

Paper 98-30

Best Practices in Enterprise Data Management

Gary Mehler, SAS Institute Inc., Cary, NC

ABSTRACT

The powerful SAS[®]9 Enterprise ETL Server platform has support for multi-user, multi-server and multi-level deployments. This paper discusses best practices for planning, implementing, and deploying a production end-to-end data management environment to support intelligence initiatives across an enterprise. In addition, considerations for advanced data management and high-performance computing are covered.

INTRODUCTION

This paper covers advanced data management topics that are typically encountered with a large-scale data warehousing project. The common use of “data warehouse” refers to a body of information that is collected and structured to facilitate a task such as business intelligence. Another term in this area is ETL (Extract, Transform, and Load), which is the process of constructing the elements of a data warehouse. To build a data warehouse, ETL processes are defined. The overall life cycle of developing and maintaining a data warehouse is called data management.

The information presented in this paper follows from a paper “Next Generation Warehousing with Version 9” (Mehler, 2004) that discusses the foundational elements of data management, which includes metadata, planning for a multi-user and a multi-level environment, integrating data quality, and working with a wide range of source system types. The previous paper presents the more foundational aspects of data management use. This paper takes a more pragmatic approach to explain how the foundational components can be used to produce more value in the real world. Examples in this paper are demonstrated in SAS ETL Studio, the visual ETL design component of SAS ETL Server.

A few main topics will be covered in turn, with a final section on Tips and Tricks, which are really common practices. The Tips and Tricks section can help explain how some seemingly minor decisions can lead to increased productivity during data management projects.

REUSABLE TRANSFORMATIONS

As mentioned earlier, ETL processes are defined in order to build a data warehouse. The active part of ETL is the Transform step. In a transformation, source data is transformed—either in structure or in content—to more closely meet the needs of the load target. Transformations can range from very simple operations to complex tasks. In addition, larger warehousing projects can require long sequences of transformations to accomplish various types of required actions. Figure 1 shows how larger chains of transformational logic can be built to meet data warehousing needs.

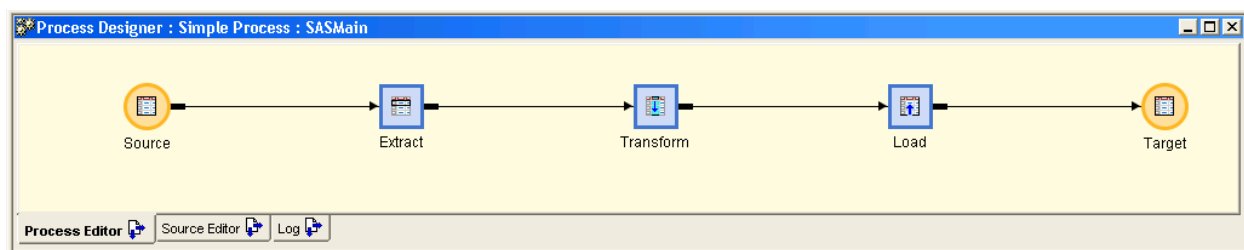


Figure 1. A Basic ETL Flow Shown in SAS ETL Studio: **Extract** Data from a Source Table, **Transform** it, and **Load** into a Target Table.

In the most basic example, data uses some extraction logic (either all rows of a table can be used, or some subset can be applied such as only including last year’s values), some transformation logic, and loading logic (to specify whether resulting data is appended to or replaces existing data). The transformation logic can range from a very simple operation to something more complex. Clearly, the time that’s spent developing a complex transformation suggests that you want to reuse it, but the point is that *any* transformation logic is important and a candidate to become a standard, reusable process.

Let’s look at a few examples of common transformational logic and how it might be reused.

