Paper AD08-2011

Leveraging the SAS® Open Metadata Architecture

Ray Helm & Yolanda Howard, University of Kansas, Lawrence, KS

Abstract

In the SAS Enterprise BI and Data Integration environments, the SAS Metadata Model contains a wealth of information about system configuration, data objects, report metadata, users, groups, and security authorizations. The ability to query the metadata adds a powerful tool to the SAS inventory and can be used for a wide range of purposes including surfacing metadata related to the data delivered to end users, adding relevant information to metadata objects, and performing bulk alterations to the metadata itself. This paper will provide an overview of the key tools for leveraging the Open Metadata Architecture including: the PROC METDATA procedure, the Metadata Browser utility, and the SAS XML Mapper. We will present examples of how we have employed these tools for bulk modifications of stored processes/application server associations and for generating a Data Elements Dictionary for use by our Data Warehouse consumers.

Introduction

The SAS Metadata Server is a crucial component of the SAS Enterprise BI Server, Enterprise Data Integration Server, and many of the industry specific solutions offered by SAS. The SAS metadata holds virtually all of the critical information including: system architecture; user authentication and authorization; data storage locations and table structure. The SAS Open Metadata Architecture provides the capacity to extract this information into reports or other data systems, generate reports on specific aspects of the system (such as user authorizations), and integrate the metadata with data from other enterprise systems to create a comprehensive view of the relationship between the business intelligence data and ERP data.

Additionally, the Open Metadata Architecture allows metadata to be updated programmatically. This can be a powerful tool for the system administrator. Routine tasks can be automated, and large, complex administrative operations can be performed in a structured manner with minimal risk of errors due to operator oversight.

This paper will provide two examples that take advantage of the SAS Open Metadata Architecture. The first example demonstrates a system administration task performed programmatically as part of a system migration. The second shows how the metadata from an Enterprise Data Integration system can be combined with metadata from the source ERP systems to provide a comprehensive Data Elements Dictionary.

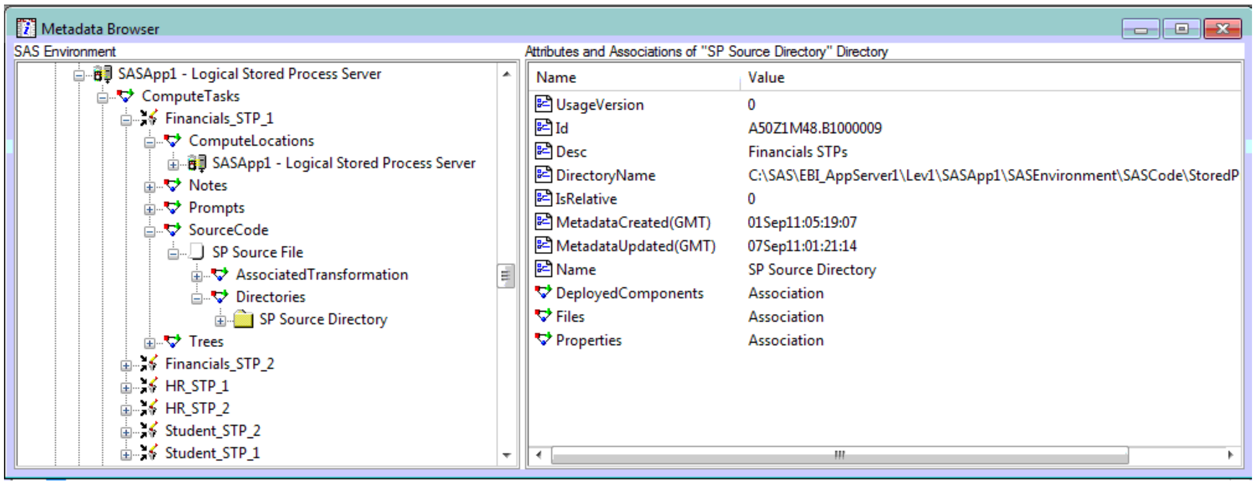
Useful Tools

There are a few useful tools for working with the SAS Open Metadata Interface. The Metadata Browser feature of the SAS Foundation software is essential for visualizing the metadata objects, identifying attributes and associations, and finding specific objects of interest. Our methods rely heavily on the PROC METADATA procedure which sends requests and receives responses using XML. The SAS XML Mapper is extremely useful in generating XML map files and SAS code for reading the XML response files into SAS data sets. It should be noted that it is also possible to perform the operations described here using SAS DATA step functions which do not require any XML knowledge or tools.

Using the Open Metadata Architecture in System Administration

As part of our migration from a SAS 9.1.3 Business Intelligence Platform to SAS 9.2, we were also upgrading our hardware architecture from a two server (one mid-tier and a combined metadata/application server) to a three server design. In order to employ the SAS Migration Utility, the first step required a migration from the two server architecture to an identical two server environment. This created a 9.2 system with a single server running both the Metadata Server and Application Server (SASMain) components. The installation and configuration of the new Application Server (SASApp) was independent of the migration. Rather than manually reassign over 150 stored processes from the old SASMain Application Server to the new SASApp Application Server, the SAS Open Metadata Interface tools were used to identify the stored processes in the metadata and reassign them to the new application server. Additionally, the directory metadata objects that stored the paths of the SAS programs executed by the stored processes were also reassigned to SASApp (the actual program files were moved manually). While this saved us quite a bit of tedious work reassigning stored processes to servers using the SAS Management Console, more importantly, it retained the associations between the stored processes and their links in the Information Delivery Portal. Exporting the stored processes from one application server and importing them into the other would have broken the association between the portal content and the stored process, requiring these associations to be rebuilt manually.

For the purposes of this paper, this migration scenario was recreated on a test Enterprise BI Server installation on a Windows workstation with a metadata instance containing just a few stored processes all linked to a single application server (SASApp1). A second application server (SASApp2) was also installed on the same workstation. Figure 1 shows the Metadata Browser view of the stored processes associated with the SASApp1 – Logical Stored Process Server. The Financials_STP_1 stored process has been expanded to show the ComputeLocations association and displays the attributes and associations pane for the SP Source Directory. The directory shows that the physical location of the SAS program associated with the stored process is also under the SASApp1 installation directory.


Metadata Browser
SAS Environment
Attributes and Associations of "SP Source Directory" Directory

```
SASApp1 - Logical Stored Process Server
  ComputeTasks
    Financials_STP_1
      ComputeLocations
        SASApp1 - Logical Stored Process Server
      Notes
      Prompts
      SourceCode
        SP Source File
          AssociatedTransformation
          Directories
            SP Source Directory
      Trees
    Financials_STP_2
    HR_STP_1
    HR_STP_2
    Student_STP_2
    Student_STP_1
```

Name	Value
UsageVersion	0
Id	A50Z1M48.B1000009
Desc	Financials STPs
DirectoryName	C:\SAS\EBI_AppServer1\Lev1\SASApp1\SASEnvironment\SASCode\StoredP
IsRelative	0
MetadataCreated(GMT)	01Sep11:05:19:07
MetadataUpdated(GMT)	07Sep11:01:21:14
Name	SP Source Directory
DeployedComponents	Association
Files	Association
Properties	Association

Figure 1. A Metadata Browser view of the SASApp1 - Logical Stored Process Server showing the stored processes under the ComputeTasks and the Attributes and Associations of the "SP Source Directory" Directory for the Financials_STP_1 stored process

Identifying Metadata to be Updated

Before you can perform any updates on a metadata object, the unique identifier of the object must be obtained. In this case, the metadata identity of all stored processes associated with the SASApp1 - Logical Stored Process Server need to be determined. One method for obtaining this is to create an XML input file with the appropriate request syntax and submit this using the PROC METADATA procedure. The XML below returns the Id and Name attributes for all ClassifierMaps (stored processes are ClassifierMap metadata objects) associated with the SASApp1 – Logical Stored Process Server:

```
<GetMetadataObjects>
  <Reposid>$METAREPOSITORY</Reposid>
  <Type>LogicalServer</Type>
  <Objects/>
  <NS>SAS</NS>
  <!-- set Flags for OMI_XMLSELECT(128)+OMI_GET_METADATA(256)+OMI_TEMPLATE(4) -->
  <Flags>388</Flags>
   <Options>
    <XMLSELECT search="*[@Name='SASApp1 - Logical Stored Process Server']" />
     <Templates>
       <LogicalServer Id="" Name="" >
         <ComputeTasks />
       </LogicalServer >
         <ComputeTasks >
           <ClassifierMap />
         </ComputeTasks >
           <ClassifierMap Id="" Name="" />
    </Templates>
    </Options>
</GetMetadataObjects>
```

The above XML was saved to “C:\sasfiles\SASApp_STPSvr_Request.xml” and sent to the metadata server as shown here:

```
/* Set Metadata Connectiion */
options metaserver="Mymetaserver" metaport=8563 metaprotocol=bridge
        metauser="mrhelm" metapass="*****"
        metarepository="Foundation";

/* GetMetadataObject request  file */
filename request “C:\sasfiles\SASApp_STPSvr_Request.xml” lrecl=1024;
/* Metadata response output file */
filename response "C:\sasfiles\SASApp_STPSvr_Response.xml" lrecl=1024;
proc metadata in=request out=response;
run;
```

This generates an XML file, "C:\sasfiles\SASApp_STPSvr_Response.xml" containing the results of the GetMetadataObjects query.

