Paper 3645-2019

# Making your Way through the Metadata Maze

Frank Poppe, Laurent de Walick, PW Consulting

## ABSTRACT

SAS® metadata contains extensive information about all elements of a SAS site. Important parts of that metadata are surfaced through clients such as SAS® Management Console and SAS® Data Integration Studio.

But a lot is not accessible that way. For example, SAS Data Integration Studio does not tell you if another table has a foreign key relation to a table you might delete.

A good understanding of the relations between different elements in the metadata can be acquired only by browsing through the metadata. You might want a list of all logins and the persons that use them.

This paper describes an easily installed SAS® Stored Process that enables the user to navigate through the entire metadata of a SAS®9 installation, using only the web browser. The metadata is also available for people who do not have any of those SAS clients (dependent on authorization, of course).

Information about jobs, transformation steps, tables read and written, users and their group memberships, and so on, are presented in a structured way.

This Metadata Explorer is available in the public domain, through www.pwconsulting.nl, and can be installed without any preconditions.

This paper describes the way the SAS Stored Process is organized and explains several technical tricks, like the STREAM procedure to create HTML and XML files on the fly; the XSL procedure to transform the XML results of a metadata query into a more easily processed form; a new ODS tagset for the REPORT procedure to produce folding tables using Bootstrap CSS, and so on.

## INTRODUCTION

This paper describes an easily installable SAS Stored Process that enables the user to navigate through the entire metadata of a SAS v9 installation, using only the web browser.

The metadata is the central repository of everything of a SAS v9 site. Important parts of that metadata are surfaced through clients as the SAS Management Console and DI Studio. But there is a hoard of other information that is not directly accessible. Often a good understanding of the relations between different elements in the metadata can only be acquired by browsing through the metadata. The metadata also stores a wealth of statistical information on your jobs and tables that can only be extracted if you have a good understanding of its relations.

To make it possible to browse the metadata without any requirements (except access to it, of course) we developed the web based PW Metadata Explorer, an easily installable SAS Stored Process.

The Stored Process is available in the public domain, through https://www.pwconsulting.nl/metadata-explorer-en/.

## WHAT'S THE POINT?

Anything and everything that makes up a SAS version 9 implementation is stored in the metadata. Which servers there are and what functions they have, the users that are defined and what authorizations they have, the tables that exist and where they

physically reside, the jobs that have been created and the transformations that make them up, etcetera. Everything is stored in metadata objects, and the objects all are associated to other objects. The next paragraph describes the concepts and the organization of the SAS metadata.

Much of that information can be viewed and changed through the different SAS client applications, through desktop applications as DI Studio and the SAS Management Console and web clients like SAS Studio and the SAS Environment Manager. But there also is a lot of information that cannot be viewed that way. It often can be useful to browse the metadata freely, in order to understand how different objects relate to each other.
And questions can arise that cannot be answered using the SAS clients. We will describe two such examples, but first we want to explain how the idea for a metadata explorer started.

With the first versions of SAS version 9 also came an unsupported Web Application that enabled users to navigate the metadata from a web browser. With the release of new versions of SAS, with changes in the metadata, this application at some point stopped working.

But the questions that could be answered with the tool continued to exist, of course.

At some point one of us decided to put together his own set of Stored Processes that provided that functionality. Later we joined forces, and this evolved in the current application, with more attention to the user interface and maintainability.

## METADATA: WHAT IS IT, AND HOW DO OBJECTS HANG TOGETHER?

Metadata is data that provides information about other data. In the SAS context metadata provides information about the configuration of the SAS Platform and location and structure of enterprise data. SAS provides common metadata services across SAS applications through the SAS Open Metadata Architecture. Using the metadata architecture different SAS applications can use and exchange the same metadata; making it easier to work together.

The metadata architecture consists of a metadata model, an API and a metadata server.

The metadata model provides classes and objects that define different types of application metadata. This information is stored in a repository. The metadata model defines valid relationships between metadata types (associations), uses the inheritance of attributes and associations to affect common behaviors, and uses subclassing to extend behaviors.

The SAS Open Metadata Interface (OMI) provides methods for reading and writing metadata and administering the SAS Metadata Server. The PW Metadata Explorer uses SAS procedures to access the API and present the metadata information.

The SAS Metadata Server is a server that provides centralized controlled access to SAS Metadata [SAS-3]

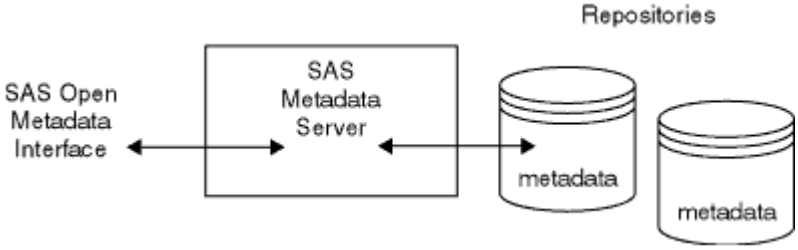



**Figure 1. Basic concept of the SAS Metadata**

The SAS Metadata Model is an object-oriented, hierarchical model. As hierarchical model every metadata type has a set of inherited and unique properties. The SAS Metadata Model provides metadata types in two namespaces: SAS and REPOS.

