PharmaSUG2010 - Paper DM01

Editing SAS Metadata – Automated From CSV Files Using XML String in SAS Data Integration Studio

Annie Guo, ICON Clinical Research, San Francisco, CA

ABSTRACT

The SAS Metadata Server® introduces a new world to clinical data programmers. It is a storage centre to store information about every single object there is in the SAS System®. Not only the table level and column level attributes such as SAS table labels and column lengths, extended attributes are also built in and extensible to keep valuable business data in this centralized location. To access the metadata, SAS Data Integration Studio® or SAS Management Console® provides a GUI environment. However, they both lack a single interface to manage all the related metadata for a table including the extended attributes.

This paper presents an alternative approach. That is to maintain the metadata in text-based CSV files, dynamically generate XML strings with Data _Null_ based on the contents in the CSV files, and update, add or delete the metadata in SAS with the XML string. The entire process is modularized and incorporated into a SAS DI job as the front end to enable automation.

This approach gains efficiency especially when there are a lot of changes to make in the metadata. The technique provides an automated mechanism to ensure data quality as opposed to manual editing via SAS DI Studio.

INTRODUCTION

There are two basic types of metadata concerning SDTM formatted databases in the SAS Metadata Server. The first is the traditional data level and column level attributes such as SAS table names, column names, and column lengths. Ultimately they construct the physical SAS tables.

The other basic type of metadata is referred as extended attributes. They are not needed when building the physical SAS tables, but can contain important business information, and in the case of SDTM, are required for electronic data submission. Some of the extended attributes are predefined and rarely changed, for example, the Class and Purpose of SDTM domains. Others may change depending on the study, for example, the Structure and Keys of SDTM domains, and the Origin, Length, and Controlled Terminology of SDTM columns.

Though managing the metadata can be done using DI Studio or Management Console, either tool lacks a single interface to manage all the related metadata for a table including the attributes and extended attributes. In addition, both tools require manual editing and point-and-clicks for each attribute or extended attribute, per table or column, so the process can be tedious and time consuming. To make the matter worse, as a CRO many sponsor studies are run simultaneously, and each may have its own study-specific data specifications, no matter how slightly different. So it takes even greater effort to efficiently maintain the quality of the metadata for all studies.

As opposed to the manual editing via DI Studio or Management Console, SAS Open Metadata Interface is an application programming interface that uses XML as its transport language to perform metadata management tasks. So the idea behind the algorithm presented in this paper is if we can put together a file consisting of all the attributes and extended attributes that are to be written to the SAS Metadata Server, the creation of XML can be automated following its syntax, and then it takes just a SAS procedure to pass the XML string to the metadata server to complete the task. This programmatic approach will be efficient and ensure the quality of the metadata eliminating the error prone manual process.

DATA SPECIFICATIONS IN CSV FILES

Study data specifications are commonly documented in MS Excel® files. However, it is decided to use text-based CSV files to interact with SAS, so the XLS files are saved as CSV files. There will be two CSV files, one for table specifications, and the other for column specifications. Figures 1 and 2 are sample CSV files for one of the SDTM domains, the QS table, and for the QSCAT column in the QS table, respectively. In both figures the headings in green indicate extended attributes.

Domain	Labels	Class	Purpose	Structure	Keys
QS	Questionnaires	Findings	Tabulation	One record per question per time point per visit per subject	STUDYID, USUBJID, QSCAT, QSTESTCD, QSDTC, VISITNUM

Figure 1. QS table specifications in CSV file, extended attributes in green color

Variable	Label	Type	Length	Origin	Core	Role	Controlled Terminology
QSCAT	Category of Question	Char	40	CRF	Req	Grouping Qualifier	EASTERN COOPERATIVE ONCOLOGY GROUP

Figure 2. QSCAT column specifications in CSV file, extended attributes in green color

INITIALIZING A NEW STUDY REPOSITORY

To use the metadata server, first of all a new repository is built for the study. This can be done via SAS Management Console. Then the Register SDTM Domains plug-in, shown in Figure 3, that comes with SAS Clinical DI is used to register tables as SDTM domains for the study. The way the plug-in works is that there is the default standard repository containing the published SDTM domains, and the plug-in copies the metadata from the default repository to the new study repository. Consider the QS table as an example. At this point there is only the metadata, inherited from the default repository, which may not all be the same as the study specific data specifications.

Figure 3: Register SDTM Domains plug-in in Clinical DI

UPDATING ATTRIBUTES

The table level and column level attributes are easy to update. This is because DI Studio features the Update Table Metadata plug-in, which runs Proc Metalib to synchronize the table level and column level attributes with their corresponding physical SAS tables. So first of all, the physical SAS tables are created in Base SAS programming according to the study data specifications. Next, in DI Studio, the Update Table Metadata plug-in is used, shown in Figure 4. In addition to the table and column attributes, this plug-in also updates the order of the columns in the metadata, dropping columns that do not exist in the physical table, and dropping or updating the index.

Note that with DI Studio version 3.4 there is a reported issue with this plug-in that does not synchronize the SAS tables label in the metadata.

Figure 4: Update Table Metadata plug-in in Clinical DI

Figure 5 shows the column level attributes in SAS metadata after the Update Table Metadata plug-in is run. They are consistent with the column label, type and length from the study column specifications (Figure 2).