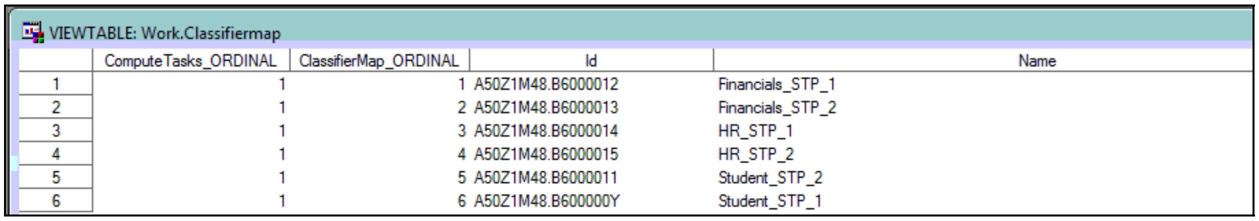
Loading the XML Response File into SAS

The next step is to convert the XML response into a SAS data set for use in constructing a set of instructions to update the metadata. The SAS XML Mapper software makes short work of constructing the necessary SAS code to read the response XML data into SAS data sets. All that needs to be done is to open the XML response file and use the “Automap using XML” tool to generate an XML Map. Then save the XML Map file. Once this is complete, go to the SAS Code Example tab and uncheck all of the “Show” options except for “Copy to WORK” and either save the code as a separate SAS program or copy and paste it into the SAS Program Editor. The code generated will create SAS data sets based on the response XML in the session WORK library:

```
/********************************************************************************
 *  Generated by XML Mapper, 902000.3.6.20090116170000_v920
 ********************************************************************************/

/*
 *  Environment
 */
filename  SASApp 'C:\sasfiles\SASApp_STPSvr_Response.xml';
filename  SXLEMAP 'C:\sasfiles\SASApp_STPSrv.map';
libname   SASApp xml xmlmap=SXLEMAP access=READONLY;

/*
 *  Local Extraction
 */

DATA GetMetadataObjects; SET SASApp.GetMetadataObjects; run;
DATA Objects; SET SASApp.Objects; run;
DATA LogicalServer; SET SASApp.LogicalServer; run;
DATA ComputeTasks; SET SASApp.ComputeTasks; run;
DATA ClassifierMap; SET SASApp.ClassifierMap; run;
DATA Options; SET SASApp.Options; run;
DATA XMLSELECT; SET SASApp.XMLSELECT; run;
DATA Templates; SET SASApp.Templates; run;
DATA LogicalServer1; SET SASApp1S.LogicalServer1; run;
DATA ComputeTasks1; SET SASApp1S.ComputeTasks1; run;
DATA ClassifierMap1; SET SASApp1S.ClassifierMap1; run;
```

Building and submitting the UpdateMetadata request

The data set ClassifierMap contains the necessary metadata information (specifically, the metadata Id) to build metadata update statements to re-associate each stored process to the SASApp2 – Logical Stored Process Server.


VIEWTABLE: Work.Classifiermap

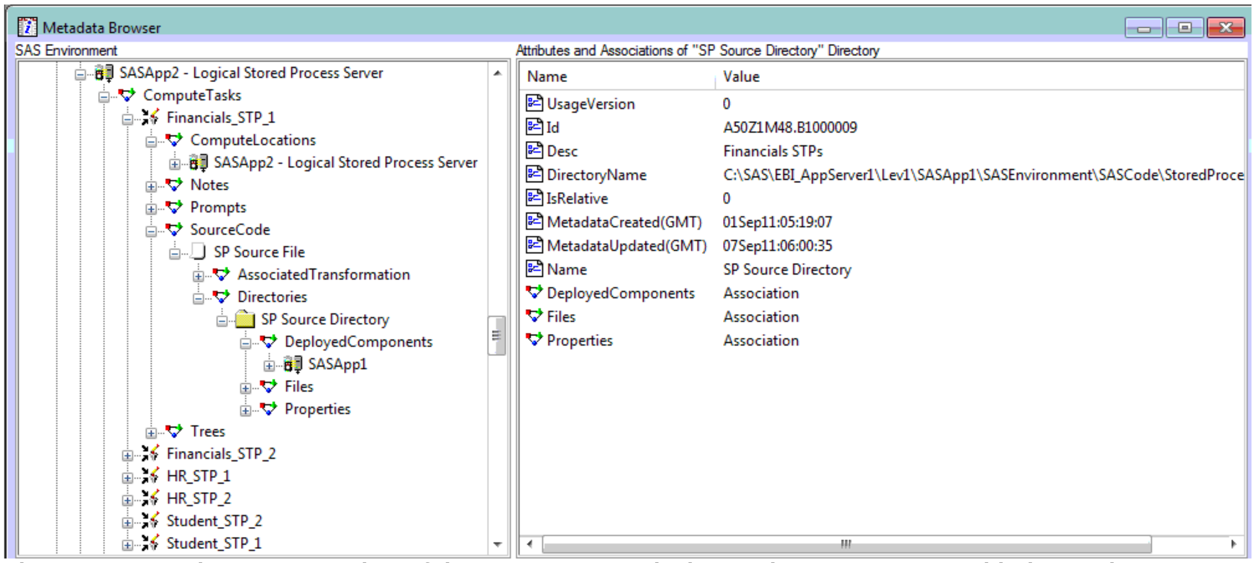
	ComputeTasks_ORDINAL	ClassifierMap_ORDINAL	Id	Name
1	1	1	A50Z1M48.B6000012	Financials_STP_1
2	1	2	A50Z1M48.B6000013	Financials_STP_2
3	1	3	A50Z1M48.B6000014	HR_STP_1
4	1	4	A50Z1M48.B6000015	HR_STP_2
5	1	5	A50Z1M48.B6000011	Student_STP_2
6	1	6	A50Z1M48.B600000Y	Student_STP_1

Figure 2. The ClassifierMap Data Set

The following code uses the ClassifierMap data set to build an UpdateMetadata XML file which is then submitted using the PROC METADATA procedure. The ObjRef and Name values for the SASApp2 – Logical Stored Process Server were obtained using the Copy URI to Clipboard and Copy features of the Metadata Browser tool:

```
filename InReq "C:\sasfiles\UpdateClassifierMaps.xml" lrecl=1024;
data _null_;
set ClassifierMap END=OVER;
file InReq;

if _n_=1 then put
'<Multiple_Requests>' /
  '<UpdateMetadata >' /
  '  <Metadata >';

put
  '    <ClassifierMap Id="' Id '" >' /
  '      <ComputeLocations Function="Replace"> ' /
  '        <LogicalServer ObjRef="A50Z1M48.AT00000B" Name="SASApp2 - Logical
Stored Process Server" /> ' /
  '      </ComputeLocations >' /
  '    </ClassifierMap > ';

if Over then put
  '  </Metadata >' /
  '  <Reposid >$METAREPOSITORY</Reposid >'/
  '  <NS>SAS</NS>' /
  '  <!-- OMI_TRUSTED_CLIENT (268435456) OMI_RETURN_LIST (1024) -->'/
  '  <Flags>268436480</Flags>' /
  '  <Options /> '/
  '</UpdateMetadata >'
'</Multiple_Requests >'
  ;
run;

filename result "C:\sasfiles\UpdateClassifierMaps_result.xml" lrecl=1024;
proc metadata in=InReq out=result;
run;
```

Figure 3 shows the stored processes associated to the SASApp2 – Logical Stored Process Server after the PROC METADATA execution. Note that the directory location of the SP Source Directory still points to the SASApp1 Application Server installation path and is still associated with the SASApp1 server. At this point, the stored processes have one foot in the SASApp2 server and one in SASApp1 and execution would be problematic. Another set of metadata queries and updates is necessary to fully disassociate the stored processes from the SASApp1 Application Server.


Metadata Browser
SAS Environment
Attributes and Associations of "SP Source Directory" Directory

```
SASApp2 - Logical Stored Process Server
  ComputeTasks
    Financials_STP_1
      ComputeLocations
        SASApp2 - Logical Stored Process Server
      Notes
      Prompts
      SourceCode
        SP Source File
          AssociatedTransformation
          Directories
            SP Source Directory
              DeployedComponents
                SASApp1
              Files
              Properties
      Trees
    Financials_STP_2
    HR_STP_1
    HR_STP_2
    Student_STP_2
    Student_STP_1
```

Name	Value
UsageVersion	0
Id	A50Z1M48.B1000009
Desc	Financials STPs
DirectoryName	C:\SAS\EBI_AppServer1\Lev1\SASApp1\SASEnvironment\SASCode\StoredProce
IsRelative	0
MetadataCreated(GMT)	01Sep11:05:19:07
MetadataUpdated(GMT)	07Sep11:06:00:35
Name	SP Source Directory
DeployedComponents	Association
Files	Association
Properties	Association

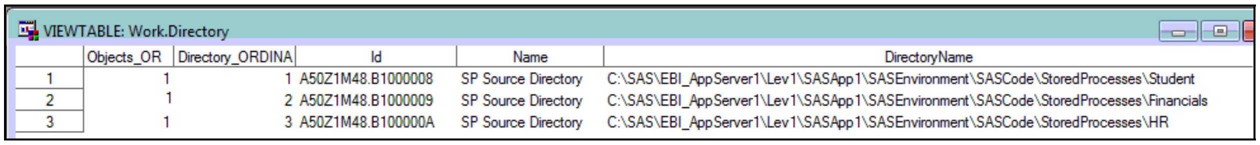
Figure 3. A Metadata Browser view of the SASApp2 - Logical Stored Process Server with the newly associated stored processes

Updating the Stored Process Source Code Directory Metadata

Updating the server and directory for each of the stored process source directories follows the same basic steps that were used in updating the server associations. An XML file is created containing a GetMetadataObjects request to retrieve the metadata Id of all Directory objects having a Name attribute of “SP Source Directory”. In addition to the Id, the DirectoryName attribute is also requested as this is the physical path on the application server where the SAS code resides:

```
<GetMetadataObjects>
    <Reposid>$METAREPOSITORY</Reposid>
     <Type>Directory</Type>
    <Objects/>
    <NS>SAS</NS>
    <!-- OMI_XMLSELECT(128)+OMI_GET_METADATA(256)+OMI_TEMPLATE(4) -->
    <Flags>388</Flags>
    <Options>
   <XMLSELECT search="*[@Name='SP Source Directory']" />
   <Templates>
       <Directory Id="" Name="" DirectoryName="" />
   </Templates>
    </Options>
</GetMetadataObjects>
```

This XML file is submitted using the PROC METADATA procedure in the same fashion as previously described to obtain an XML response file from the metadata server. The methods for converting the XML response into a SAS data set using the SAS XML Mapper are also the same as those used for identifying the stored processes and will not be described in detail here. The end result of the process is a SAS data set WORK.DIRECTORY containing the Id, Name, and DirectoryName for each of the stored process directories (Figure 4).