This paper will only focus on the SAS namespace which describes application elements, defined in about 160 metadata types [SAS-2].

Only a subset of the objects is visible as such in the SAS clients. These objects have the PublicType attribute set. The value of the PublicType attribute refers to a PublicType object of that name. The PublicType object describes the role of the object, and which icon should be used to show it.

Most objects types only have a single role, so they always have the same value for the PublicType attribute. An object of type Column e.g., has the PublicType attribute always set to Column. But some objects can have different roles, like the JFJob object, which can serve as a DeployedJob but also as a DeployedFlow.

PublicType objects themselves have the PublicType attribute set as well, not surprisingly to the value PublicType. They can be seen in the Systems folder.

Objects function that do not have the PublicType attribute set, function in the background, often containing properties of the objects that are shown, or defining the nature of the relation between public objects.

## USE CASES

In this paper we discuss two use cases of information that can be extracted from the SAS Metadata, but which are not readily available through standard SAS applications. For each use case we explain how to find the right starting point and understand how to browse through the objects and associations to uncover the requested information.

### CASE 1: WHERE IS A JOB USED?

We want to modify a job and want to know what processes are impacted by the change. We are certain this job is used in multiple flows. We also suspect that the job has been embedded in another job.

There is no easy method to determine all places where the job is used from the SAS Management Console (SMC) or DI Studio (DIS). In SMC we need to check every flow separately to see if the job is included in a flow and if that flow is a subflow of another flow. In DIS we need to open each job to see if the job has been embedded in another job.

#### The starting point

Any object can be found by name or ID. For jobs we can select them in the metadata tree in DIS and view the basic properties. Both the Name and Metadata ID are available as attribute.

Basic Properties

| Name | Value |
|---|---|
| Name | Metadata Explorer |
| Description | |
| Folder Location | /User Folders/Laurent de Walick/... |
| Checked Out By | |
| Tables Loaded | |
| Last Modified By | Laurent de Walick |
| Metadata Modified | 19-mrt-2019 15:27:41 (by Versio... |
| Metadata Created | 19-mrt-2019 15:27:41 |
| Metadata Archived | |
| Metadata Archived As | |
| Logical Type | Job |
| Metadata ID | A54I9U2T.BV000106 |
| Usage Version | 1.0 |

**Display 1. Basic Properties in DI Studio**

## The Objects and Associations

Next we discuss the objects and associations in a bit more detail, which will also give an impression of how the objects you see in your clients are represented by metadata objects in the background. The relations are shown schematically in Figure 2. Objects are represented by shaded boxes, the associations by light boxes surrounded by dashed lines. Associations always have two names, one for each direction. For example, the top right Job object has a *JFJobs* relation to the JFJob object, and that object has an *AssociatedJob* association back to the Job.