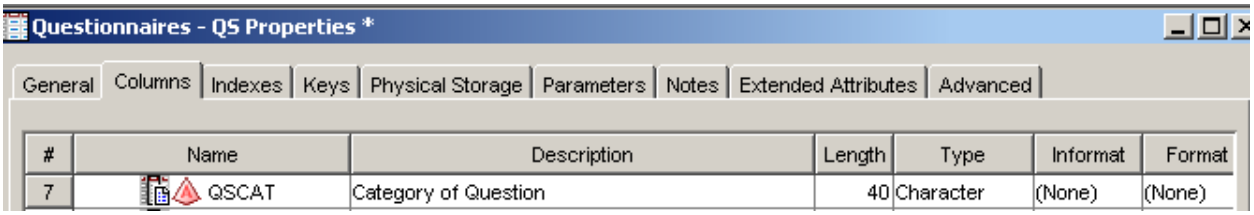


Figure 5. QSCAT column attributes consistent with data specs, after Update Table Metadata plug-in is run

EDITING EXTENDED ATTRIBUTES

Modifying the extended attributes, and also the table labels if using DI Studio version 3.4, is the focus of this paper. Before moving on, consider a snapshot of the current metadata, as shown in Figures 6, 7 and 8.

In Figure 6, the SAS table label "Questionnaires – QS" is different from "Questionnaires" in the study table specifications (Figure 1).

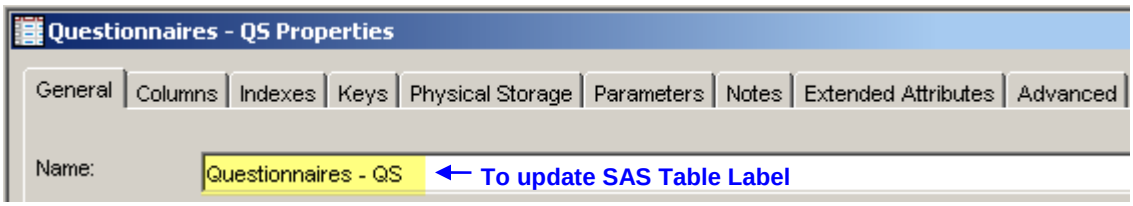


Figure 6. QS table label in SAS metadata before Metadata Editor is run

In Figure 7, the value on the extended attribute Structure is different from "One record per question per time point per visit per subject" in the study table specifications (Figure 1), and also the extended attribute Keys is not there.

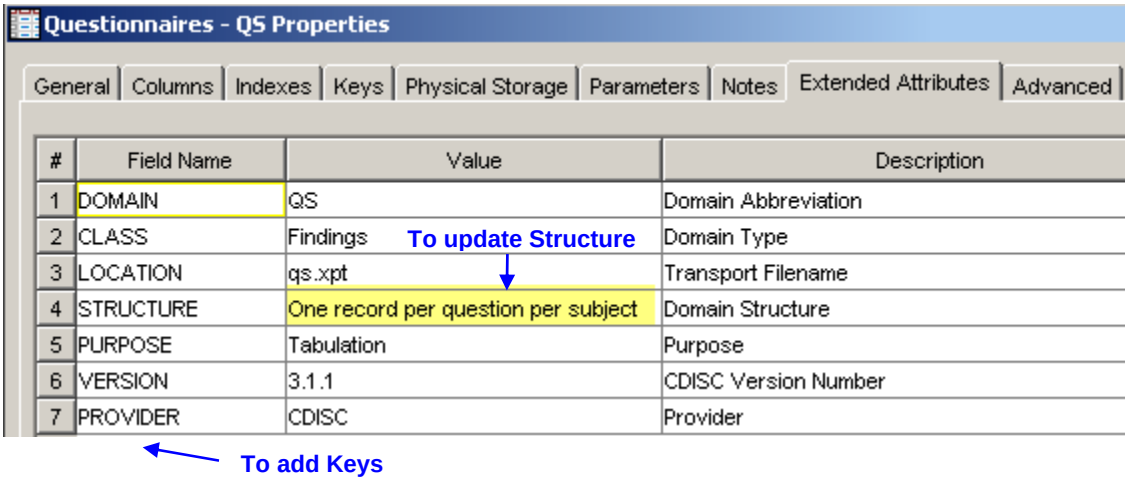
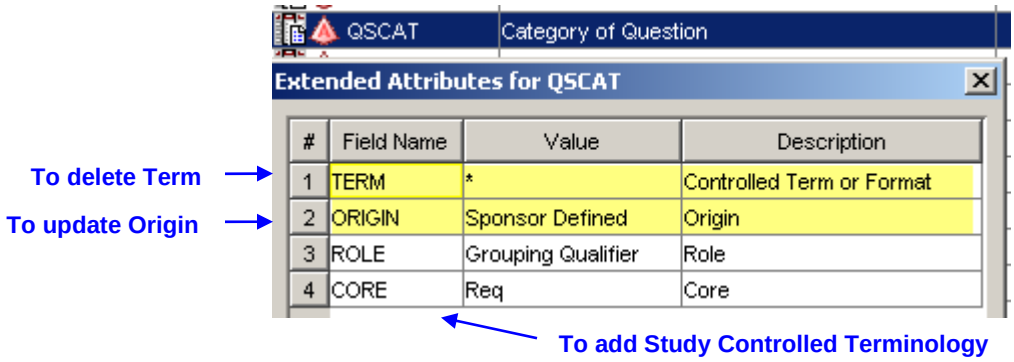


Figure 7. QS table extended attributes in SAS metadata before Metadata Editor is run

For column extended attributes shown in Figure 8, after comparing with the study column specifications (Figure 2), the Origin needs to be updated to "CRF", Term needs to be deleted, and the missing Study Controlled Terminology needs to be added.



