Paper 071-2010

Getting Your Metadata Using PROC METADATA

Edwin J. van Stein, Astellas Pharma Global Development Europe, The Netherlands

ABSTRACT

A lot of useful information from metadata is available to your programs through PROC METADATA. It can be hard to figure out how to extract the necessary data though. The purpose of this paper is to provide easy examples for some basic things you can get from metadata using PROC METADATA and XML maps. The examples that will be given were created on SAS® 9.1.3 on UNIX.

INTRODUCTION

When creating stored processes to be run through the SAS® Stored Process Web Application and managing users, logins and user groups with SAS® Management Console a lot of useful information is stored in metadata. This data is available to your programs through PROC METADATA or DATA step functions. At Astellas we mainly use this to get user group memberships from metadata to check which stored processes people are allowed to run and what data they should have access to.

The examples given were created on SAS® 9.1.3 on UNIX combined with SAS® Management Console 9.1 on Windows. The things described in this paper can also be done using DATA step functions or the SAS® Java Metadata Interface, but this is not within the scope of this paper.

GETTING REPOSITORY ID

In some cases it can be useful to know the ID's and names of the repositories present on the server. This can be done by sending an instruction formatted as XML (the XML-formatted method call) to the metadata server telling it to get all repositories:

```
proc metadata in='<GetRepositories>
                    <Type>Repository</Type>
                  </GetRepositories>';
run;
```

This is an example of PROC METADATA in its most basic form. The IN= argument tells the metadata server to get repositories from metadata, report objects with Type set to Repository and write the output to the log (OUT= argument is not set).

SERVER CONNECTION ARGUMENTS

The following server connection arguments can be set in the PROC METADATA call, but this is only needed if the corresponding system options are not usable:

- SERVER: the host name or IP address of the metadata server, this can be localhost if the SAS® session connects to the metadata server on the same computer. If not specified then the system option METASERVER= is used.
- PORT: the TCP port the metadata server is listening to. If not specified then the system option METAPORT= is used.
- USER: user ID on the metadata server used to extract the metadata. If not specified then the system option METAUSER= is used.
- PASSWORD: password for the user ID on the metadata server used to extract the metadata, this can be a password encoded using PROC PWENCODE. If not specified then the system option METAPASS= is used.
- PROTOCOL: the network protocol to connect to the metadata server, the only option currently is BRIDGE. If not specified then the system option METAPROTOCOL is used.

- REPOSITORY: the name of the repository used when the substitution variable $METAREPOSITORY is used in the XML-formatted method call. If not specified then the system option METAREPOSITORY is used.

Since all server connection arguments default to a system option, it's often easiest to set the system options, so these arguments don't have to be set in every PROC METADATA call. A useful exception is USER and PASSWORD. Not every user has the rights to see all relevant metadata. To ensure that all metadata is extracted it can be necessary to set USER and PASSWORD. In that case it's advisable to use PROC PWENCODE to encode the password:

```
proc pwencode in='my password';
run;
```

This will write a string like {sas001}bXkgcGFzc3dvcmQ= to the log which can be used as password. Of course it's still a good idea to take other safety precautions to prevent user ID's and passwords from falling in the wrong hands such as restricting access to programs and logs and restricting what gets written to the logs.

OUTPUT ARGUMENTS

The following output arguments can be set in the PROC METADATA call:

- OUT: fileref to store the output in, this can be a temporary fileref that is not physically saved on the server. If not specified then the output is written to the log.
- HEADER (NONE, FULL or SIMPLE): this specifies whether a header needs to be written to the output XML. The safest option is to set it to FULL, because the output XML will then contain an XML declaration that specifies encoding of the SAS® session that created the output.
- VERBOSE: tells SAS® to also print the XML-formatted method call in the log.

The HEADER output argument, combined with the specifying encoding in the FILENAME statement for the output file, can be especially useful when experiencing difficulties with strange characters stored in metadata.