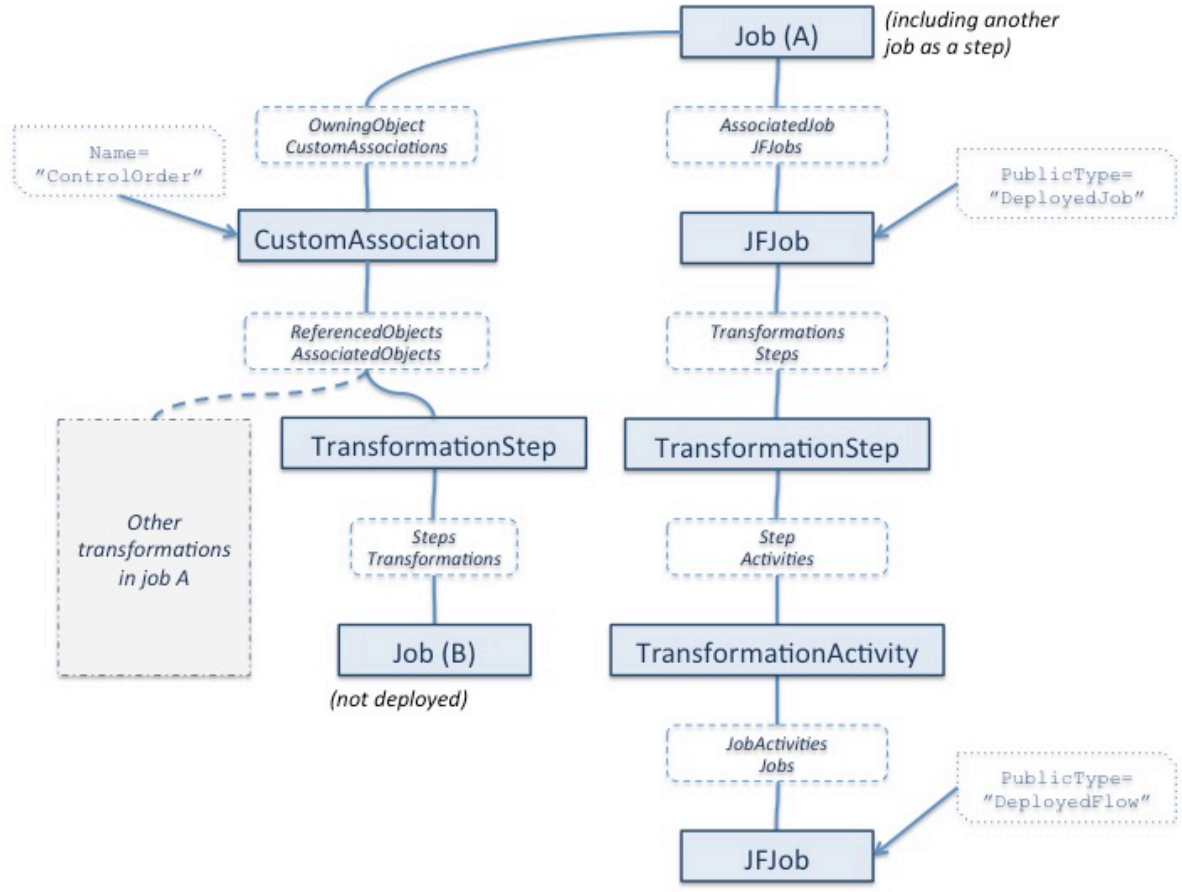



**Figure 2. Relation diagram for Job and Flow**

The right sequence show what metadata objects make up the deployment of a job. The job (A) is represented by a Job object, and after deployment the job has a JFJobs

association to a JFJob object with the PublicType attribute set to “DeployedJob”.
This is the Deployed Job object you see in the folder pane of DI Studio and the SAS Management Console.
From that object there is a Steps association to a TransformationStep object. This is the job node you see in the flow diagram in the Schedule Manager of the SAS Management Console.

There then remain two steps: an Activities association to a TransformationActivity object, and a Jobs association to a JFJob object with the PublicType attribute set to “DeployedFlow”. This last object is the Flow you see in the folder structure, and which you can schedule.

The left sequence shows in what way that job is made up out of transformation steps. There is a CustomAssociations link to a CustomAssociation object with a Name attribute value of “ControlOrder”. This object has an AssociatedObjects association to all steps in the job (this is one of the few cases where the order of associations is relevant). These steps are represented by TransformationStep objects, and they link through a Transformations association to the transformation itself. These are the SQL Joins, the Table Loaders, etc., but they can also be a Job, as shown here.

Note that the included Job (B) is not deployed, but will be part of the code that will be executed through the flow (bottom right in the figure).

Note also that nor in DI Studio, nor in the SAS Management Console, you can trace the relation from Job (B) to its inclusion in Job (A), or can you establish if the DeployedJob object for job (A) has been included in one or more flows.

## CASE 2: IS A COLUMN USED AS A FOREIGN KEY?

In DI Studio you can define primary and foreign key relations between tables. You then can use the standard options of the Table Loader transformation to enforce the constraints for those relations.

Sometimes a table may seem obsolete. DI Studio has the Analyze function to check the impact of a table, both forward and reverse. If a table is not used in any job it seems obvious that it can be deleted from your repository. But primary-foreign key relations do not show up in the analysis that DI Studio gives you. So e.g. a reference table, that is not read in any job, and is not populated through a job, might still serve foreign key restrictions for other tables. If the table is being deleted the Table Loader code for those jobs will end in error.

But, as anything, these relations are stored in the metadata, and can be traced with the PW Metadata Explorer.

The relations are stored both at the table level and at the column level. The quickest way to find if such relations exist is through the table relations. If necessary one can always go a level deeper to the columns involved.
Figure 3 shows the relations.