#	Field Name	Value	Description
1	TERM	*	Controlled Term or Format
2	ORIGIN	Sponsor Defined	Origin
3	ROLE	Grouping Qualifier	Role
4	CORE	Req	Core

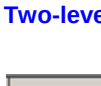
Figure 8. QSCAT column extended attributes in SAS metadata before Metadata Editor is run

METADATA EXTRACTOR

Prior to generating XML, there is the Metadata Extractor custom transformation to be run. To understand why it is placed here, we need some knowledge about the architecture of the SAS Metadata Server. The SAS Metadata Server is a relational database, where every piece of information is an object. Examples of objects are SAS tables, SAS columns, attributes and extended attributes. Each object has its unique, two-level instance identifier. In addition, the objects are related to one another in a parent-child structure, so there is the owning object associated with each object in the metadata. When performing metadata management tasks, we access them by their instance identifiers and if required by syntax, the instance identifier of their owning object. Since we are going to batch process the metadata, it is more efficient if the instance identifiers of all the extended attributes involved are gathered upfront, which in turn facilitates the creation of XML.

The Metadata Extractor custom transformation that comes with Clinical DI is used for this purpose. It utilizes an XML map and generates metadata requests to interact with the metadata server. The extracted metadata are saved in physical SAS data sets as in Figures 9, 10, 11 and 12. Note that the extracted metadata are consistent with what we have seen in Figures 6, 7 and 8.

In Figure 9, the instance identifier of the QS table is A5W5GGYZ.AH0008IJ. The first part of the two-level identifier, A5W5GGYZ, indicates the study repository where this particular QS table resides. The other part of the instance identifier, AH0008IJ, is the unique identifier of the QS table within this study repository.



TableId	SASTableName	SASTableLabel
A5W5GGYZ.AH0008IJ	QS	Questionnaires - QS

Figure 9. QS table label and instance identifier from Metadata Extractor

For the QS table extended attributes, for example Structure shown in Figure 10, its instance identifier is A5W5GGYZ.AK000DI1. It tells us that this object resides in the repository A5W5GGYZ that is the same as the QS table. The unique identifier of this extended attribute within the repository is AK000DI1. Further, the owning object of this extended attribute is the QS table, of which the instance identifier is A5W5GGYZ.AH0008IJ, the same as in Figure 9.

Owning object

ExtAttribID	ExtAttribName	ExtAttribValue	ExtAttribDesc	TableID	SASTableName
A5W5GGYZ.AK000DHZ	CLASS	Findings	Domain Type	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DHY	DOMAIN	QS	Domain Abbreviation	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DI0	LOCATION	qs.xpt	Transport Filename	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DI4	PROVIDER	CDISC	Provider	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DI2	PURPOSE	Tabulation	Purpose	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DI1	STRUCTURE	One record per question per subject	Domain Structure	A5W5GGYZ.AH0008IJ	QS
A5W5GGYZ.AK000DI3	VERSION	3.1.1	CDISC Version Number	A5W5GGYZ.AH0008IJ	QS

Figure 10, QS table extended attributes and instance identifiers from Metadata Extractor

Similarly in Figure 11, the QSCAT column has the instance identifier A5W5GGYZ.ZI000DV3. And Figure 12 shows that the QSCAT column as the owning object of its extended attributes.

ColumnId	SASColumnName	ColumnDesc	SASColumnLength	SASColumnType	TableId	SASTableName
A5W5GGYZ.AI000DV3	QSCAT	Category of Question	40	C	A5W5GGYZ.AH0008IJ	QS

Figure 11. QSCAT column attributes and instance identifiers from Metadata Extractor

Owning object

ExtAttribID	ExtAttribNmae	ExtAttribValue	ExtAttribDesc	ColumnID	SASColumnName
A5W5GGYZ.AK000DFP	CORE	Req	Core	A5W5GGYZ.AI000DV3	QSCAT
A5W5GGYZ.AK000DFN	ORIGIN	Sponsor Defined	Origin	A5W5GGYZ.AI000DV3	QSCAT
A5W5GGYZ.AK000DFO	ROLE	Grouping Qualifier	Role	A5W5GGYZ.AI000DV3	QSCAT
A5W5GGYZ.AK000DFM	TERM	*	Controlled Term or Format	A5W5GGYZ.AI000DV3	QSCAT

Figure 12. QSCAT column extended attributes and instance identifiers from Metadata Extractor

METADATA EDITOR

A custom transformation called Metadata Editor is developed to generate XML and write to the SAS metadata. It focuses on the table and column extended attributes. For user interface purposes, the custom transformation is incorporated into a SAS DI job as the front end.

Figure 13 is a simplified diagram of the DI job. There are four basic steps. Firstly it imports the study data specifications from the two CSV files into SAS tables. Secondly, via the DI job, users define the names of the extended attributes that are to be edited. Then the names are mapped to macro variables. And finally the macro variables are passed to the Metadata Editor custom transformation, which runs to create XML and edit the metadata.