- **String manipulation of dates.** When bringing data together from multiple sources, a data integration problem can result if the sources have different formats for the strings that are used to store values. These are commonly encountered with proprietary date formats. If a lot of time is spent on the logic, the process should be documented for the benefit of others and to make it easy to reuse. Documentation is relatively easy if the organization has a forum for sharing this type of information, but ease of reuse is an important related concern. In addition, documenting the process makes it easier for users of the transformational logic to see whether they have correctly completed the desired task.
- **Accounting functions.** When performing basic financial operations, tasks can actually be harder than they appear. For example, determining all revenue for a product might seem to be a straightforward task of multiplying unit price by quantity for each sale. However, when discounts, returns, and currency issues are considered, this can become a very complex operation. It might not be apparent to developers of the transformations that they have all the logic correct, unless they have a standard report to compare against. Even worse, they might not have sufficient sample data available to completely validate the transformational logic, which makes their task even more difficult.

In addition to the need for these common transformations, having a convenient place to store and retrieve them is a secondary concern. If ETL developers must go out of their way to locate and use a customized transformation, the likelihood of them using it is reduced.

Figure 2 shows the Process Library tree in SAS ETL Studio. In addition to the basic transformations in various areas, a new group of "Best Practice Transformations" has been added to this library. This type of group can be created by a set of users in an organization to maintain their library of standard functions that they expect to be reused.

SAS ETL Studio has a facility named the Transformation Generator, which is used to define custom transformations and add them to a Process Library. See the results in Figure 2. The Transformation Generator captures the transformational logic and any parameters and inputs or outputs that are used by the designed transformation.

The transformational logic that's supported by SAS ETL Studio is SAS language code, which can include the SAS 4GL DATA step language, SAS procedures, or SAS macro language. In this way, any pre-existing library of standard functions that are written in the SAS language can be incorporated directly into the SAS ETL Studio environment.

Parameters enable ease-of-use and flexibility when using this transformation in ETL flows. Parameters that can control how the inner workings of the transformation code really execute are defined in the transformation, for example, if a transformation needs to support multiple compatibility modes of operation. By specifying the compatibility mode when used, the ETL developer can determine whether old or new functionality is desired. Figure 3 shows the selection panel for a reusable transformation. The types of options in the panel, as well as constraints on allowed values (either from a list or a lookup table), can be set up when the transformation is designed. Other types of selectable options can be numeric, retrieved from a lookup table, involve selection of a registered table or a column, or several other types of selections.

These values are configured when the transformation is designed and selected whenever an instance of the transformation is used in a process flow.

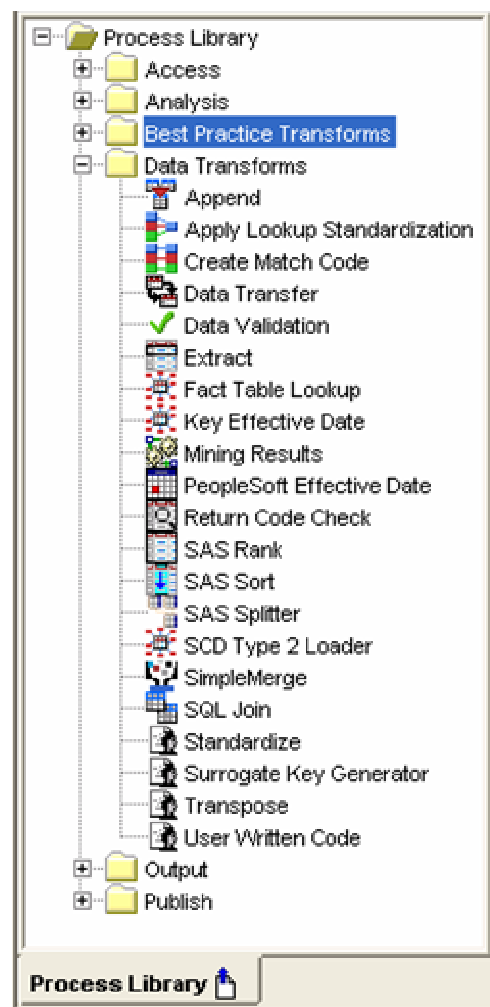


Figure 2. The SAS ETL Studio Process Library Tree Showing a Set of Standard Transformations and a Group Added for Best Practice Transformations