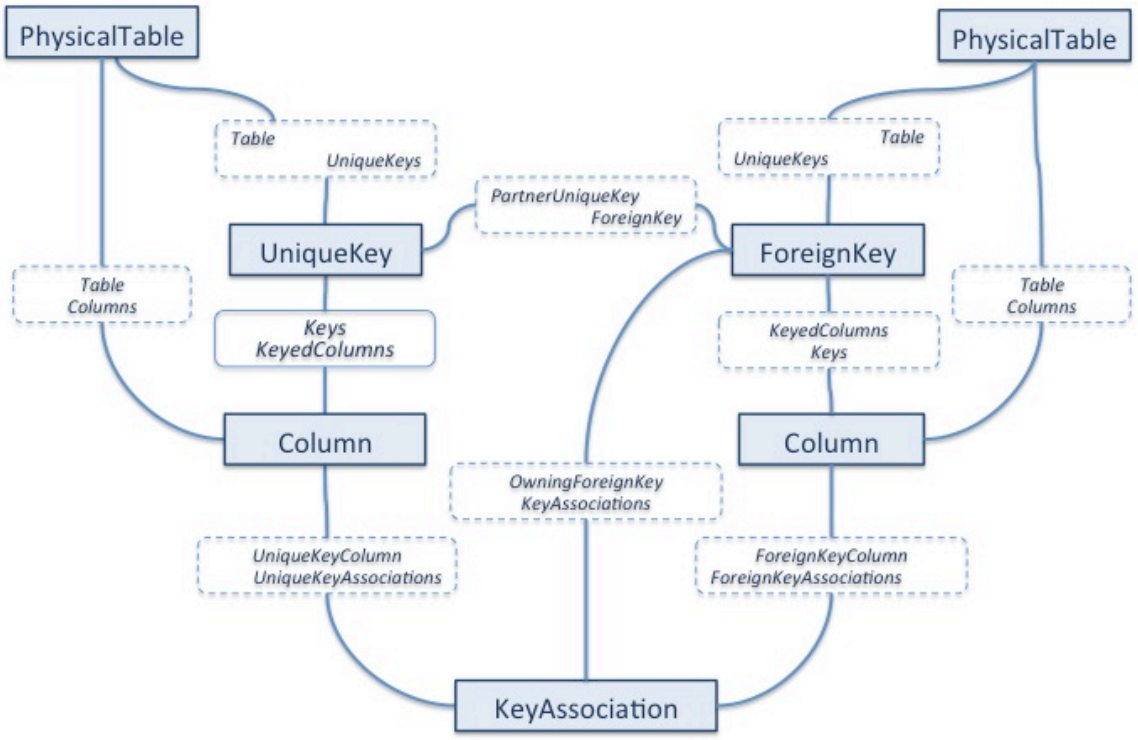



**Figure 3. Foreign key relations**

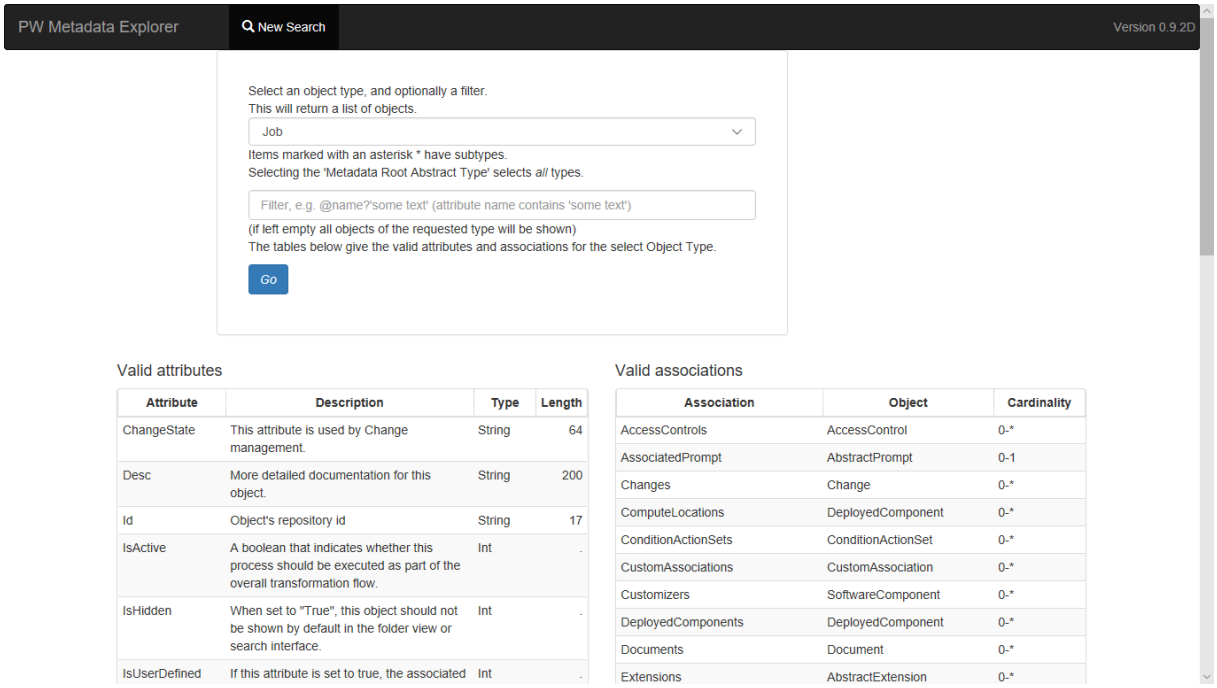
Tables are represented by PhysicalTable objects. If a primary key has been defined this will be a UniqueKeys association to a UniqueKey object. Now, if that key serves as a foreign key for another table, there will be ForeignKey association to a ForeignKey object.

If you then also want to know which table is involved, there will be Table association to the other table object.

## HOW DO YOU USE THE PW METADATA EXPLORER?

The PW Metadata Explorer functions in general as follows:

At the start the user is shown a form with a list of all the object types in the SAS metadata. In this screen shot the object type Job is selected. The two tables at the bottom will show the attributes and associations valid for the selected object.