VIEWTABLE: Work.Directory

	Objects_OR	Directory_ORDINA	Id	Name	DirectoryName
1	1	1	A50Z1M48.B1000008	SP Source Directory	C:\SAS\EBI_AppServer1\Lev1\SASApp1\SASEnvironment\SASCode\StoredProcesses\Student
2	1	2	A50Z1M48.B1000009	SP Source Directory	C:\SAS\EBI_AppServer1\Lev1\SASApp1\SASEnvironment\SASCode\StoredProcesses\Financials
3	1	3	A50Z1M48.B100000A	SP Source Directory	C:\SAS\EBI_AppServer1\Lev1\SASApp1\SASEnvironment\SASCode\StoredProcesses\HR

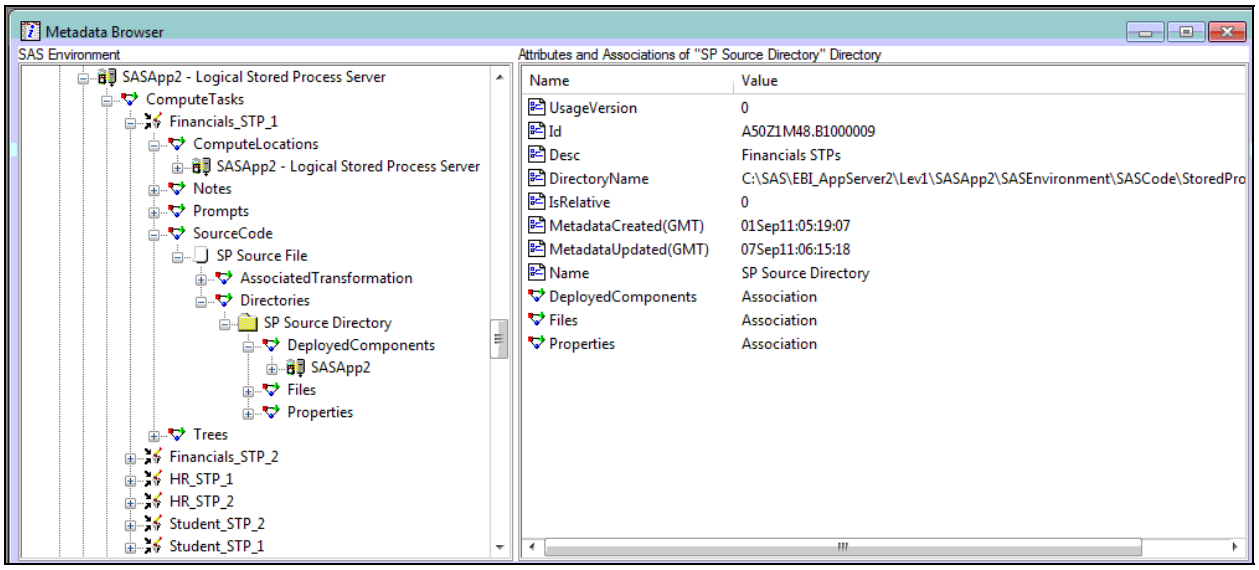
Figure 4. The Directory Data Set

Using the data contained in WORK.DIRECTORY, the following code can be submitted to build an UpdateMetadata request which changes the server association for the directories to SASApp2 and alters the physical path in the DirectoryName attribute to the appropriate path for the SASApp2 installation path:

```
filename InReq2 “C:\sasfiles\UpdateSTPDirectories.xml" lrecl=1024;
```

```
data _null_;
set Directory END=OVER;
/* Modify DirectoryName to point to ...\EBI_AppServer2\Lev1\SASApp2\... */
IF INDEX(DIRECTORYNAME,'EBI_AppServer1') gt 0 THEN DO;
  DirectoryName=TRANWRD(DirectoryName, 'EBI_AppServer1', 'EBI_AppServer2');
  DirectoryName=TRANWRD(DirectoryName, 'SASApp1', 'SASApp2');
END;

file InReq2;

if _n_=1 then put
'<Multiple_Requests>' /
  '<UpdateMetadata >' /
  '  <Metadata >';

put
  '<Directory Id="' Id '" ' 'Name="' Name '" ' 'DirectoryName="' DirectoryName '"
  >' /
  '      <DeployedComponents Function="Replace"> ' /
  '        <ServerContext ObjRef="A50Z1M48.AQ000003" Name="SASApp2" /> ' /
  '      </DeployedComponents >' /
  '    </Directory > ';

if Over then put
  '  </Metadata >' /
  '  <Reposid >$METAREPOSITORY</Reposid >'/
  '  <NS>SAS</NS>' /
  '  <!-- OMI_TRUSTED_CLIENT (268435456) OMI_RETURN_LIST (1024) -->'/
  '  <Flags>268436480</Flags>' /
  '  <Options /> '/
  '</UpdateMetadata >'
'</Multiple_Requests >'   ;
run;
```

After submitting the generated file, UpdateSTPDirectories.xml, via PROC METADATA, the source directories for the stored processes are now associated with the SASApp2 application server with an appropriate physical path defined (Figure 5). The process is now complete and the stored processes are ready for executing using the new application server.


Metadata Browser
SAS Environment
Attributes and Associations of "SP Source Directory" Directory

```
SASApp2 - Logical Stored Process Server
  ComputeTasks
    Financials_STP_1
      ComputeLocations
        SASApp2 - Logical Stored Process Server
      Notes
      Prompts
      SourceCode
        SP Source File
          AssociatedTransformation
          Directories
            SP Source Directory
              DeployedComponents
                SASApp2
              Files
              Properties
      Trees
    Financials_STP_2
    HR_STP_1
    HR_STP_2
    Student_STP_2
    Student_STP_1
```

Name	Value
UsageVersion	0
Id	A50Z1M48.B1000009
Desc	Financials STPs
DirectoryName	C:\SAS\EBI_AppServer2\Lev1\SASApp2\SASEnvironment\SASCode\StoredPro
IsRelative	0
MetadataCreated(GMT)	01Sep11:05:19:07
MetadataUpdated(GMT)	07Sep11:06:15:18
Name	SP Source Directory
DeployedComponents	Association
Files	Association
Properties	Association

Figure 5. Metadata Browser view of Stored Processes associated with SASApp2 - Logical Stored Process Server with correct "SP Source Directory" locations.

Data Elements Dictionary (DED) – A Practical Example

Our data warehouse is comprised of elements from various enterprise systems across campus. Primarily, the data is built from PeopleSoft/Oracle tables and documentation is sparse. Table/ field definitions and relationships between source data and data warehouse tables can be tedious to document and difficult to maintain. Our goal is to have Business Analysts/Subject Matter Experts provide business logic and definitions for source tables and fields while developers add metadata as they create new target objects. Using the SAS Open Metadata Interface we extract and combine this metadata to create a robust data dictionary.