Step 1: Importing CSV files

The DI job has two sources of user-controlled input. The first is the two CSV files containing the study-specific table specifications (Figure 1) and column specifications (Figure 2). They are read in and saved as two temporary SAS data sets named TableSpec and ColumnSpec.

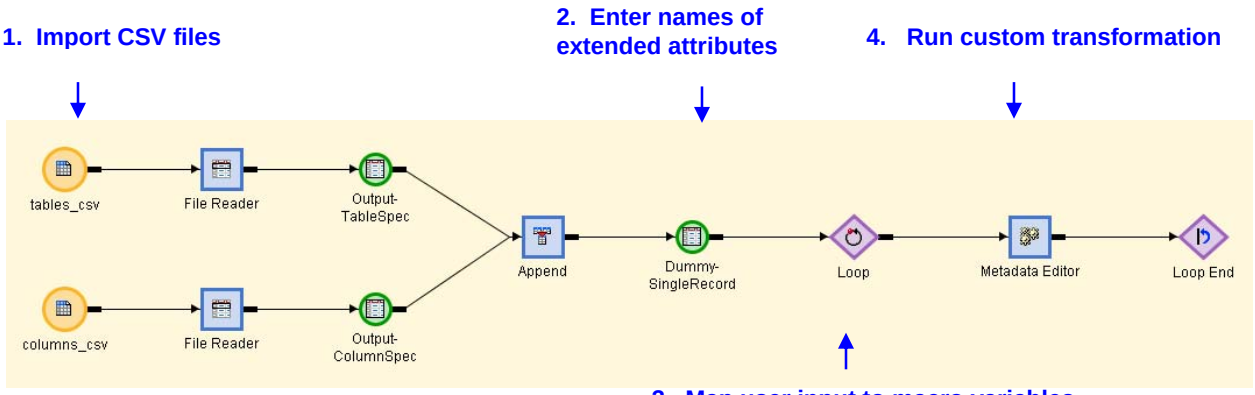


Figure 13. Diagram of SAS DI job

Step 2: Entering names of extended attributes

The second source of user-controlled input is the names of the extended attributes of interest. They are to be defined by users prior to executing the DI job. The user input is kept in a temporary SAS data set that contains a dummy record. Figure 14 shows the structure of this dummy record. On the left, the Column displays the pre-defined variable names in this dummy record, and on the right the Column Description.

The middle column Expression in Figure 14 is entered by users and shows the values of the variables in the dummy record. The values on the first six variables are indeed the names of the extended attributes. In this example, they are the table extended attribute “Structure“ to be updated, “Keys“ to be added, nothing to be deleted, the column extended attribute “Origin“ to be updated, “StudyControlledTerminology“ to be added, and “Term“ to be deleted. The corresponding variable names are update_table, add_table, delete_table, update_column, add_column and delete_column, respectively.

The values on the last three variables in Figure 14 are the physical locations of the output SAS data sets from the Metadata Extractor (Figures 9 through 12), the resulting XML, and the response messages from Proc Metadata at run time, respectively. The latter two will be discussed next in Step 4.

User input

Column	Expression	Column Description
update_table	"Structure"	Table extended attribute(s) to be updated, space delimited
add_table	"Keys"	Table extended attribute(s) to be added, space delimited
delete_table		Table extended attribute(s) to be deleted, space delimited
update_column	"Origin "	Column extended attribute(s) to be updated, space delimited
add_column	"StudyControlledTerminology"	Column extended attribute(s) to be added, space delimited
delete_column	"Term"	Column extended attribute(s) to be deleted, space delimited
permlib	compress("/sas/sasdata/t07...	Full path of Temp libname to save output data sets
request	compress("/sas/sasdata/t07...	Full path of Temp folder to save the request script
response	compress("/sas/sasdata/t07...	Full path of Temp folder to save the response script

Figure 14. Temporary SAS data set with single dummy record retaining user input

Step 3: Mapping user input to macro variables

The mapping of the user input, that is, the variables in the dummy record, to macro variables combines the DI front end and the Metadata Editor custom transformation. That is because the custom transformation is in essence a macro with predefined macro variables. The values on the macro variables are to be passed from the Loop node of the DI job (Figure 13), and the input of the Loop node is the dummy record (Figure 14). Figure 15 illustrates how such one-to-one mapping takes place. On the left side of the arrows are the columns from the dummy record, and on the right side the pre-defined Macro Variable Names from the custom transformation. In the middle the mapping is shown

in the form of the arrows and is done manually at design time. The Column names and the Macro Variable names do not have to be the same, but here they are identical by design for easier maintenance.