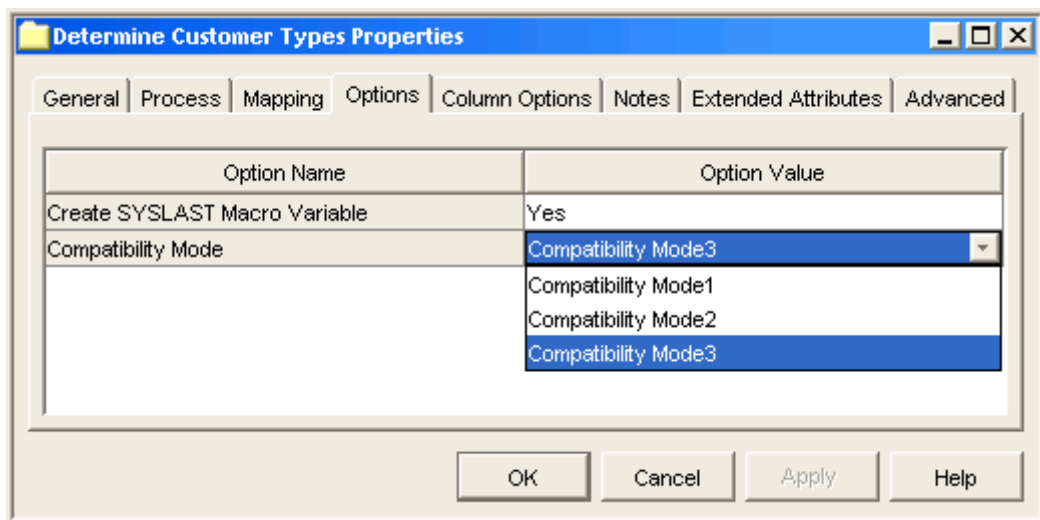


Figure 3. Properties in the New Transformation Support Modes of Operation

Figure 3 shows that this transformation supports three compatibility modes. Input/output parameters control how the transformation fits into the SAS ETL Studio Process Designer. In Figure 1, you see that the node named “Transform” has both an input table and an output table. This is a standard data-flow transform because it propagates data from its source to a target. You can also design non-standard data flows such as a transformation that requires two inputs (for example, for actual input data and a reference table of allowed values) or can generate multiple outputs (such as customer segmentation). The logic in the transformation is known to be correct and consistent across multiple uses of it in various ETL processes.