Step 1 - Create XML File:

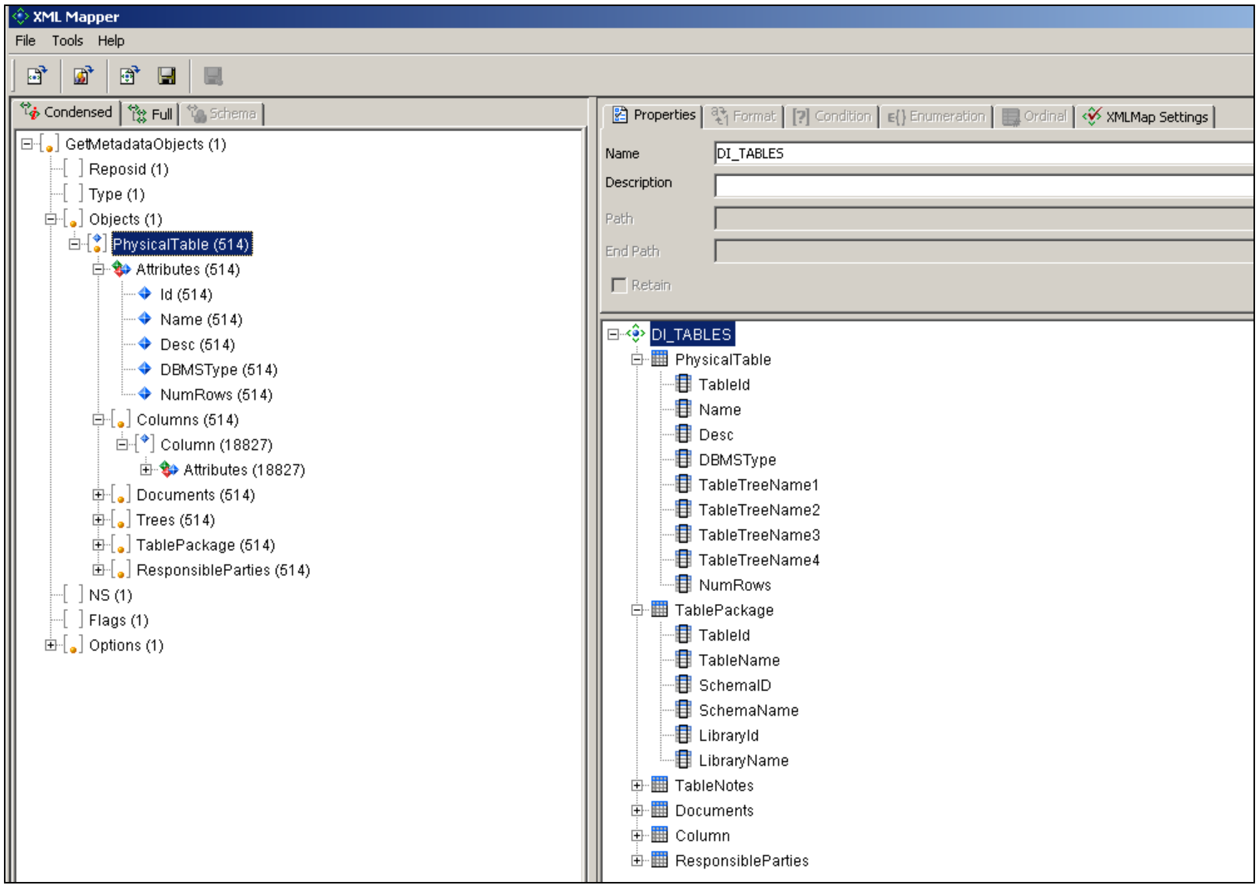
The following code creates an XML file that queries the metadata using the GetMetadataObjects method. In this step, we are returning all metadata objects with a type of PhysicalTable. We then further specify the associations we would like to see returned in the XML hierarchy by identifying the templates to use. These include: Columns, Documents, Trees, TablePackage, and ResponsibleParties.

```
FILENAME TBLREQ 'U:\ETL 9.2\YDH_TABLEQRY.QXML';
FILENAME TBLRESP "U:\ETL 9.2\DITABLES_&SYSDATE..XML" LRECL=1024;

DATA _NULL_;
FILE TBLREQ;
INFILE CARDS4;
LENGTH LONG $256;
INPUT;
LONG=_INFILE_;
PUT LONG ' ';
CARDS4;
<GETMETADATAOBJECTS>
  <REPOSID>$METAREPOSITORY</REPOSID>
  <TYPE>PHYSICALTABLE</TYPE>
  <OBJECTS/>
  <NS>SAS</NS>
  <FLAGS>388</FLAGS><!-OMI_XMLSELECT(128)+OMI_GET_METADATA(256)+OMI_TEMPLATE(4) -->
  <OPTIONS>
     <TEMPLATES>
      <PHYSICALTABLE ID="" COLUMNNAME="" DESC="" DBMSTYPE="" NUMROWS="">
          <COLUMNS/>
          <DOCUMENTS/>
          <TREES/>
          <TABLEPACKAGE/>
          <RESPONSIBLEPARTIES/>
     </PHYSICALTABLE>
          <COLUMN ID="" NAME="" COLUMNLENGTH="" SASFORMAT=""SASINFORMAT=""/>
          <DOCUMENT ID = "" NAME="" >
             <NOTES/>
          </DOCUMENT>
             <NOTE NAME="">
               <TEXTSTORE/>
             </NOTE>
               <TEXTSTORE NAME="" STOREDTEXT="" />
          <TREE ID="" NAME="">
             <PARENTTREE ID="" NAME="" />
          </TREE>
          <DATABASESCHEMA ID="" NAME="" SCHEMANAME="">
             <USEDBYPACKAGES/>
          </DATABASESCHEMA>
          <SASLIBRARY ID="" NAME=""/>
          <RESPONSIBLEPARTY ID="" NAME="" ROLE="">
             <PERSONS/>
          </RESPONSIBLEPARTY>
             <PERSON ID="" DISPLAYNAME="" />
     </TEMPLATES>
   </OPTIONS>
</GETMETADATAOBJECTS>;;;;
RUN;
```

```
PROC METADATA IN=TBLREQ OUT=TBLRESP;
RUN;
```

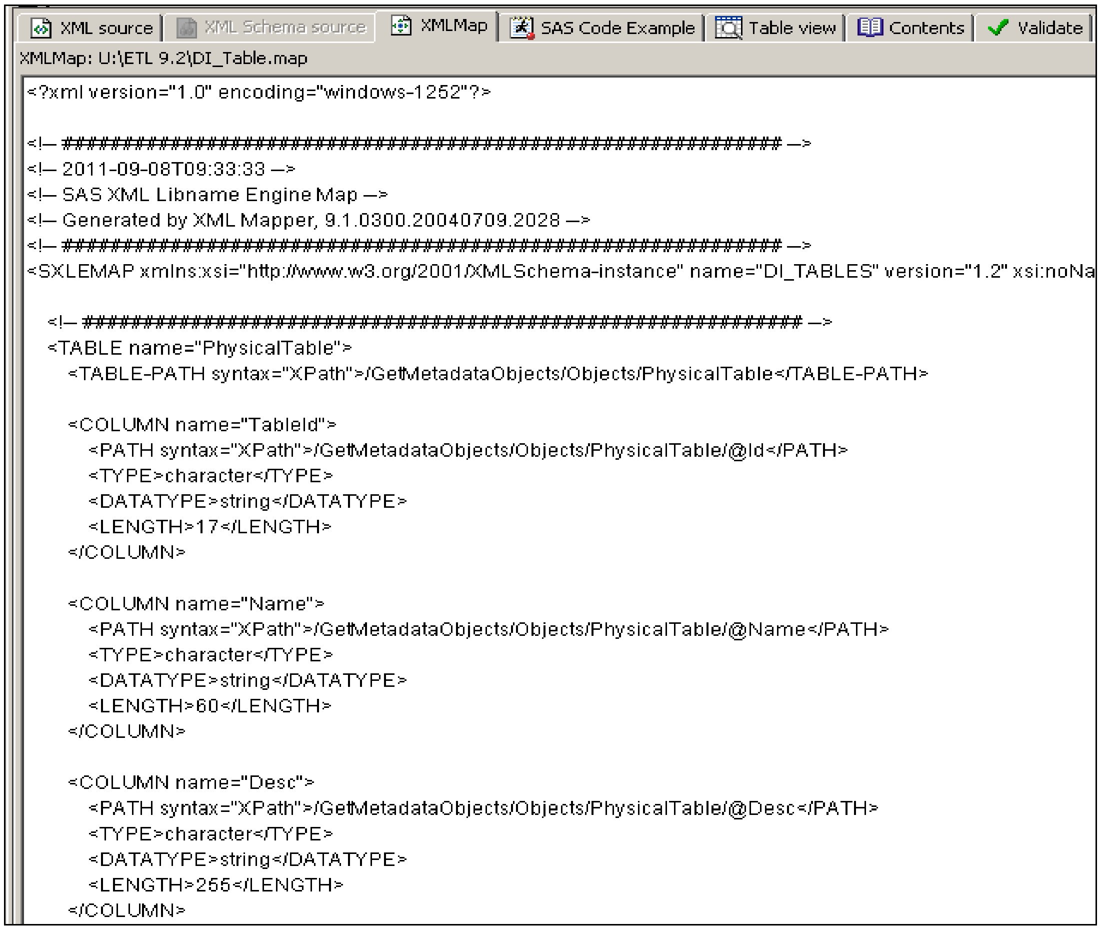
Step 2 - Build XML Map:

Using SAS XMLMapper, we open the XML file created in the previous step and create a map to read the results into SAS datasets. Navigating through the XML hierarchy is straightforward and this tool allows us to browse the object then drag and drop to create custom data sets. Figure 6.a below illustrates how we’ve mapped the PhysicalTable metadata object along with its related associations and attributes into our core tables. Figure 6.b shows the resulting XMLMap.



Name DI_TABLES

Figure 6.a: A custom map “DI_TABLES” is created that will create in the following tables: PhysicalTable, TablePackage, TableNotes, Documents, Column and ResponsibleParties.