CONVERTING THE OUTPUT XML TO SAS® DATA SETS

Using the information above, a physical XML file can be created that contains all repositories that the user sasadm has access to using the following code (assuming the password for sasadm is 'my password', without quotes, and is encoded using PROC PWENCODE):

```
filename outrepos "/sas/user/test/repos.xml" encoding="utf-8";

proc metadata userid="sasadm"
              password="{sas001}bXkgcGFzc3dvcmQ="
              in='<GetRepositories>
                     <Type>Repository</Type>
                  </GetRepositories>'
              header=full
              out=outrepos;
run;
```

The resulting XML file can be opened in most modern web browsers and will look something like Figure 1.

```
<?xml version="1.0" encoding="utf-8" ?>
- <GetRepositories>
    <Type>Repository</Type>
  - <Repositories>
      <Repository Id="A0000001.A5IEV5WH" Name="Foundation" Desc="" DefaultNS="SAS" />
      <Repository Id="A0000001.A51MZCKQ" Name="Test" Desc="" DefaultNS="SAS" />
    </Repositories>
  </GetRepositories>
```

Figure 1 Output XML as displayed by web browser

By creating an XML map the output from PROC METADATA can be converted to a SAS® data set. The XML map can be created using SAS® XML Mapper or by hand. Enough information can be found on the XML LIBNAME engine on the internet to help out. An example of an XML map used to get the ID and name of repositories from the above output is:

```
<?xml version="1.0" encoding="utf-8"?>
<SXLEMAP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="ReposMap"
version="1.2"
xsi:noNamespaceSchemaLocation="http://www.sas.com/xml/schema/sxle12.xsd">
    <TABLE name="repos">
        <TABLE-PATH syntax="XPath">/GetRepositories/Repositories/Repository</TABLE-
PATH>
        <COLUMN name="Id">
            <PATH syntax="XPath">/GetRepositories/Repositories/Repository@Id</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>200</LENGTH>
        </COLUMN>
           <COLUMN name="Name">
            <PATH
syntax="XPath">/GetRepositories/Repositories/Repository@Name</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>200</LENGTH>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

This XML map can then be used by the XML LIBNAME engine to get the data into a SAS® data set:

```
libname repos xml "/sas/user/test/repos.xml" xmlmap="/sas/user/test/repos.map";

proc sql noprint;
   create table work.repos as
   select * from repos.repos;
quit;
```

GETTING GROUP MEMBERSHIPS

Once the ID for the repository is known or the relevant system options are set correctly a lot of information from metadata becomes available. For instance e-mail addresses can be taken from metadata to use SAS® to send an e-mail to someone with an EMAIL FILENAME statement or the things people can run or do can be restricted by getting user group memberships. As long as the data is entered in metadata it is available.

The difficult part in getting the proper data from metadata is building the XML-formatted method call. To get group memberships there are some things that need to be specified:

- Group memberships are stored as IdentityGroups for persons in metadata. Persons (but also IdentityGroups) are objects in metadata, so PROC METADATA needs to be told to get objects using <GetMetadataObjects> tags.
- Repository ID needs to be specified when using <GetMetadataObjects>, but if the system option METAREPOSITORY is correctly set the substitution variable $METAREPOSITORY within <Reposid> tags can be used.
- To get persons with their subsequent group memberships, type needs to be set to Person within <Type> tags.
- <GetMetadataObjects> requires the namespace to be defined within <Ns> tags. Namespace can either be REPOS (types that describe repositories) or SAS (types that describe application elements such as tables and columns, but also users, groups, stored processes, etc.). In case of doubt just try the different namespaces, since there are only 2 possibilities.
- The metadata server needs to know what to do for all objects that are found. This is done by setting flags within <Flags> tags. To get all the properties for the requested objects (flag OMI_ALL = 1) and to execute a GetMetadata call for each object that is found (flag OMI_GET_METADATA = 256) the flags should be set to 257. The different OMI flags are described in the documentation from SAS®. When reading metadata specifying flags as 257 is often a safe choice, because it simply returns all information found.

The above results in the following XML-formatted method call:

```
IN='<GetMetadataObjects>
       <Reposid>$METAREPOSITORY</Reposid>
       <Type>Person</Type>
       <Ns>SAS</Ns>
       <Flags>257</Flags>
    </GetMetadataObjects>'
```

Using PROC METADATA with the XML-formatted method call will result in an XML file similar to Figure 2 (a lot of excess data not displayed for readability).