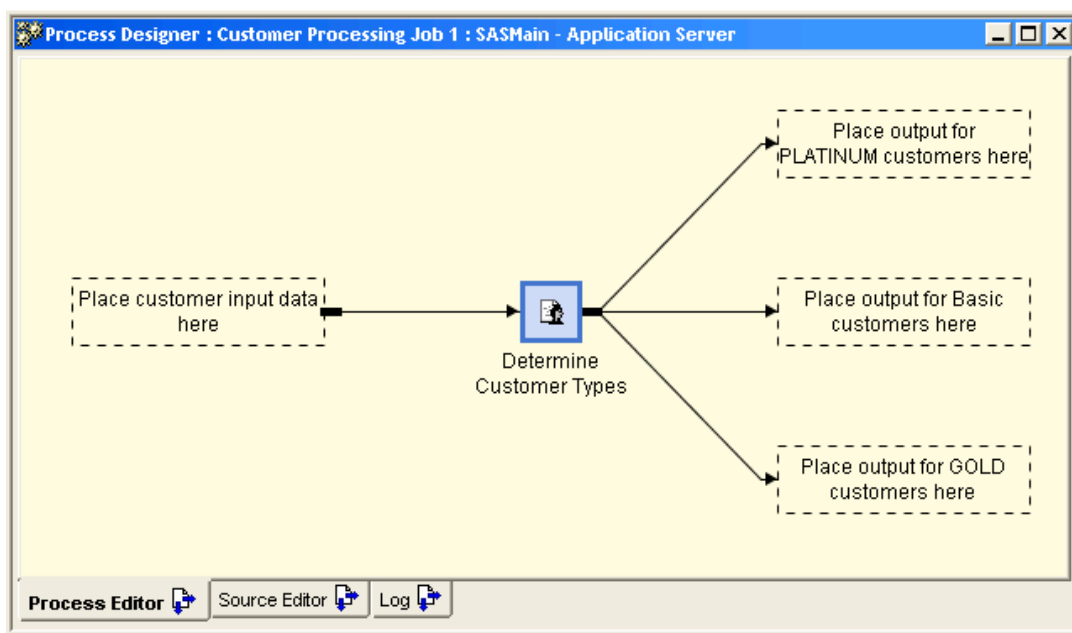


Figure 4. A Best Practice Transformation That Handles a Specific Set of Inputs and Outputs

The transformation shown in Figure 4 segments a set of customers into PLATINUM, Basic, and GOLD, as shown by the text in the template positions. The template positions indicate the expected use of the transform.

After specifying inputs and outputs, the transformation has a basic template format that is shown whenever the transformation is dropped onto the SAS ETL Studio Process Editor. The format can also be order-dependent and have a variable set of either inputs or outputs. For example, the transform shown in Figure 4 has a range of 1 to 3 for

output template slots. This means that the transform will work with a single output for PLATINUM customers, and generate output for GOLD customers and then for Basic customers, if needed.

This is a safe way to ensure that customer segmentation is done consistently (the logic is in the SAS ETL Studio Process Library), configurably (a compatibility mode and type of desired outputs are specified when the transform is used) and easily (just drag the transformation onto the Process Editor to see the template structure).

After a transformation is used, it is probable that it will need modification in the future to support a new compatibility mode. If this occurs, it will be necessary to perform an impact analysis to find the places in which the transformation is being used in order to assess whether the new mode is needed in those cases. This impact analysis (Figure 5) is important for two reasons: 1) to assess and handle cases in which the new mode will be used, and 2) to provide a validation list for cases in which the transform might need to be re-tested even if no new mode is needed. This is an important life-cycle consideration for the transform. It should always be assumed that everything will change over time and having mechanisms in place to locate and treat those changes makes life easier.

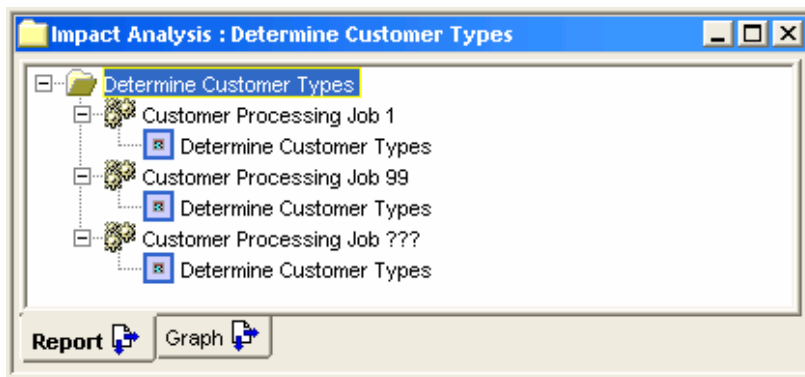


Figure 5. Impact Analysis on the Transform Shows Three Places Where It Is Used

Because the point of Best Practices is to ensure consistent and reliable results, encapsulating logic into reusable transforms as described earlier is a very good way to ensure that maximum benefit is achieved. Reusable transforms are easy to use, can encapsulate complex logic, have a definable input/output interface and optional modes, and can be tracked by performing an Impact Analysis to help ensure that they're being consistently and correctly used.

WORKING WITH FLAT FILES

Although the data management process involves placing data into relational (tabular) or multidimensional (OLAP) form, much of the world's data still resides in text storage on various systems. Commonly called flat files, these are unstructured, basic files that can reside on PC or UNIX file systems as text files or on mainframes in VSAM storage. Some estimates indicate that a very large portion of the world's data still resides in VSAM, so being able to work with that data format is important in many data management projects.

