a number of other steps (also `.bat` scripts) SAS finally is being started in batch mode. The intermediate `.bat` scripts can attach a number of options to the final SAS command to be generated.

The Options tab also has a field *Logs Directory*. The content of this field becomes the first part of value of the `-log` parameter of the SAS command. To this will be added the name of the process, and to that a text based on the content of the field under *Advanced Options*. This field contains placeholders for date and time, which will be replaced by actual values at run time. An example of the content of this field is `_#Y.#m.#d_#H.#M.#s.log`.

The macro being discussed in this paper picks up the final value of the `-log` parameter and stores that in the metadata. As the metadata can be read and used from very different contexts and using different clients it is important to specify the location of the log file in a way that is meaningful from different contexts. In general this will mean that indications of physical drives (`E:\` etc.) are to be avoided, and that specifications using UNC notations (`\\server\share`) are to be preferred.

DESCRIPTION OF THE CODE

In this paragraph the full code is given, with some explanations.
The macro to call in the *Job Status* option dialog is `%js2md` (short for *Job Status to Meta Data*). It hardly uses macro syntax; essentially it is a DATA Step using Metadata functions. It uses a utility macro `%GetOrCreate`. This utility macro is described after the main macro. Also this utility macro does not use macro syntax, it simply repeatedly inserts DATA Step code. Its purpose is to check whether a metadata object associated with the job already exists, and when it doesn't, to create it.

WRITING JOB STATUS TO METADATA

The information that will be written into the metadata consists of:

- the current value of the `DATETIME` function (stored into object TimeFinishLastRun),
- the current value of the macro variable `&job_rc` (object ReturnCodeLastRun),
- the value of the macro variables `&metarepository` and `&sysuserid`, and the physical path to the base location from where the job was run (object JobContextLastRun),
- the value of the `LOG` option (object LogFileLastRun).

The following assumptions have been made about the code that DI Studio generates:

- The code for the job must set a value for the macro variable `&jobid`, containing the id of the job in the repository.
- The macro variables mentioned above must have gotten their values
- The necessary options to access the metadata for the repository have been set.

The version of the code given below contains a lot of `PUT` statement to see what is going on. This may change in a later version.

The first thing to do in the code is turn off 'syntax check mode' because it may have been set through previous errors. This would cause the SAS System only to check the syntax of the following code, not to run it. But in this case we want to run this code even if there had been serious errors, because that exactly the information we want to store in the metadata.

```
%macro js2md  ;
%let etls_syntaxcheck=%sysfunc(getoption(syntaxcheck));
options nosyntaxcheck;
```

The next thing is to determine the current default location in the SAS environment. Maybe there are other ways, but the following code puts it in the macro variable `&baseloc`.

```
libname baseloc  '' ;
%let baseloc = %sysfunc ( pathname ( baseloc ) ) ;
libname baseloc clear ;
```

Start the DATA step and define and initialize some variables.

```
data _null_ ;
    length jobId objID logFile objName $ 256 ;
    jobId = '' ;
    objName = '' ;
    objId = '' ;
```

Determine the complete jobId, this is more efficient to use later then the possibly abbreviated form that has been stored in the macro variable.

```
    rc = metadata_getNObj ( "&jobid" , 1 , jobID ) ;
    put 'complete jobID' @30 rc= jobID= / ;
```

Now use the macro `getOrCreate` to find or create an object that is linked through the association *Notes*, of type *TextStore* and with the name *TimeFinishLastRun*. The type of association and the object type follow from the metadata model: a *TextStore* is the best-suited object type to store this kind of information. The object *TimeStamp* would seem a more appropriate candidate to contain this information, but one of the disadvantage is that it isn't possible to surface it in the DI Studio dialog screens, as can be done with the *TextStore* object.
Notes is the way to associate a *TextStore* with a *Job*. The name has been chosen arbitrarily.
Then determine the current date and time, and store that information in the *StoredText* attribute.
Finally give the attribute *TextRole* the value *Note*, this is a signal for DI Studio to show the note (on the *Notes* tab of the *Job Properties* dialog).

```
    %getOrCreate ( Notes , textStore , TimeFinishLastRun ) ;
    now = datetime () ;
    rc = metadata_setAttr ( objID , 'StoredText' , put ( now , datetime. ) ) ;
    put 'set attrib StoredText' @30 rc= ;
    rc = metadata_setAttr ( objID , 'Desc' , put ( now , datetime. ) ) ;
    put 'set attrib Desc ' @30 rc= ;
    rc = metadata_setAttr ( objID , 'TextRole' ,  "Note"  ) ;
    put 'set attrib TextRole ' @30 rc=  ;
```

Now repeat this for an object with the name *ReturnCodeLastRun* and store the value of the macro variable `&job_rc`.

```
    %getOrCreate ( Notes , textStore , ReturnCodeLastRun ) ;
    rc = metadata_setAttr ( objID , 'Desc' ,  "jobrc: &job_rc"  ) ;
    put 'set attrib Desc ' @30 rc=  ;
    rc = metadata_setAttr ( objID , 'StoredText' ,  "jobrc: &job_rc"  ) ;
    put 'set attrib StoredText ' @30 rc=  ;
    rc = metadata_setAttr ( objID , 'TextRole' ,  "Note"  ) ;
    put 'set attrib TextRole ' @30 rc=  ;
```