XMLMap: U:\ETL 9.2\DI_Table.map

```
<?xml version="1.0" encoding="windows-1252"?>

<!-- ############################################################ -->
<!-- 2011-09-08T09:33:33 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.1.0300.20040709.2028 -->
<!-- ############################################################ -->
<SXLEMAP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="DI_TABLES" version="1.2" xsi:noNa

  <!-- ############################################################ -->
  <TABLE name="PhysicalTable">
    <TABLE-PATH syntax="XPath">/GetMetadataObjects/Objects/PhysicalTable</TABLE-PATH>

    <COLUMN name="TableId">
      <PATH syntax="XPath">/GetMetadataObjects/Objects/PhysicalTable/@Id</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>17</LENGTH>
    </COLUMN>

    <COLUMN name="Name">
      <PATH syntax="XPath">/GetMetadataObjects/Objects/PhysicalTable/@Name</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>60</LENGTH>
    </COLUMN>

    <COLUMN name="Desc">
      <PATH syntax="XPath">/GetMetadataObjects/Objects/PhysicalTable/@Desc</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>255</LENGTH>
    </COLUMN>
```

Figure 6.b: A portion of the XMLMap generated by SAS XMLMapper

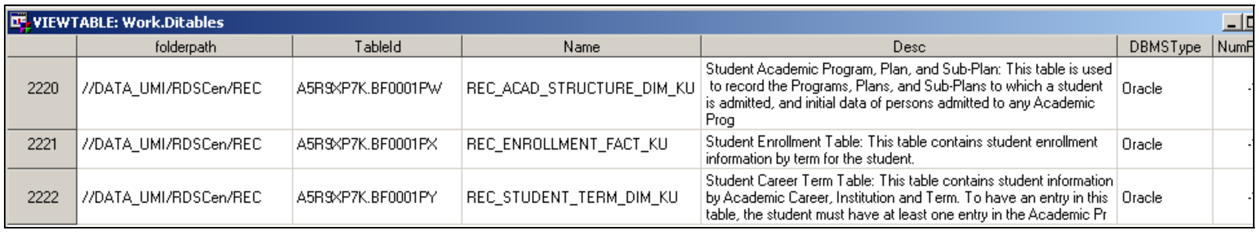
We then bring the XML data into SAS datasets using the XML engine in a libname statement with the following:

```
FILENAME DI_TBL 'U:\ETL 9.2\DI_TABLE.MAP';
LIBNAME  TBLRESP XML XMLMAP=DI_TBL ACCESS=READONLY;
```

Step 3 - Create DED Tables:

The following data step creates one of the core dictionary tables.

```
DATA DITABLES;
  LENGTH FOLDERPATH $200;
  SET TBLRESP.PHYSICALTABLE;
  FOLDERPATH=CATX("/",TABLETREENAME1,TABLETREENAME2,TABLETREENAME3,TABLETREENAME4);
  DROP TABLETREENAME:;
RUN;
```


VIEWTABLE: Work.Ditables

	folderpath	TableId	Name	Desc	DBMSType	NumF
2220	//DATA_UMI/RDSCen/REC	A5R9XP7K.BF0001PW	REC_ACAD_STRUCTURE_DIM_KU	Student Academic Program, Plan, and Sub-Plan: This table is used to record the Programs, Plans, and Sub-Plans to which a student is admitted, and initial data of persons admitted to any Academic Prog	Oracle	-
2221	//DATA_UMI/RDSCen/REC	A5R9XP7K.BF0001PX	REC_ENROLLMENT_FACT_KU	Student Enrollment Table: This table contains student enrollment information by term for the student.	Oracle	-
2222	//DATA_UMI/RDSCen/REC	A5R9XP7K.BF0001PY	REC_STUDENT_TERM_DIM_KU	Student Career Term Table: This table contains student information by Academic Career, Institution and Term. To have an entry in this table, the student must have at least one entry in the Academic Pr	Oracle	-

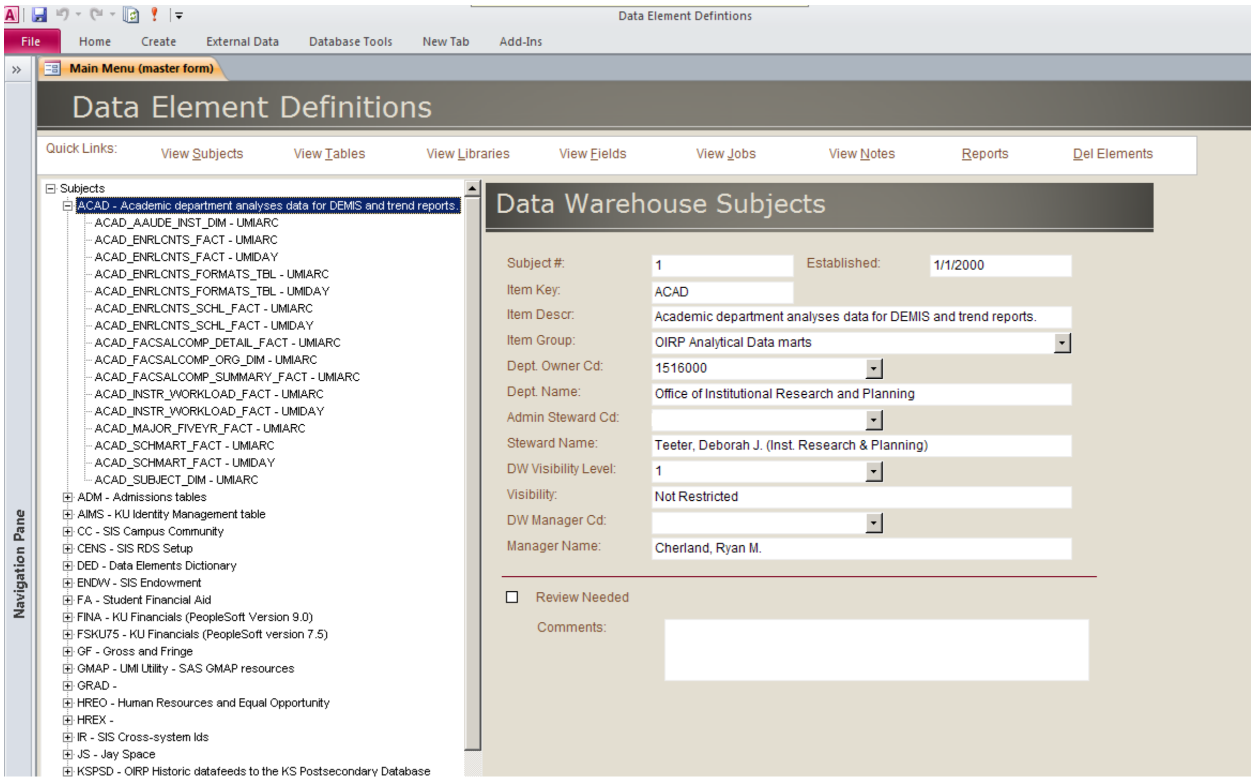
Figure 7: A sample of PhysicalTable metadata

Each step above is repeated for the core sections of our data dictionary and run routinely to keep the dictionary current.

Putting It All Together

Using Microsoft Access as a user interface, the Data Elements Dictionary (or DED) becomes an interactive tool for both end users and developers.

Tables are organized into subject areas that have their own data steward and data warehouse manager (developer).



Subjects
- ACAD - Academic department analyses data for DEMIS and trend reports.
  - ACAD_AAUDE_INST_DIM - UMIARC
  - ACAD_ENRLCNTS_FACT - UMIARC
  - ACAD_ENRLCNTS_FACT - UMIDAY
  - ACAD_ENRLCNTS_FORMATS_TBL - UMIARC
  - ACAD_ENRLCNTS_FORMATS_TBL - UMIDAY
  - ACAD_ENRLCNTS_SCHL_FACT - UMIARC
  - ACAD_ENRLCNTS_SCHL_FACT - UMIDAY
  - ACAD_FACSALCOMP_DETAIL_FACT - UMIARC
  - ACAD_FACSALCOMP_ORG_DIM - UMIARC
  - ACAD_FACSALCOMP_SUMMARY_FACT - UMIARC
  - ACAD_INSTR_WORKLOAD_FACT - UMIARC
  - ACAD_INSTR_WORKLOAD_FACT - UMIDAY
  - ACAD_MAJOR_FIVEYR_FACT - UMIARC
  - ACAD_SCHMART_FACT - UMIARC
  - ACAD_SCHMART_FACT - UMIDAY
  - ACAD_SUBJECT_DIM - UMIARC
- ADM - Admissions tables
- AIMS - KU Identity Management table
- CC - SIS Campus Community
- CENS - SIS RDS Setup
- DED - Data Elements Dictionary
- ENDW - SIS Endowment
- FA - Student Financial Aid
- FINA - KU Financials (PeopleSoft Version 9.0)
- FSKU75 - KU Financials (PeopleSoft version 7.5)
- GF - Gross and Fringe
- GMAP - UMI Utility - SAS GMAP resources
- GRAD -
- HREO - Human Resources and Equal Opportunity
- HREX -
- IR - SIS Cross-system Ids
- JS - Jay Space
- KSPSD - OIRP Historic datafeeds to the KS Postsecondary Database