Two common types of flat files are delimited and fixed-width files. Delimited files rely on a separator character such as the comma to indicate the different fields (or columns) of data to retrieve from the file. In some cases, the first row of the file can contain the list of variable (or column) names. Although this seems like all the information that's needed, you don't know the type of the variables, if a maximum length is expected, or how information about the values should be interpreted (such as using a number as a form of date value).

Fixed-width flat files don't use delimiters. These files depend on positions in the text line to determine how the values are interpreted and usually don't include any information other than the concatenated values.

The challenge of flat files is that, in most cases, they don't bring enough of their own metadata along with them. Here, "metadata" means information on how to interpret the file's contents. This is in contrast to relational table storage such as SAS data sets or DBMS tables that have this information stored in data set headers or DBMS system tables, respectively. When this information is not included in the file, it must either be specified at the time the data is used or be dependent on the consuming application to interpret the incoming data to formulate a guess about its metadata.

Because Best Practices also suggest not leaving anything to chance (as this can lead to user error or misinterpretation of data), this isn't a desirable course of action for the large volumes of flat-file data that need to be processed. What is desirable is a way to specify flat-file metadata information in a repeatable and safe way. SAS ETL Studio provides a way to define flat-file contents in—what else?—a flat file! In what is called an External Format File, a comma-separated value (CSV) text file is used as input to the flat-file reading process. This can ensure that all the needed information about the contents of a flat file is correct and complete. In SAS ETL Studio, this is called getting column definitions from a format file (Figure 6).

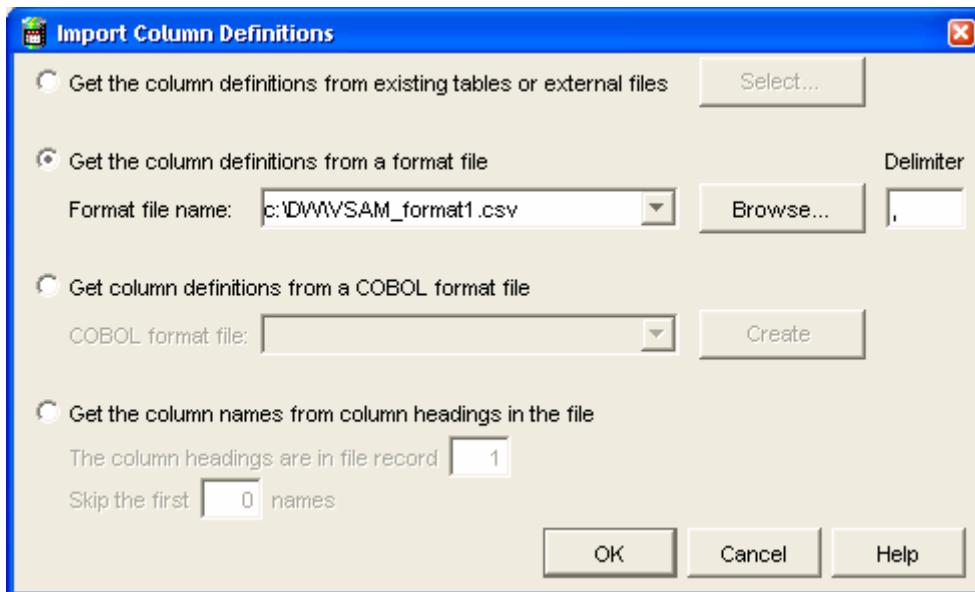


Figure 6. Importing Column Definitions from a Format File That Has a VSAM Internal Format Predefined for Reuse

As an example of what the format file can contain, here's a sample definition of a mainframe file. Because the file itself is a text file, it can be edited in any text editor and referenced from SAS ETL Studio. This file holds the contents of the format file "VSAM_format1.csv", which is entered in the field in Figure 6.

```
# This is a simple External Format File
# Header follows
SASColumnName,SASColumnType,BeginPosition,EndPosition,SASColumnLength,SASInformat
# Column definition records follow
# SASColumnLength is expressed in bytes
EFFNAME,C,1,8,8,$EBCDIC8.
EFFINITIALS,C,9,10,2,$EBCDIC2.
EFFYOBIRTH,N,11,14,8,S370FZD4.0
EFFMOBIRTH,N,15,16,2,S370FZD2.0
EFFBIRTH,C,17,17,1,$EBCDIC1.
```

The format file can contain a wide range of information about the contents of a flat file, as shown in the following list.