```
<?xml version="1.0" encoding="utf-8" ?>
- <GetMetadataObjects>
   <Reposid>A0000001.A5IEV5WH</Reposid>
   <Type>Person</Type>
   <Ns>SAS</Ns>
   <Flags>257</Flags>
  - <Objects>
    - <Person Id="A5IEV5WH.AR000001" Name="Person 1" ChangeState="" Desc="" LockedBy=""
       MetadataCreated="22Sep2008:11:17:48" MetadataUpdated="22Sep2008:11:17:48" Title="">
       .... Excess data not displayed ....
      - <IdentityGroups>
         <IdentityGroup Id="A5IEV5WH.A30000RZ" Name="Group 1" Desc="" />
         <IdentityGroup Id="A5IEV5WH.A3000004" Name="Group 2" Desc="" />
         <IdentityGroup Id="A5IEV5WH.A30000RX" Name="Group 3" Desc="" />
         <IdentityGroup Id="A5IEV5WH.A3000003" Name="Group 4" Desc="" />
       </IdentityGroups>
       .... Excess data not displayed ....
     </Person>
    - <Person Id="A5IEV5WH.AR000002" Name="Person 2" ChangeState="" Desc="" LockedBy=""
       MetadataCreated="22Sep2008:11:17:48" MetadataUpdated="22Sep2008:11:17:48" Title="">
       .... Excess data not displayed ....
      - <IdentityGroups>
         <IdentityGroup Id="A5IEV5WH.A3000004" Name="Group 2" Desc="" />
         <IdentityGroup Id="A5IEV5WH.A3000003" Name="Group 4" Desc="" />
       </IdentityGroups>
       .... Excess data not displayed ....
     </Person>
     .... Excess data not displayed ....
   </Objects>
 </GetMetadataObjects>
```

Figure 2 Output XML for group memberships

When converting the output XML to a SAS® data set the fact that every person can be member of multiple groups has to be taken into account:

- The <TABLE-PATH> tags determine the number of observations that will be in the data set. For instance setting path to /GetMetadataObjects/Objects/Person will result in only one record per person in the data set. To get a record per group membership path should be specified as /GetMetadataObjects/Objects/Person/IdentityGroups/IdentityGroup.
- Setting the <TABLE-PATH> to /GetMetadataObjects/Objects/Person/IdentityGroups/IdentityGroup will mean that any attributes taken from the <Person> tags will only be filled for the first IdentityGroup per person. To ensure that this does not happen retain="YES" can be added in the <COLUMN> tag in the XML map.

The above results in the following XML map to get a data set with only the name of persons and the names of the groups they are a member of:

```
<?xml version="1.0" encoding="utf-8"?>
<SXLEMAP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="GroupMap"
version="1.2"
xsi:noNamespaceSchemaLocation="http://www.sas.com/xml/schema/sxle12.xsd">
    <TABLE name="groups">
        <TABLE-PATH
syntax="XPath">/GetMetadataObjects/Objects/Person/IdentityGroups/IdentityGroup</TABL
E-PATH>
```

```
        <COLUMN name="Name" retain="YES">
            <PATH syntax="XPath">/GetMetadataObjects/Objects/Person/@Name</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>200</LENGTH>
        </COLUMN>
  <COLUMN name="Group">
            <PATH
syntax="XPath">/GetMetadataObjects/Objects/Person/IdentityGroups/IdentityGroup/@Name
</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>200</LENGTH>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

MAKING IT MORE EFFICIENT

In the examples above all attributes were extracted from metadata and then the XML map decided which data to actually convert to SAS® data sets. The XML-formatted method call can be made more efficient so that the output only contains the data needed. For instance to get group memberships for a specific person and his/her full name is known (in a stored process through the macro variable &_METAPERSON for example) then the XML-formatted method call can be constructed as follows:

- The basics like <GetMetadataObjects>, <Type> and <Ns> can remain unchanged.
- Within the <Flags> tag the following things can be specified:
    - a template will be supplied for PROC METADATA to fill (flag OMI_TEMPLATE = 4);
    - an XMLSelect element will be supplied to filter on (flag OMI_XMLSELECT = 128);
    - the metadata found should be returned (flag OMI_GET_METADATA = 256);
    - the XMLSelect element should be case-sensitive (flag OMI_MATCH_CASE = 512, if not specified then the XMLSelect element is case-insensitive).