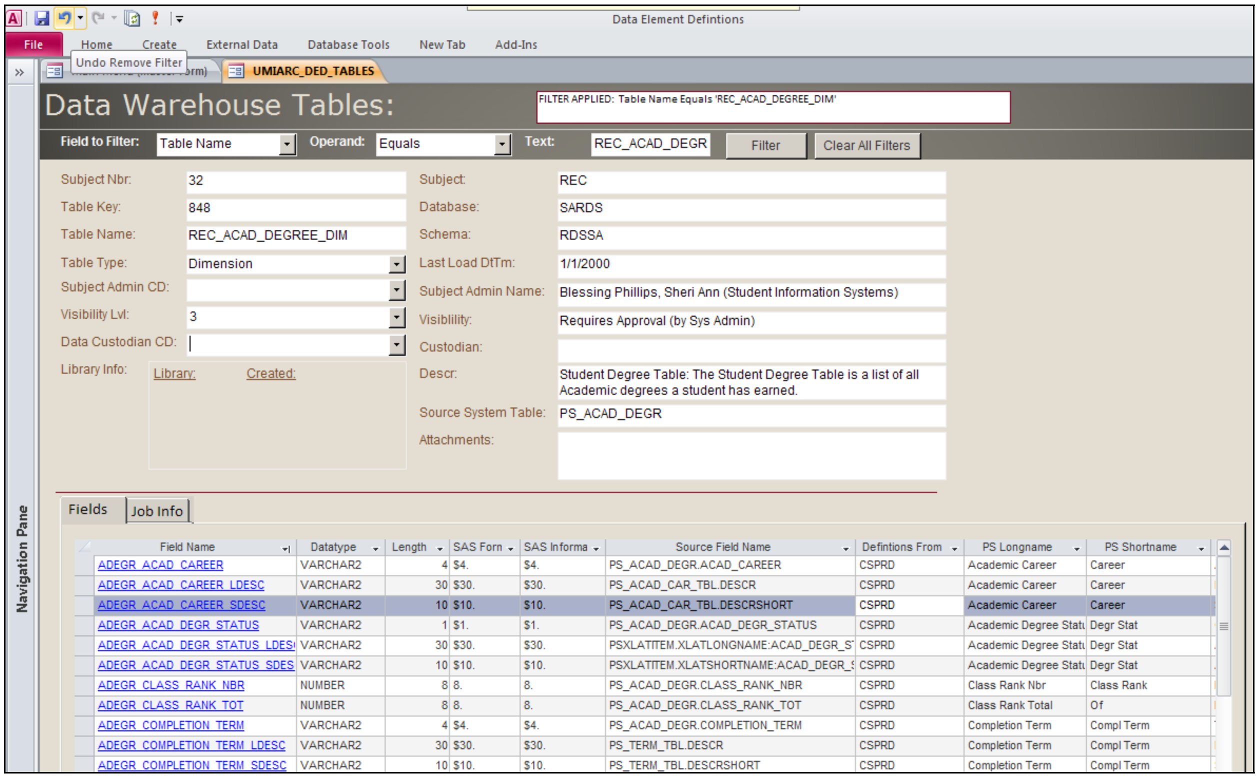
Subject #:	1	Established:	1/1/2000
Item Key:	ACAD		
Item Descr:	Academic department analyses data for DEMIS and trend reports.		
Item Group:	OIRP Analytical Data marts		
Dept. Owner Cd:	1516000		
Dept. Name:	Office of Institutional Research and Planning		
Admin Steward Cd:			
Steward Name:	Teeter, Deborah J. (Inst. Research & Planning)		
DW Visibility Level:	1		
Visibility:	Not Restricted		
DW Manager Cd:			
Manager Name:	Cherland, Ryan M.		

☐ Review Needed

Comments:

Figure 8: Data warehouse subject areas

For each target table, we maintain description, data steward, fields, table type, data library, and schema information. Each column (field) is displayed along with its source information and description. We also are able to link into our 3rd party job scheduling tables to display information about the job that created and/or updated the table.



Data Warehouse Tables:

FILTER APPLIED: Table Name Equals 'REC_ACAD_DEGREE_DIM'

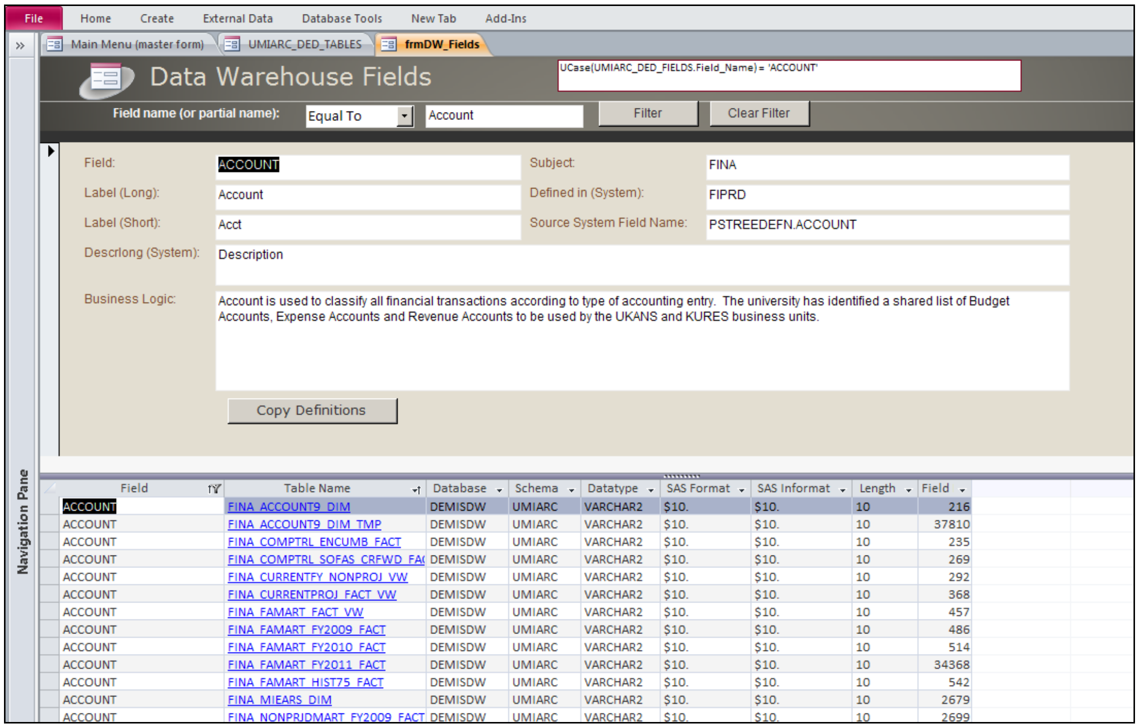
Field to Filter: Table Name | Operand: Equals | Text: REC_ACAD_DEGR | Filter | Clear All Filters

Subject Nbr:	32	Subject:	REC
Table Key:	848	Database:	SARDS
Table Name:	REC_ACAD_DEGREE_DIM	Schema:	RDSSA
Table Type:	Dimension	Last Load DtTm:	1/1/2000
Subject Admin CD:		Subject Admin Name:	Blessing Phillips, Sheri Ann (Student Information Systems)
Visibility Lvl:	3	Visiblility:	Requires Approval (by Sys Admin)
Data Custodian CD:		Custodian:	
Library Info:	Library: Created:	Descr:	Student Degree Table: The Student Degree Table is a list of all Academic degrees a student has earned.
		Source System Table:	PS_ACAD_DEGR
		Attachments:	

Field Name	Datatype	Length	SAS Forn	SAS Informa	Source Field Name	Defintions From	PS Longname	PS Shortname
ADEGR_ACAD_CAREER	VARCHAR2	4	$4.	$4.	PS_ACAD_DEGR.ACAD_CAREER	CSPRD	Academic Career	Career
ADEGR_ACAD_CAREER_LDESC	VARCHAR2	30	$30.	$30.	PS_ACAD_CAR_TBL.DESCR	CSPRD	Academic Career	Career
ADEGR_ACAD_CAREER_SDESC	VARCHAR2	10	$10.	$10.	PS_ACAD_CAR_TBL.DESCRSHORT	CSPRD	Academic Career	Career
ADEGR_ACAD_DEGR_STATUS	VARCHAR2	1	$1.	$1.	PS_ACAD_DEGR.ACAD_DEGR_STATUS	CSPRD	Academic Degree Statu	Degr Stat
ADEGR_ACAD_DEGR_STATUS_LDES	VARCHAR2	30	$30.	$30.	PSXLATITEM.XLATLONGNAME:ACAD_DEGR_ST	CSPRD	Academic Degree Statu	Degr Stat
ADEGR_ACAD_DEGR_STATUS_SDES	VARCHAR2	10	$10.	$10.	PSXLATITEM.XLATSHORTNAME:ACAD_DEGR_S	CSPRD	Academic Degree Statu	Degr Stat
ADEGR_CLASS_RANK_NBR	NUMBER	8	8.	8.	PS_ACAD_DEGR.CLASS_RANK_NBR	CSPRD	Class Rank Nbr	Class Rank
ADEGR_CLASS_RANK_TOT	NUMBER	8	8.	8.	PS_ACAD_DEGR.CLASS_RANK_TOT	CSPRD	Class Rank Total	Of
ADEGR_COMPLETION_TERM	VARCHAR2	4	$4.	$4.	PS_ACAD_DEGR.COMPLETION_TERM	CSPRD	Completion Term	Compl Term
ADEGR_COMPLETION_TERM_LDESC	VARCHAR2	30	$30.	$30.	PS_TERM_TBL.DESCR	CSPRD	Completion Term	Compl Term
ADEGR_COMPLETION_TERM_SDESC	VARCHAR2	10	$10.	$10.	PS_TERM_TBL.DESCRSHORT	CSPRD	Completion Term	Compl Term

Figure 9: Table data elements defined

At the field level, we maintain the business logic (or definition), data type and format, and the source (tablename.fieldname). We can quickly determine what the impact will be if the structure of a single field is changed in an underlying source system. An extremely useful feature of the DED is that end users are also able to mass update the business logic for fields that appear in multiple tables.