And the same again for a TextStore named JobContextLastRun.

```
%getOrCreate ( Notes , textStore , JobContextLastRun ) ;
rc = metadata_setAttr ( objID , 'Desc' , "Context in which the job was run"  ) ;
put 'set attrib Desc ' @30 rc=  ;
rc = metadata_setAttr ( objID , 'StoredText' ,
       "repository: &metarepository, user: &sysuserid, basisloc: &baseloc"  ) ;
put 'set attrib StoredText ' @30 rc=  ;
rc = metadata_setAttr ( objID , 'TextRole' ,  "Note"  ) ;
put 'set attrib TextRole ' @30 rc=  ;
```

Now get a *Document* object with a *Documents* association. This object type has an *URI* attribute which in most clients becomes a hyperlink to its value. Sometimes the *URItype* attribute influences the behavior, through trial and error is was determined that the best value seems to be `"file:"`. There is no documentation on the role of these attributes.

```
%getOrCreate ( Documents , document , LogFileLastRun ) ;

Logfile = getOption ( 'log' ) ;
put 'current logfile is ' @30 logfile ;
rc = metadata_setAttr ( objID , 'Desc' ,
       "Location of logfile for job &etls_jobName" ) ;
put 'set attrib Desc ' @30 rc= ;
rc = metadata_setAttr ( objID , 'URI' ,  logfile  ) ;
put 'set URI' @30 rc= ;
rc = metadata_setAttr ( objID , 'URItype' ,  'file:' ) ;
put 'set URItype ' @30 rc= ;
```

Close the DATA step, and set the syntaxcheck option back to the value it had.

```
run ;
options &etls_syntaxcheck;
%mend ;
```

GET OR CREATE A METADATA OBJECT

The utility macro `%GetOrCreate` has three positional parameters, with the following meaning.
For the object `&jobId` (a macro variable with a value in the outer context)

1. look through association **`&association`**
2. for object of type **`&objType`**
3. named **`&objName`**

If such an object does not exist, create it.

```
%macro getOrCreate ( association , objType , objName ) ;
    put "Looking through association &association for object of type &objType named
&objName" ;
    rcAss = 1 ;
    n = 1 ;
    found = 0 ;
```

Start a DO WHILE loop through all association of the requested type.

```
do while ( rcAss > 0 ) ;
       rcAss = metadata_getNAsn ( jobId , "&association" , n , objID ) ;
       if rcAss > 0 then do ;
              put "association &association found" @30 rcass= objID= ;
```

For each found association determine the object type. The object type is part of the `objID`. These id's have the following general structure: `"OMSOBJ:ObjectType\RepositoryID.ObjectID"`. By using the double quote and the backslash as word delimiters the SCAN function will select the right part.

Then check this against the requested object type, and if it is the right type check the object name against the requested name. If that matches as well the loop can be exited.

```
                objType = scan ( objId , 2 , ':\' ) ;

                if upcase ( objType ) = upcase ( "&objType" ) then do ;
                       rc = metadata_getAttr ( objId , 'Name' , objName ) ;
                       put 'name associated object' @30 rc= objName= @ ;
                       if objName = "&objName" then do ;
                           put ' - found' ;
                              found = 1 ;
                              rcAss = -999 ;
                       end ; /* objName check */
                       else put ' - not the one...' ;
                end ; /* objType check */
          end ; /* rcAss > 0, i.e. something found */
          n+1;
end ; /* loop through the associations */
```

Now if the loop has been exited but the requested object has not been found create it.

```
if NOT found then do ; /* did not yet exist */
   /* create a new object associated with the job */
    rc = metadata_newObj ( "&objType" , objID , "&objName" ,
       "&metarepository" , jobID , "&association" );
    put "&objName created" @30 rc= objID= ;
end ;
```

Now either way the data step variable objID has got a valid value, and control can be returned.

```
%mend ;
```

CONCLUSION

Using the code discussed in this paper and activating it using the methods described will create a situation in which the different areas of information are brought together. Many cross-references between these areas will be possible through hyperlinks. From any textual information a hyperlink can be created using the Metadata Explorer (a web application) to an item in the metadata. Within DI Studio a 'document' can be attached to a job, which is in fact a link to Wiki-based information system.

For a job using the custom macro action some basic information on the last actual run will be visible in the metadata. The name of the SAS log file of that last run will be shown. Depending on the possibilities and settings of the client it will be possible to use that as a hyperlink to open that log file.

Although it still takes some actions from the developer to create this situation, it is a useful addition to the standard behavior of the SAS clients.

CONTACT INFORMATION

If you want to discuss the content of this paper, or have any suggestions or comments, please feel free to send me an email at Frank.Poppe@PWconsulting.nl.