  This results in flags being set to 900.
- Both the template and the XMLSelect element need to be within <Options> tags.
- In the template it can be specified that for Person only the Name attribute and the IdentityGroups should be returned and that for IdentityGroup (within Person that is, because Type is set to Person) only the Name attribute should be returned.
- In the XMLSelect element it can be specified that only data should be returned where the Name attribute for Person is a specific case-sensitive value (flag OMI_MATCH_CASE used).

The resulting XML-formatted method call is then (since the IN= argument is enclosed in single quotes in PROC METADATA the string to search for in the XMLSelect element is enclosed in two single quotes):

```
IN='<GetMetadataObjects>
       <Reposid>$METAREPOSITORY</Reposid>
       <Type>Person</Type>
       <NS>SAS</NS>
       <Flags>900</Flags>
       <Options>
          <Templates>
             <Person Name="">
                <IdentityGroups />
             </Person>
             <IdentityGroup Name=""/>
          </Templates>
          <XMLSelect search="@Name=''Person 1''"/>
       </Options>
    </GetMetadataObjects>'
```

Using this in PROC METADATA will result in output XML similar to Figure 3. The XML map to convert this to a data set remains unchanged.

```
<?xml version="1.0" encoding="utf-8" ?>
- <GetMetadataObjects>
    <Reposid>A0000001.A5IEV5WH</Reposid>
    <Type>Person</Type>
    <NS>SAS</NS>
    <Flags>900</Flags>
  - <Options>
    - <Templates>
      - <Person Name="">
          <IdentityGroups />
        </Person>
        <IdentityGroup Name="" />
      </Templates>
      <XMLSelect search="@Name='Person 1'" />
    </Options>
  - <Objects>
    - <Person Id="A5IEV5WH.AR000001" Name="Person 1">
      - <IdentityGroups>
          <IdentityGroup Id="A5IEV5WH.A30000RZ" Name="Group 1" />
          <IdentityGroup Id="A5IEV5WH.A3000004" Name="Group 2" />
          <IdentityGroup Id="A5IEV5WH.A30000RX" Name="Group 3" />
          <IdentityGroup Id="A5IEV5WH.A3000003" Name="Group 4" />
        </IdentityGroups>
      </Person>
    </Objects>
  </GetMetadataObjects>
```

Figure 3 Output XML for group memberships using a template and XMLSelect element

A WORD OF WARNING

In the examples above <GetMetadataObjects> was used. This will only extract metadata and not add or edit it. To add metadata <AddMetadata> should be used in the IN= argument of PROC METADATA. Also available are things like <UpdateMetadata> and <DeleteMetadata>, which do what their respective names say they do. Beware however that <AddMetadata>, <UpdateMetadata> and <DeleteMetadata> can seriously mess up the metadata, so it is advisable only to use this on a production server when absolutely certain that it will work as expected. If these three methods really need to be used then the OMI_TRUSTED_CLIENT flag should be set.

CONCLUSION

PROC METADATA can be very helpful in retrieving metadata. However, when creating XML maps to read in XML data a run without warnings or errors does not necessarily mean that you have all the data. <TABLE-PATH> plays a vital role in this and the retain attribute can help in filling in all data. Using a template and XMLSelect element can result in a very efficient PROC METADATA call that returns only the data needed. When it is unclear what data and attributes are available it can help to first simply extract everything for an object (with flags set to 257) and then based on the output XML start building an efficient XML-formatted method call (containing a template and an XMLSelect element).

ACKNOWLEDGMENTS

Thanks to Phil Mason for proof reading this paper.

RECOMMENDED READING

- SAS 9.1.3 Open Metadata Interface: Reference, Second Edition
- SAS 9.1.3 Open Metadata Interface: User's Guide

CONTACT INFORMATION

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

Edwin J. van Stein
Clinical Data Science - Clinical Programming
Astellas Pharma Global Development Europe
Elisabethhof 19 P.O.Box 108
2350 AC Leiderdorp
The Netherlands
edwin.stein@eu.astellas.com / ejvanstein@gmail.com