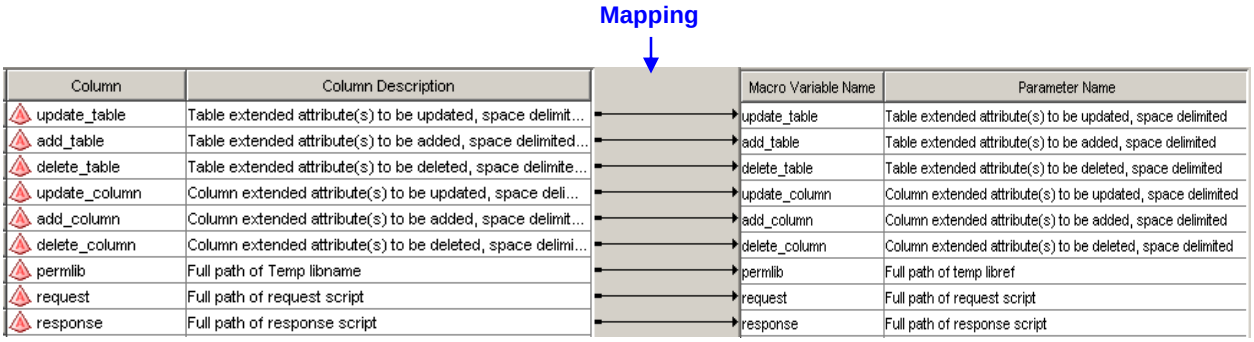


Figure 15. User input mapped to macro variables

Step 4: Running Metadata Editor custom transformation

The custom transformation is a macro itself written in Base SAS programming. First of all it merges the study data specifications from the CSV files with the instance identifiers from the Metadata Extractor. Next it uses %DO-%END blocks to process the values on each of the six macro variables, &update_table, &add_ table, &delete_table, &update_column, &add_column, and &delete_column (Figure 15).

Figure 16 is partial source code from the custom transformation. It shows the %DO-%END block for the macro variable &update_column. Recall the value on &update_column is “Origin” (Figure 14), so the Data _Null_ step inside this %DO-%END block runs only once which is for the column extended attribute Origin. Should there be multiple extended attribute names on &update_column, the Data _Null_ step would repeat for each extended attribute.

Within the Data _Null_ step, there are two PUT statements to write text to an external text file. The first PUT statement is to output an XML comment stating the column extended attribute being processed - in this case, Origin. The second PUT statement is to write text for each of the SDTM columns that have the column extended attribute Origin in the SAS metadata.

As a side note about the Metadata Editor custom transformation, there is the Loop node in the DI job (Figure 13), which means it has the functionality to run the custom transformation multiple times. However, by design, only the dummy record feeds the Loop node, so the entire custom transformation executes one time only; the actually looping is carried out by the %DO-%END block and PUT statements in the Data _Null_ step.

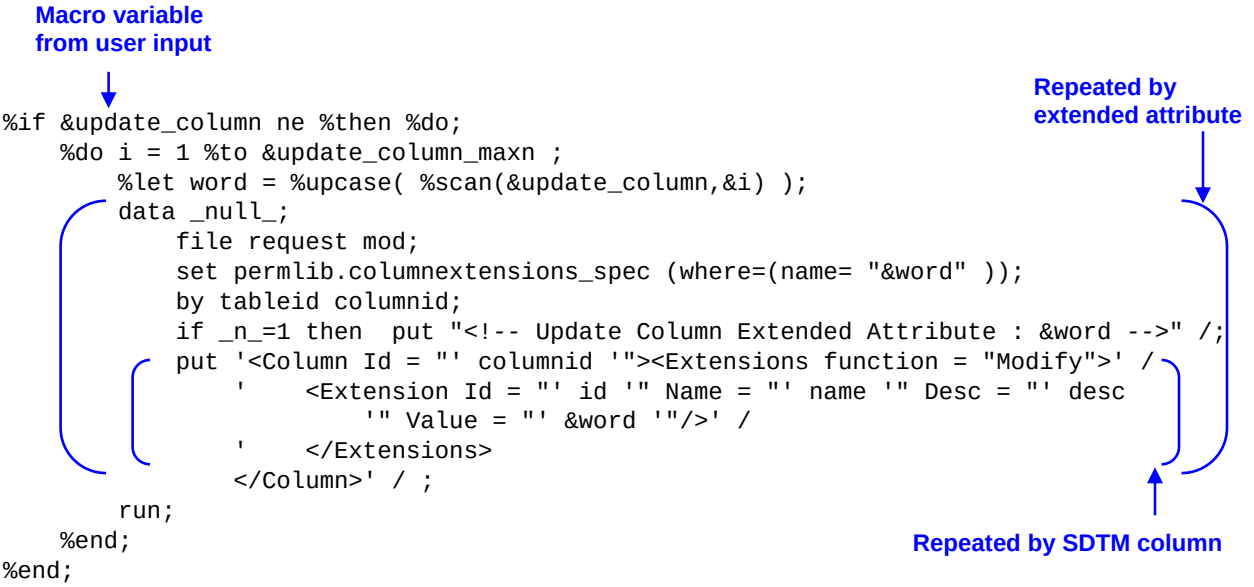
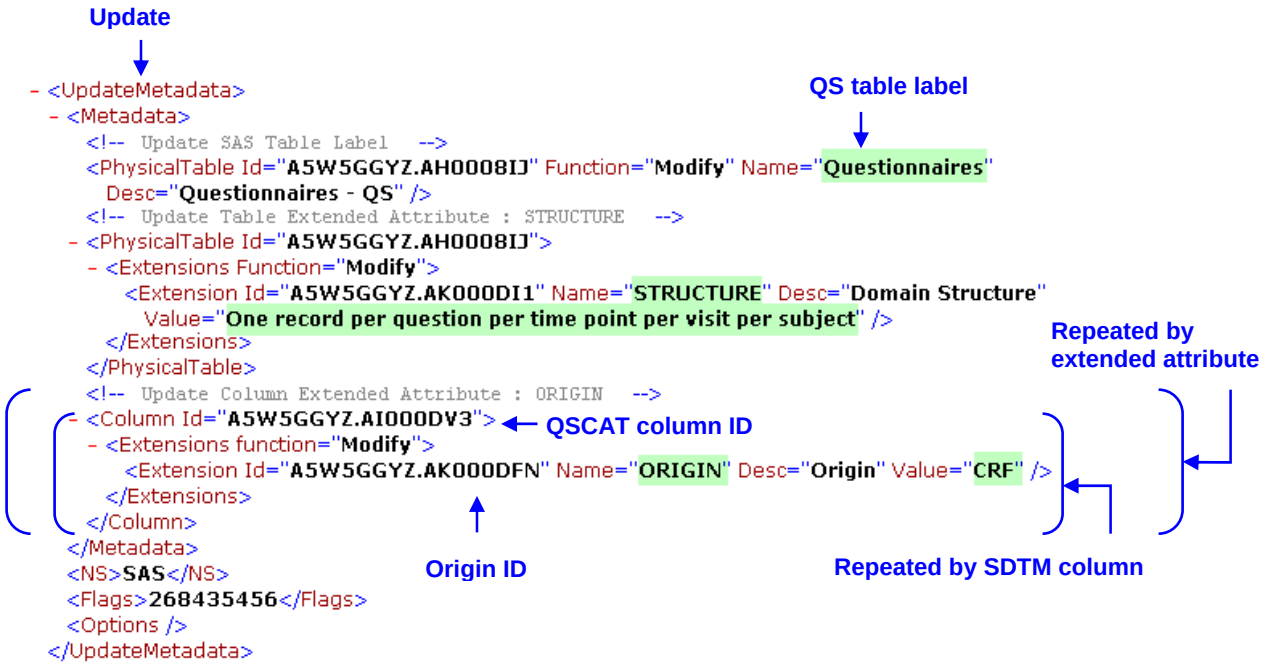