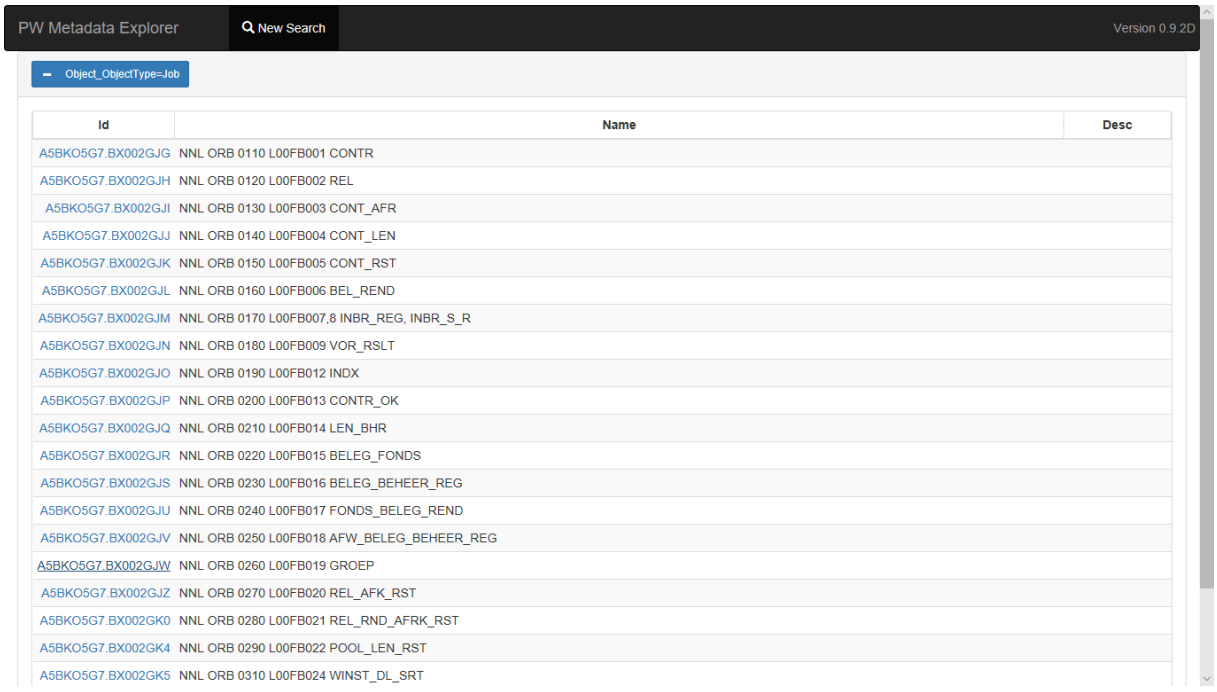



**Display 2. Opening screen of the PW Metadata Explorer**

The associations also display the cardinality of an association. Many associations have a cardinality of 0-∗, meaning that in that direction the object can have zero to multiple associations to the partner type mentioned. A 1-∗ cardinality means that there should be at least one associated object of that type.

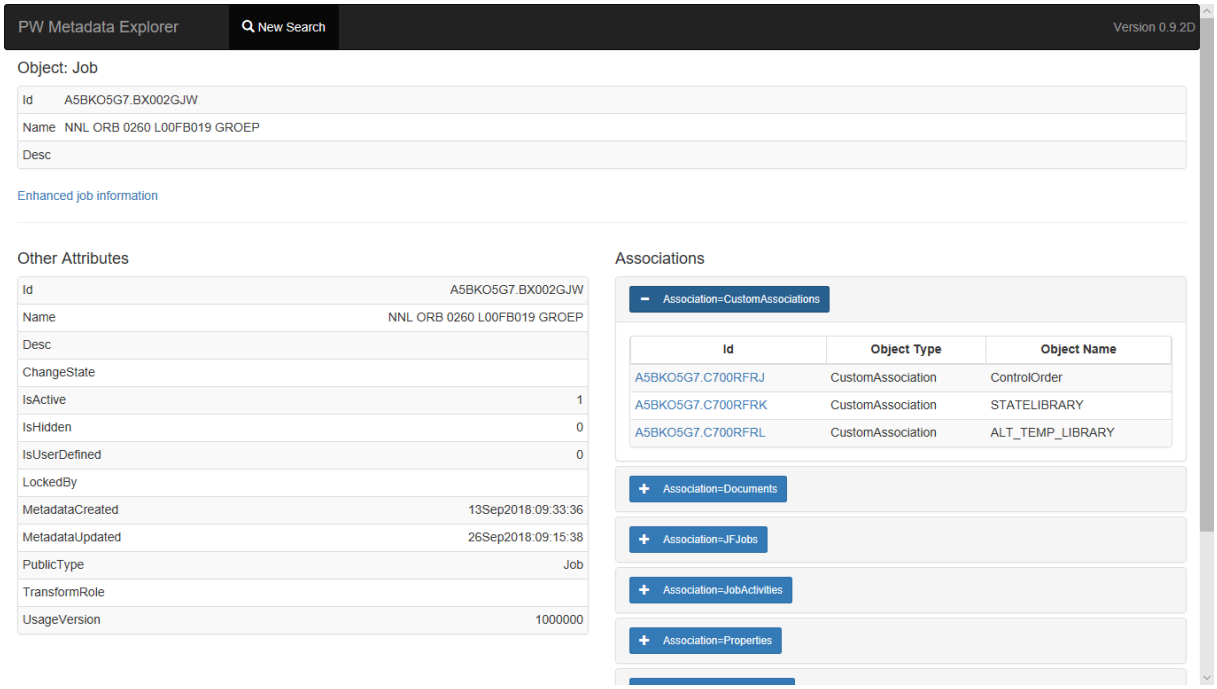
The cardinality can (and often is) different in the reverse direction: A Column object has a 1-1 association with a Table object: it cannot exist without a table, and can only belong to one table. But a Table has a 0-∗ association to Column objects: it can even exist without any coiumn.

Usually one will add a filter as well, and after clicking <OK>, a list will be shown of all objects of that type (with Id, Name and Description).




**Display 3. Result of a query for objects**

When clicking on any of those objects the next screen will show all attributes, and all associations to other objects.