Data Warehouse Fields

UCase(UMIARC_DED_FIELDS.Field_Name) = 'ACCOUNT'

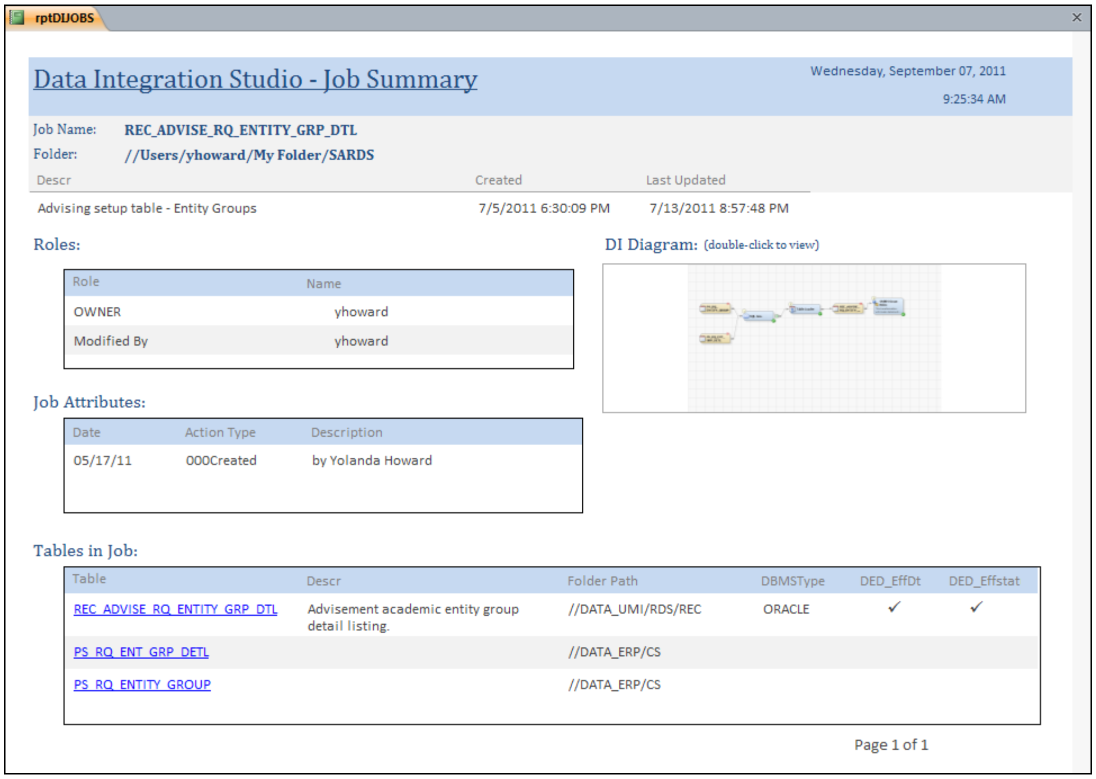
Field name (or partial name): Equal To | Account | Filter | Clear Filter

Field:	ACCOUNT	Subject:	FINA
Label (Long):	Account	Defined in (System):	FIPRD
Label (Short):	Acct	Source System Field Name:	PSTREEDEFN.ACCOUNT
Descrlong (System):	Description		
Business Logic:	Account is used to classify all financial transactions according to type of accounting entry. The university has identified a shared list of Budget Accounts, Expense Accounts and Revenue Accounts to be used by the UKANS and KURES business units.		

Field	Table Name	Database	Schema	Datatype	SAS Format	SAS Informat	Length	Field
ACCOUNT	FINA_ACCOUNT9_DIM	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	216
ACCOUNT	FINA_ACCOUNT9_DIM_TMP	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	37810
ACCOUNT	FINA_COMPTRL_ENCUMB_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	235
ACCOUNT	FINA_COMPTRL_SOFAS_CRFWD_FA(	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	269
ACCOUNT	FINA_CURRENTFY_NONPROJ_VW	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	292
ACCOUNT	FINA_CURRENTPROJ_FACT_VW	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	368
ACCOUNT	FINA_FAMART_FACT_VW	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	457
ACCOUNT	FINA_FAMART_FY2009_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	486
ACCOUNT	FINA_FAMART_FY2010_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	514
ACCOUNT	FINA_FAMART_FY2011_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	34368
ACCOUNT	FINA_FAMART_HIST75_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	542
ACCOUNT	FINA_MIEARS_DIM	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	2679
ACCOUNT	FINA_NONPRJDMART_FY2009_FACT	DEMISDW	UMIARC	VARCHAR2	$10.	$10.	10	2699

Figure 10: Field definitions

In Data Integration Studio 4.21, developers add standardized notes and attributes to each table and job they create. We extract that metadata and are able to produce a report of each job. Another very useful piece of information is the Process Flow Diagram created in DI Studio. The diagram can can be saved as an image file and then brought into an Access table with an attachement field. The image can then be accessed in an Access report. This allows end users to quickly see how a particiular table in the data warehouse was created.


rptDIJOBS

Data Integration Studio - Job Summary

Wednesday, September 07, 2011
9:25:34 AM

Job Name: REC_ADVISE_RQ_ENTITY_GRP_DTL
Folder: //Users/yhoward/My Folder/SARDS

Descr	Created	Last Updated
Advising setup table - Entity Groups	7/5/2011 6:30:09 PM	7/13/2011 8:57:48 PM

Roles:

Role	Name
OWNER	yhoward
Modified By	yhoward

DI Diagram: (double-click to view)

Job Attributes:

Date	Action Type	Description
05/17/11	000Created	by Yolanda Howard

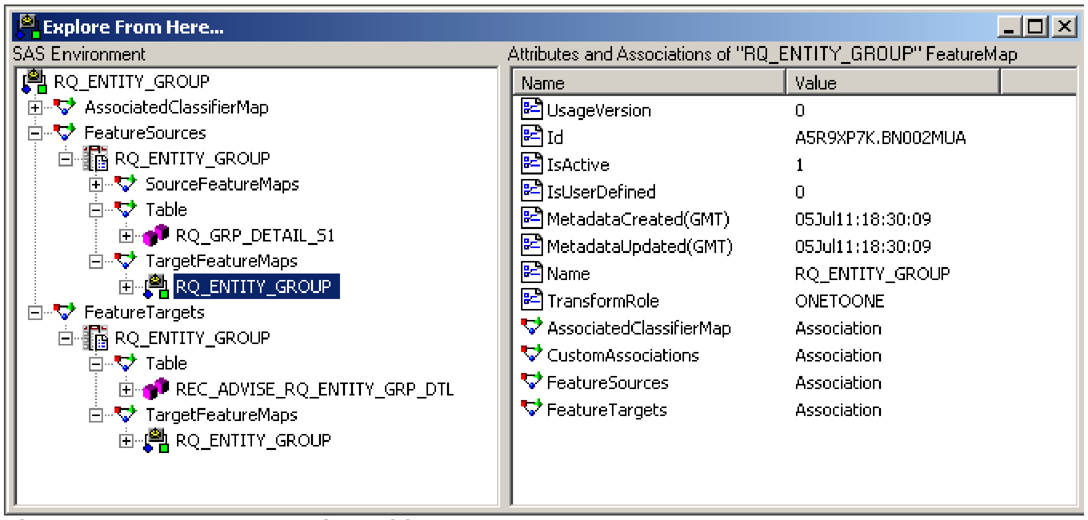
Tables in Job:

Table	Descr	Folder Path	DBMSType	DED_EffDt	DED_Effstat
REC_ADVISE_RQ_ENTITY_GRP_DTL	Advisement academic entity group detail listing.	//DATA_UMI/RDS/REC	ORACLE	✓	✓
PS_RQ_ENT_GRP_DETL		//DATA_ERP/CS			
PS_RQ_ENTITY_GROUP		//DATA_ERP/CS			

Page 1 of 1

Figure 11: Data Integration Studio Job Summary Report

Using the FeatureMap metadata object we are able to document the job flow and identify the source/target tables. The FeatureMap outlines each source (FeatureSource) and target (FeatureTarget).


Explore From Here...

SAS Environment

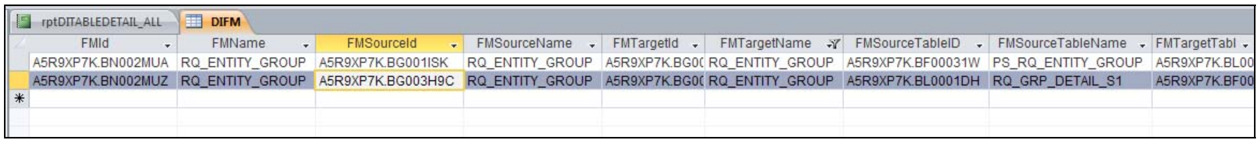
```
RQ_ENTITY_GROUP
 + AssociatedClassifierMap
 - FeatureSources
    - RQ_ENTITY_GROUP
       + SourceFeatureMaps
       - Table
          + RQ_GRP_DETAIL_S1
       - TargetFeatureMaps
          + RQ_ENTITY_GROUP
 - FeatureTargets
    - RQ_ENTITY_GROUP
       - Table
          + REC_ADVISE_RQ_ENTITY_GRP_DTL
       - TargetFeatureMaps
          + RQ_ENTITY_GROUP
```