Figure 16. Partial source code in Metadata Editor custom transformation

The output of the Data _Null_ step is an XML string. Figure 17 shows sample XML that is used to update the column extended attribute Origin. First of all, XML declares the metadata method UpdateMetadata for updating. The XML comment "<-- Update Column Extended Attribute : ORIGIN -->" signals the beginning of updating the Origin object for each SDTM column that owns such a named object in the study repository. This XML comment corresponds to the first PUT statement in the Data _Null_ step (Figure 16).

The inner pair of brackets in Figure 17 correspond to the second PUT statement in the Data _Null_ step. The <Column> element locates the owning object, which is the QSCAT column, by its instance identifier A5W5GGYZ.AI000DV3 (Figure 12). It calls the Modify function in the <Extensions> element, which means it is going to update the extended attributes associated with the QSCAT column. Then it goes on with the <Extension> element to access the specific extended attribute Origin by its instance identifier A5W5GGYZ.AK000DFN (Figure 12), and finally assigns the value of 'CRF" to the Origin extended attribute (Figure 2).

Updating the table extended attributes, for example, Structure on the QS table, is similar, except that the <PhysicalTable> element is used in place of the <Column> element as shown in Figure 17. This XML example also updates the QS table label to "Questionnaires", which resides in the Name property of the QS physical table object.




```
- <UpdateMetadata>
  - <Metadata>
      <!-- Update SAS Table Label   -->
      <PhysicalTable Id="A5W5GGYZ.AH0008IJ" Function="Modify" Name="Questionnaires"
        Desc="Questionnaires - QS" />
      <!-- Update Table Extended Attribute : STRUCTURE   -->
    - <PhysicalTable Id="A5W5GGYZ.AH0008IJ">
      - <Extensions Function="Modify">
          <Extension Id="A5W5GGYZ.AK000DI1" Name="STRUCTURE" Desc="Domain Structure"
            Value="One record per question per time point per visit per subject" />
        </Extensions>
      </PhysicalTable>
      <!-- Update Column Extended Attribute : ORIGIN   -->
    - <Column Id="A5W5GGYZ.AI000DV3">
      - <Extensions function="Modify">
          <Extension Id="A5W5GGYZ.AK000DFN" Name="ORIGIN" Desc="Origin" Value="CRF" />
        </Extensions>
      </Column>
    </Metadata>
    <NS>SAS</NS>
    <Flags>268435456</Flags>
    <Options />
  </UpdateMetadata>
```

Figure 17. XML to update SAS table label, table extended attribute Structure, and column extended attribute Origin

In Figure 18, the metadata method AddMetadata is to add new objects to the QS table. For example, the new extended attribute Keys. Since this deals with new objects that do not yet exist, XML leaves the instance identifier blank, but defines the Name, Value and Desc of the new extended attribute. The <OwningObject> element defines the instance identifier of an existing object that owns the new object. In this example, the particular Keys object belongs to the QS table of which the instance identifier is A5W5GGYZ.AH0008Ij (Figure 10).