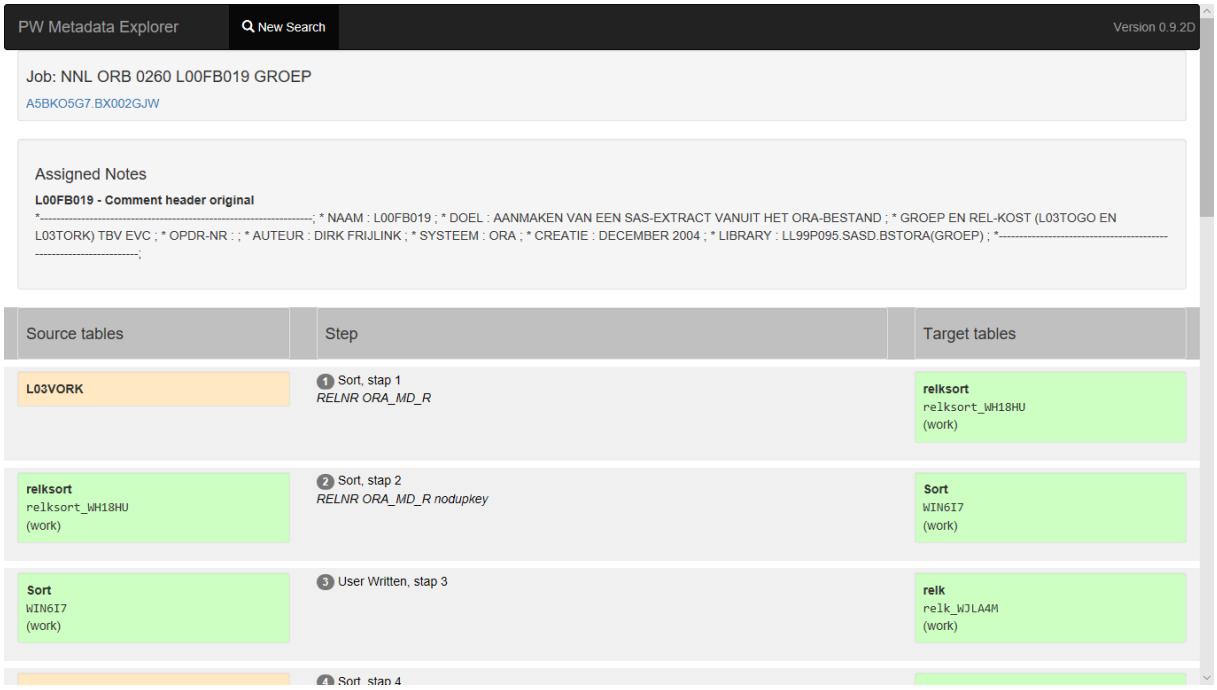



**Display 4. Information for a specific object**

The associations are grouped per type, and the groups can be expanded one at a time.

One can click on any of those associated objects, and that will show similar information for that object, ad infinitum.

For jobs so called enhanced information is available, showing all notes added to the job and the steps in the jobs, and showing all the steps in order of execution with the source and target tables for each step.




**Display 5. Enhanced information on Jobs, showing all tables read and written**

As on the DI Studio canvas, temporary tables are shown in green, while persistent, registered, tables are shown in orange.

## ORGANIZATION OF THE STORED PROCESS

In order to facilitate an easy installation process, we organized all the functions in a single stored process. Although the whole application consists of more than a dozen files with SAS code (plus several SAS data sets and other supporting files), there is only one file that has to be defined as 'the' Stored Process: PW_MDE_UI.sas. The combination of parameters that is used in calling the Stored Process will determine which other SAS files will be included.

The main task of this primary file thus is to determine, from the parameters, the combination of files to include. In general, the processing of a call consists of three phases:

- Collecting information using the METADATA procedure;
- Processing that information;
- Presenting the results.

Therefor the code of this first file ends with three %include statements, which refer to macro variables that have been assigned values based on the type of request. The code looks like this (a bit simplified here):

```
%include "&stpPath\&phase1..sas" ;
%include "&stpPath\&phase2..sas" ;
%include "&stpPath\&phase3..sas" ;
```

It can happen that in one of the first two phases things have to be changed. The code then just reassigns the macro variable. This is e.g. the case when the filter on the request for metadata objects does not return a result: there then is nothing to process, nor present.

The macro variable &stpPath is the physical location of the starting file PW_MDE_UI.sas, where also all the other files can be found.

Since SAS version 9.4M5 this location is known during execution through the automatic macro variable &SYSINCLUDEFILEDIR, but we did not develop in that version. And the application should work under older versions as well. So we needed some initial metadata requests to get at that location.

As the name of the Stored Process is known, the metadata object for the Stored Process can be found, and from there the metadata object that represents the source code, and eventually the name of the physical directory for that source code, which contains the full path and thus can be used as the value of &stpPath.

At various moments in the whole process pieces of HTML code may be produced that will become part of the page that is presented as result in the user's browser. These pieces are written to different temporary fileref's, which are included in the final result that is written to the fileref _webout. This is the fileref that is standard available to receive the HTML code that should be returned to the browser.

Combining these different fileref's is done using the PROC STREAM. This is the subject of the next paragraph.

## TECHNICAL DETAILS

### USING PROC STREAM

What PROC STREAM does, does not look very sensational: it takes the character stream following the BEGIN keyword on the PROC statement and copies everything to the designated file destination, until four semi-colons are found [SAS-1].