Attributes and Associations of "RQ_ENTITY_GROUP" FeatureMap

Name	Value
UsageVersion	0
Id	A5R9XP7K.BN002MUA
IsActive	1
IsUserDefined	0
MetadataCreated(GMT)	05Jul11:18:30:09
MetadataUpdated(GMT)	05Jul11:18:30:09
Name	RQ_ENTITY_GROUP
TransformRole	ONETOONE
AssociatedClassifierMap	Association
CustomAssociations	Association
FeatureSources	Association
FeatureTargets	Association

Figure 12: FeatureMap Metdata Object

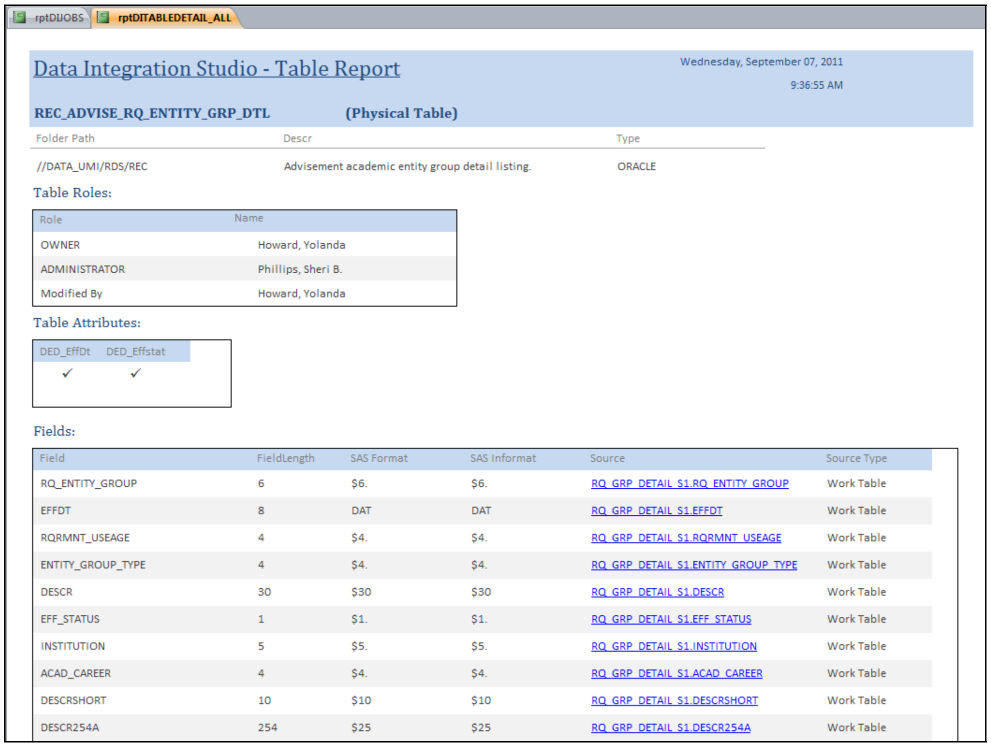
Using the Metadata URI as a key in our DED allows us to map a single column to its FeatureMap and then we can determine the source. In many cases there is a work table in between so we extract the WorkTable type objects as well. This allows us to drill through to the primary source. Figure 13 below show the final source to target mapping table using the FeatureMaps.



rptDITABLEDETAIL_ALL | DIFM

FMId	FMName	FMSourceId	FMSourceName	FMTargetId	FMTargetName	FMSourceTableID	FMSourceTableName	FMTargetTabl
A5R9XP7K.BN002MUA	RQ_ENTITY_GROUP	A5R9XP7K.BG001ISK	RQ_ENTITY_GROUP	A5R9XP7K.BG0(	RQ_ENTITY_GROUP	A5R9XP7K.BF00031W	PS_RQ_ENTITY_GROUP	A5R9XP7K.BL00
A5R9XP7K.BN002MUZ	RQ_ENTITY_GROUP	A5R9XP7K.BG003H9C	RQ_ENTITY_GROUP	A5R9XP7K.BG0(	RQ_ENTITY_GROUP	A5R9XP7K.BL0001DH	RQ_GRP_DETAIL_S1	A5R9XP7K.BF00

Figure 13: Mapping of Source/Target Tables



rptDIJOBS | rptDITABLEDETAIL_ALL

Data Integration Studio - Table Report

Wednesday, September 07, 2011

9:36:55 AM

REC_ADVISE_RQ_ENTITY_GRP_DTL (Physical Table)

Folder Path	Descr	Type
//DATA_UMI/RDS/REC	Advisement academic entity group detail listing.	ORACLE

Table Roles:

Role	Name
OWNER	Howard, Yolanda
ADMINISTRATOR	Phillips, Sheri B.
Modified By	Howard, Yolanda

Table Attributes:

DED_EffDt	DED_Effstat
✓	✓

Fields:

Field	FieldLength	SAS Format	SAS Informat	Source	Source Type
RQ_ENTITY_GROUP	6	$6.	$6.	RQ_GRP_DETAIL_S1.RQ_ENTITY_GROUP	Work Table
EFFDT	8	DAT	DAT	RQ_GRP_DETAIL_S1.EFFDT	Work Table
RQRMNT_USEAGE	4	$4.	$4.	RQ_GRP_DETAIL_S1.RQRMNT_USEAGE	Work Table
ENTITY_GROUP_TYPE	4	$4.	$4.	RQ_GRP_DETAIL_S1.ENTITY_GROUP_TYPE	Work Table
DESCR	30	$30	$30	RQ_GRP_DETAIL_S1.DESCR	Work Table
EFF_STATUS	1	$1.	$1.	RQ_GRP_DETAIL_S1.EFF_STATUS	Work Table
INSTITUTION	5	$5.	$5.	RQ_GRP_DETAIL_S1.INSTITUTION	Work Table
ACAD_CAREER	4	$4.	$4.	RQ_GRP_DETAIL_S1.ACAD_CAREER	Work Table
DESCRSHORT	10	$10	$10	RQ_GRP_DETAIL_S1.DESCRSHORT	Work Table
DESCR254A	254	$25	$25	RQ_GRP_DETAIL_S1.DESCR254A	Work Table

Figure 14: Data Integration Studio Detailed Table Report

Keeping the Metadata in Sync

The DED tables in Microsoft Access are linked to Oracle tables in our warehouse. As end users update the business logic in the DED, the Oracles tables are directly updated. This information is then pushed back into the metadata using XML and the UpdateMetadata method. This keeps the DED and the SAS metadata in sync.

We use PROC SQL to determine which metadata object definitions do not match and create a work table with the object URI (in the following example, this is the field TABLEID):

```
PROC SQL;
  CREATE TABLE MISSING_DESC AS (
  SELECT A.FOLDERPATH, A.TABLEID, A.NAME, A.DESC, A.SCHEMANAME, B.TABLE_DESCR,
HTMLENCODE(B.TABLE_DESCR,'AMP LT GT APOS QUOT') AS NEWDESCR
  FROM DITABLES_NEW A, DED_TABLES B
  WHERE (A.TABLEID = B. TABLEID AND(A.DESC <> B.TABLE_DESCR));
QUIT;
```

Note: We use HTMLENCODE to return the encoded string in order to handle special characters correctly in the XML.

The work table is then used in the following code to process the metadata update:

```
FILENAME TDESREQ 'U:\ETL 9.2\YDH_TABLEDESCR.QXML';
FILENAME TDESRESP "U:\ETL 9.2\TABLEDESCR&SYSDATE..XML" LRECL=1024;

DATA _NULL_;
SET MISSING_DESC END=OVER;
FILE TDESREQ;
```

```
IF _N_=1 THEN PUT
 '<MULTIPLE_REQUESTS>' /
 '<UPDATEMETADATA >' /
 '  <METADATA >';
PUT
 '    <PHYSICALTABLE ID="' TABLEID '" ' /
 '      DESC ="' NEWDESCR '" />' /;
IF OVER THEN PUT
 '  </METADATA >' /
 '  <REPOSID >$METAREPOSITORY</REPOSID >'/
 '  <NS>SAS</NS>' /
 '  <!-- OMI_TRUSTED_CLIENT (268435456) OMI_RETURN_LIST (1024) -->'/
 '  <FLAGS>268436480</FLAGS>' /
 '  <OPTIONS /> '/
 '</UPDATEMETADATA >'
'</MULTIPLE_REQUESTS >'
  ;
RUN;

PROC METADATA IN=TDESREQ OUT=TDESRESP;
RUN;
```

Conclusion

Exploring the SAS Open Metadata Architecture can be daunting but it certainly provides an abundance of useful information. Using tools like the SAS Metadata Browser, SAS XML Mapper and PROC Metadata to both read and write the metadata allows us to fully take advantage of this information. These tools will be critical as we delve further into the metadata and continue to explore ways to automate and document our work.

References

SAS 9.2 Open Metadata Interface Reference and Usage. SAS Institute Inc. 2009.

SAS Institute Inc. 2009, *SAS 9.2 Metadata Model: Reference*
http://support.sas.com/documentation/cdl/en/omamodref/61849/HTML/default/viewer.htm#titlepage.htm

Recommended Reading

SAS Institute Inc. 2010, *SAS 9.2 XML LIBNAME Engine: User's Guide, Second Edition*. Cary, NC: SAS Institute Inc.

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

Contact Information

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

Ray Helm
The University of Kansas
Office of Institutional Research and Planning
1246 W. Campus Rd., Room 339
(785) 864-4412
ray-helm@ku.edu

Yolanda Howard
The University of Kansas
Office of Institutional Research and Planning
1246 W. Campus Rd, Room 339
(785) 864-4412
yolanda-howard@ku.edu