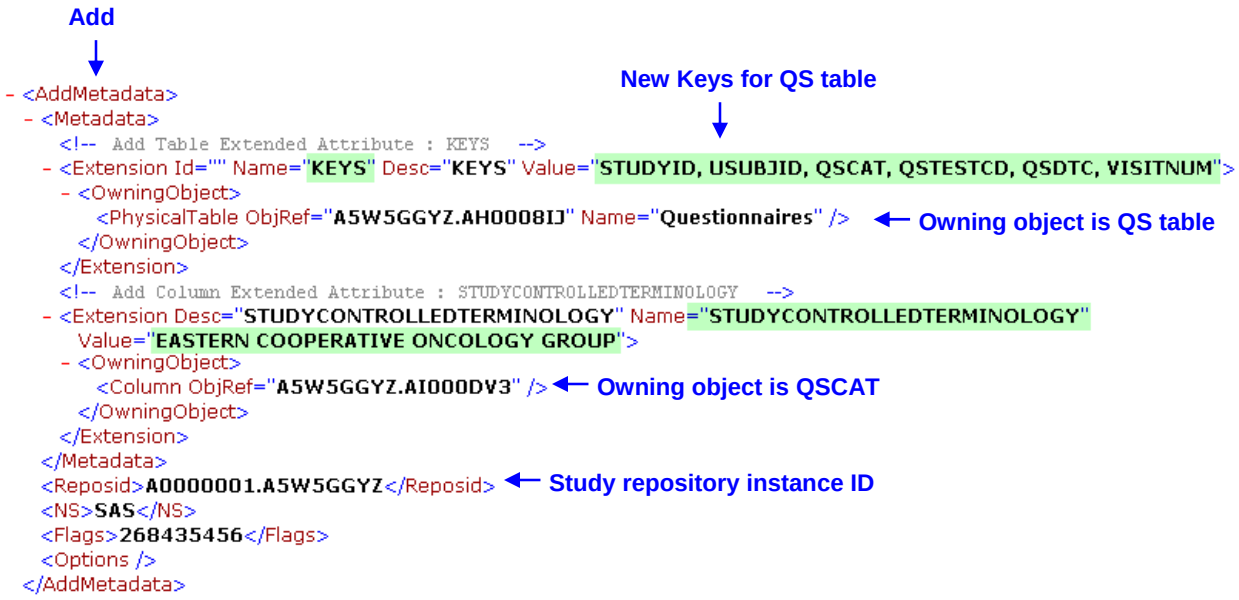


Figure 18. XML string to add table extended attribute Keys, and column extended attribute Study Controlled Terminology

Figure 19 shows how to use the metadata method DeleteMetadata to delete existing objects. XML specifies the instance identifier of the extended attribute to be deleted, A5W5GGYZ.AK000DFM (Figure 12), the Term extended attribute that belongs to the QSCAT column. In the case of deleting objects, XML does not need to spell out the owning object. It will not cause confusion since all the objects in the SAS Metadata Server have their own unique two-level instance identifier.

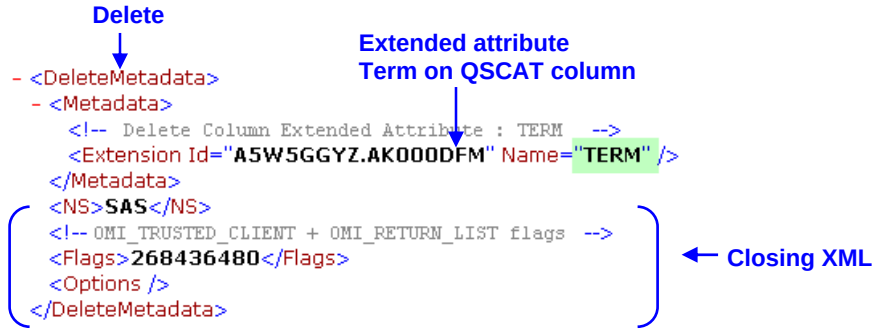


Figure 19. XML string to delete column extended attribute Term

Each of the above XML strings is saved as an external file. The location of the external file is defined by users on the variable REQUEST in the dummy record discussed previously (Figure 14). The Request file is submitted to the SAS Metadata Server with the IN option of Proc Metadata as in Figure 20. Proc Metadata returns messages at run time. The messages are saved as another external text file, the Response file, with the OUT option. The location of the Response file is also defined by users, which is the value on the variable RESPONSE in the dummy record (Figure 14).

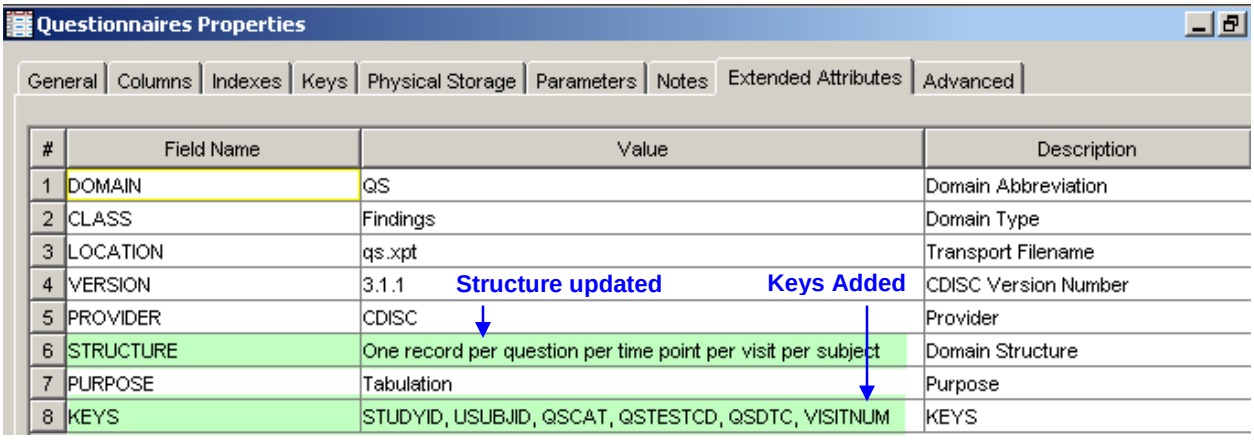
```
proc metadata in=Request out=Response verbose;
run;
```