But while doing that the macro processor executes on the character stream, and that makes it a very flexible and powerful tool. Not only references to macro variables are resolved, also macro calls are being executed before resuming the processing of the character stream.

And those macro calls can have far reaching effects: within a macro SAS data sets can be opened and read and using the %sysfunc macro function also Data Step functions can be executed. Macro variables that are created that way will be passed back to the main process.

In this application PROC STREAM is used mainly for two purposes:

- It is used to collect the pieces of HTML that have been produced and present them orderly on the page that is being send back to the browser.
- At various point utility files have to be produced with slight variations depending on the query being processed, like an XML file containing a request for PROC METADATA, or an XML map to interpret the XML-code that Proc METADATA returns.

### Collect HTML pieces

PROC STREAM has two ways to include code from another source.

The macro function %INCLUDE will read the designated file, and it will be processed as if it were part of the character stream. I.e., macro references will be resolved, and macro's will be executed.

The directive READFILE on the other hand will insert a file without processing the content.

Both methods have been used.

### Creating utility files on the fly

The code to produce XML-maps, depending on the kind of result to be processed, looks like in the following example.

```
Proc Stream outfile=request ; BEGIN
<GetMetadataObjects>
   <ReposId>$METAREPOSITORY</ReposId>
   <Type>Job</Type>
   <Objects/>
   <Ns>SAS</Ns>
   <Flags>2436</Flags>
   <Options>
     <Templates>
      /* … template code not reproduced here */
     </Templates>
       <XMLSelect search="Job[@Id='&gObject']" />
    </Options>
</GetMetadataObjects>
;;;;
```

The macro variable (marked in the code), which contains metadata-id of the object for which information is requested, will be resolved by PROC Stream before writing to the output file.

## USING PROC XSL

The communication with the metadata through PROC METADATA works with XML: you define a request in XML, and an XML file with the results is returned. SAS offers the XML libname engine as solution to read data from XML files. An XMLMap can be provided to the XML engine, where it's defined how the XML should be interpreted into one or more SAS datasets and columns [SAS-4].

To use an XMLMap the structure of the XML that needs to be interpreted, must be known and fixed. The contents of the XML return by the PROC METADATA however, will vary depending on the query. This would require a very extensive XMLMap and lots of SAS code. To simplify the XML, we rely on Extensible Style Sheet Language (XSL). XSL is a W3C standard that offers the ability to transform an XML document into another XML document [W3C]. SAS provides out-of-the-box support for XSL through PROC XSL [SAS-1]. The simplification consists of turning the various elements (XML tags) for each object into an attribute for a standard element, in our case the objects element.

We use the following stylesheet. The relevant part of our stylesheet is as marked and will be explained below.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:strip-space elements="*"/>
<!-- identity transform -->
<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>
<xsl:template match="Objects/*">
    <Object>
        <xsl:attribute name="ObjectType">
            <xsl:value-of select ="name(.)"/>
        </xsl:attribute>
        <xsl:apply-templates select="@*|node()"/>
    </Object>
</xsl:template>
</xsl:stylesheet>
```

The match directive instructs PROC XSL to apply it to all elements found under the Objects element. For each element found a new element Object is created. A new attribute is added to the Object element containing the name of the element it is replacing. All other attributes are also copied into the new element.

The other directives make sure that all other elements are copied unchanged.

The resulting XML file now always contains elements named Object, regardless of the different object types in the original result.

If the original result contained the tag sequences (as a simplified example):

```
<Objects>
  <Job name="some job">
  <JFJob name="some deployed job">
</Objects>
```

After application of the stylesheet with PROC XSL this will result in:

```
<Objects>
  <Object ObjectType="Job" name="some job">
  <Object ObjectType="JFJob" name="some deployed job">
```

```
</Objects>
```

Now it is possible to use a straightforward XML map to interpret the results of a query, regardless of the objects being returned.

## BOOTSTRAP PACKAGE AND ODS TAGSETS

The basic lay-out of the web pages our application returns is set up with the Bootstrap-package [GetBootstrap]. In order to get everything set up properly we relied a lot on the examples and tutorials of W3Schools.com [w3schools].

The Bootstrap package supplies several themes that you can use and adjust to create a similar look and feel to all the elements that make up a web page (tables, buttons, etc.). The package also makes it possible to define a hierarchical grid of rows and columns which you can use to align the different parts of the web page. It is flexible, will adjust to resizing of the browser, and will take into account the properties of the different devices (pc, tablet, etc.).

The package offers the possibility of have collapsible elements. They can be grouped together, so that when one element is expanded, the other elements of the group are collapsed. This makes it possible to provide a large amount of information on a page while giving the user the opportunity to focus only a specific part of it. This would be useful on our pages, in at least two situations:

- The result of a query for objects can result in a long list of objects of different types;
- The list of associations for an object also can be long, and will almost always have several association types.

We use the Report procedure to generate the tables, using the features of that procedure to supply the clickable links and a BY statement to group the rows of the table. So we looked for a way to keep using PROC Report, but in such a way that the BY statement would produce these collapsible portions.

We found the solution in the ODS Tagsets. That is, we did not find a tagset that could be used, but we used the TEMPLATE procedure to create such a tagset. We took the CHTML tagset as basis, and changed different parts as needed. It is beyond the scope of this paper to explain in detail how tagsets operate, and how we applied it to create HTML code that uses the bootstrap functionality. We mention a few noteworthy points (the full code that creates the tagset is part of the downloadable zip-file).

A tagset operates on events. Events are the start and the end of a document, the start and the end of a table, the start and the end of a row, etcetera. For each start and for each end the tagset describes what should be done. That will be a combination of writing to the file destination, and some internal housekeeping.

The start of a table will, for an HTML destination, involve the writing of a <TABLE> tag, with all necessary information, and the end will produce a </TABLE> tag, a.o.

For the collapsible bootstrap elements two categories of events are important:

- The start and the end of a BY group, which will become the container for a set of a collapsible elements
- The start of a BY line, which signals the beginning of a new collapsible element within that container.

Two variables are used within the tagset to create identifiers in the HTML for the container and for the elements within that container: $accordion for the container, and $bygroup for the elements. The variable $accordion is initialized at the start of the document to a random value. This is because it is very well possible that a single HTML

page will be made up of several tables created using this code, and using random values the chance that they interfere is minimized.

The definition for the bygroup events is as follows.