Figure 20. Proc Metalib

After the Metadata Editor custom transformation runs successfully, the correct extended attributes are present on the

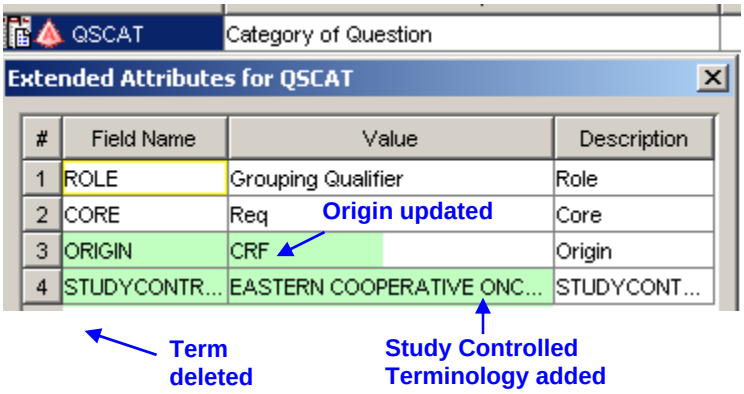
QS table and the QSCAT column as well as the QS table label are fixed. This is verified via the DI Studio interface as shown in Figures 21, 22 and 23. In Figure 21, the QS table label is updated. In Figure 22, the QS table extended attribute Structure is updated and Keys is added. And finally in Figure 23, the QSCAT column extended attribute Origin is updated, Study Controlled Terminology added, and Term deleted. The values on these extended attributes and the table label match the study-specific data specifications (Figures 1 and 2).

Figure 21. QS table label updated after Metadata Editor is run



#	Field Name	Value	Description
1	DOMAIN	QS	Domain Abbreviation
2	CLASS	Findings	Domain Type
3	LOCATION	qs.xpt	Transport Filename
4	VERSION	3.1.1	CDISC Version Number
5	PROVIDER	CDISC	Provider
6	STRUCTURE	One record per question per time point per visit per subject	Domain Structure
7	PURPOSE	Tabulation	Purpose
8	KEYS	STUDYID, USUBJID, QSCAT, QSTESTCD, QSDTC, VISITNUM	KEYS

Figure 22. QS table extended attributes updated or added after Metadata Editor is run



#	Field Name	Value	Description
1	ROLE	Grouping Qualifier	Role
2	CORE	Req	Core
3	ORIGIN	CRF	Origin
4	STUDYCONTR...	EASTERN COOPERATIVE ONC...	STUDYCONT...

Figure 23. QSCAT column extended attributes updated, added or deleted after Metadata Editor is run

CONCLUSION

The SAS Metadata Server can serve as the central storage repository of valuable information to produce documents for electronic data submission. However, the first and foremost requirement is the quality of the metadata. As opposed to manually editing that metadata using DI Studio or the Management Console, the automated approach described in the paper inputs directly from the sponsor-approved data specifications in text-based CSV files to the SAS Metadata Server via XML. This approach not only eliminates potential data entry errors, it saves time and adds efficiencies when dealing with a large number of fields.

DISCUSSION

The DI Studio version in use at the writing of this paper is version 3.4, but the newer version of DI has had many improvements. One example is the fix of the Update Table Metadata plug-in. It will update SAS table labels based on the physical SAS tables. So the XML that updates the SAS table label, for example, is not needed if using the newer version of DI.

One reason for choosing DI is for its functionality as the front end GUI interface between DI to the SAS Metadata Server. However, the interface via DI jobs where user input takes place is not the often seen in a format using dropdown lists or radio buttons. So it is recommended that the targeted users of this tool have basic knowledge of DI Studio.

Exception handling has a part in the custom transformation but not thoroughly. For example, it is assumed that the extended attribute names are the same as the variable names in the SAS data sets TableSpec and ColunnSpec imported from CSV files. Users must examine and plan for the data sources carefully. As always it is extremely important to back up the metadata prior to such batch processing.

Alternatively for Java developers, a custom plug-in using the SAS DI Studio API is a good choice to replace the Metadata Editor custom transformation and the DI job. It will provide a more user-friendly interface.

ACKNOWLEDGMENTS

The author would like to extend a special thanks to Robert Stemplinger for his review of this paper, as well as his ongoing support for progress, creativity and innovation.

RECOMMENDED READING

SAS 9.1 Open Metadata Interface User's Guide, SAS Institute
SAS 9.1 Open Metadata Interface Reference, SAS Institute

CONTACT INFORMATION

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

Name: Annie Guo
Enterprise: ICON Clinical Research
Address: San Francisco, CA, USA
Work Phone: 215-616-6597
Fax: 215-240-7595
E-mail: annie.guo@iconplc.com
Web: www.iconplc.com