```
define event bygroup ;
  start:
    eval $accordion $accordion +1 ;
      putl
        '<div class="panel-group" id="accordion'
        $accordion
        '">' ;
    eval $bygroup  0 ;
  finish:
    putl '</div>' ;
end ;
```

(The difference between the PUT and the PUTL statement is that the latter starts a new line at end of the output. A PUT statement does not.)

At the start first $accordion is incremented (so that different by groups within this document do not interfere), and then that value is used in the Id attributed of a  element. The variable $bygroup is initialized to 0.
The end event just closes the element with .
The class attribute “panel-group” activates the Bootstrap functions.

The definition for the byline events looks a bit more complicated, because more HTML code has to be produced.

```
define event byline;
  start:
    eval $bygroup   $bygroup +1 ;
    putl '<div class="panel panel-default">'  ;
    putl '<div class="panel-heading">';
    put  '<button type="button" class="btn btn-primary btn-sm acc"
            data-toggle="collapse" ' ;
    putl 'data-parent="#accordion' $accordion '" href="#bygroup'
          $accordion '_' $bygroup '">'
          VALUE '</button>'    ;
    putl "</div>"   ;
    put  '<div id="bygroup' $accordion '_' $bygroup
         '"class="panel-collapse collapse' ;
    put  ' in' / $bygroup = 1;
    putl '">'  ;
    putl '<div class="panel-body">'  ;
end;
```

First the variable $bygroup is incremented.

Two  elements are being produced. The first one is the line that will always be visible, with the <button> element that expands and collapses the next  with the actual content. The variable VALUE is the automatic variable that will contain the actual value of the By variable, and that is the value that will be displayed on the button (Display 4 shows how that works out).

The button is linked to the next  by constructing an id value using the $accordion and $bygroup values.

When the $bygroup value equals 1 the class “in” is added, meaning that it will be shown expanded initially (tagset code after a slash functions as an if statement).

There was one other aspect in the tagset that we had to take care of for this application: making sure that the ampersands and the less than and greater than symbols (& < >)

end up in the proper way in the resulting HTML. Some have to act in the HTML as such, so should remain &, < and >, while others have to be displayed as such, so should end up in the HTML as & < and &rt;. But if those strings are not introduced in the right way the characters will be encoded automatically once again, and the result will not be (.e.g.) < but < On the web page the user will see < where < was intended.

## CONCLUSIONS

The SAS metadata contains powerful information, and there is much more beyond the information that can be seen in the SAS clients. That information can be important for developers and other users.

The PW Metadata Explorer opens the SAS metadata for everybody authorized, without the need to install a client.

## REFERENCES


Bootstrap (n.d.). Retrieved from https://getbootstrap.com/

PW Consulting. (n.d.). Metadata Explorer. Retrieved from https://www.pwconsulting.nl/metadata-explorer-en/

SAS-1. (n.d.). Base SAS 9.4 Procedures Guide, Seventh Edition. Retrieved from https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.4&docsetId=proc&docsetTarget=titlepage.htm

SAS-2. (n.d.). SAS 9.4 Metadata Model: Reference. Available at http://support.sas.com/documentation/cdl/en/omamodref/67417/HTML/default/viewer.htm#titlepage.htm

SAS-3. (2018). SAS 9.4 Open Metadata Interface: Reference and Usage, Third Edition. Available at https://documentation.sas.com/api/docsets/omaref/9.4/content/omaref.pdf?locale=en

SAS-4. (2018). SAS 9.4 XMLV2 and XML LIBNAME Engines: User's Guide. Available at https://documentation.sas.com/api/docsets/engxml/9.4/content/engxml.pdf

W3C. (n.d.). Available at https://www.w3.org/Style/XSL/

W3Schools (n.d.). Bootstrap Tutorial. Available at https://www.w3schools.com/bootstrap/default.asp

Wikipedia. (n.d.). Metadata. Retrieved from https://en.wikipedia.org/wiki/Metadata


## CONTACT INFORMATION

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


Frank Poppe
PW Consulting
+31 6 2264 0854
Frank.Poppe@PWconsulting.nl
www.pwconsulting.nl

Laurent de Walick
PW Consulting
+31 6 2631 4544
Laurent.de.Walick@PWconsulting.nl
www.pwconsulting.nl