- BeginPosition
- ColumnLength
- ColumnName
- ColumnType
- Desc
- EndPosition
- IsDiscrete
- IsNullable
- Name
- ReadFlag
- SASAttribute
- SASColumnLength
- SASColumnName
- SASColumnType
- SASExtendedColumnType
- SASExtendedLength
- SASFormat
- SASInformat
- SortOrder
- SummaryRole

If the definition already exists because the file was created by COBOL and is described by a Copybook, this information also can be imported into SAS ETL Studio (See Figure 7).

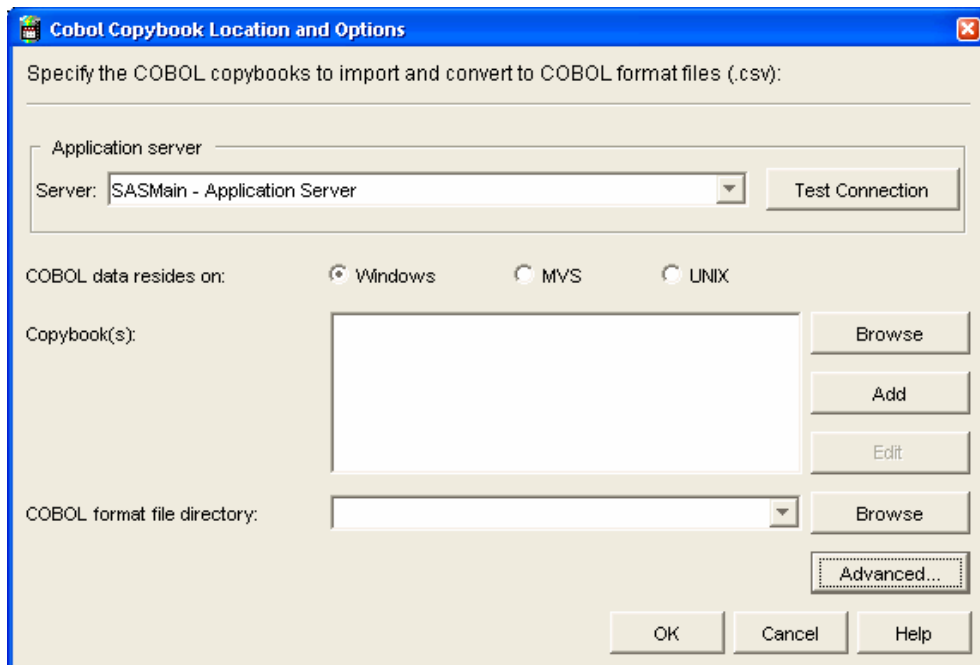


Figure 7. Using a COBOL Copybook to Define the Internal Structure of a Flat File

In addition to starting with predefined flat-file format definitions, it is important to have a powerful application to apply and validate the definitions visually on real or sample data. This is useful because it provides a quick way to ensure that the unprocessed data in the flat file looks as expected. During test processing, it is helpful to be able to easily inspect the offending data to see what might need correction. Along with inspection of unprocessed data, visual inspection of processed data can give a quick indication of whether common flat-file errors have occurred, for example, off-by-one problems in which an extraneous or missing separator causes processed columns to be placed incorrectly. In the example shown in Figure 8, external file definitions have already been imported, and it is possible to see the unprocessed (file) and column-processed (data) values, which helps ensure that the column metadata is correct for the data that's being processed

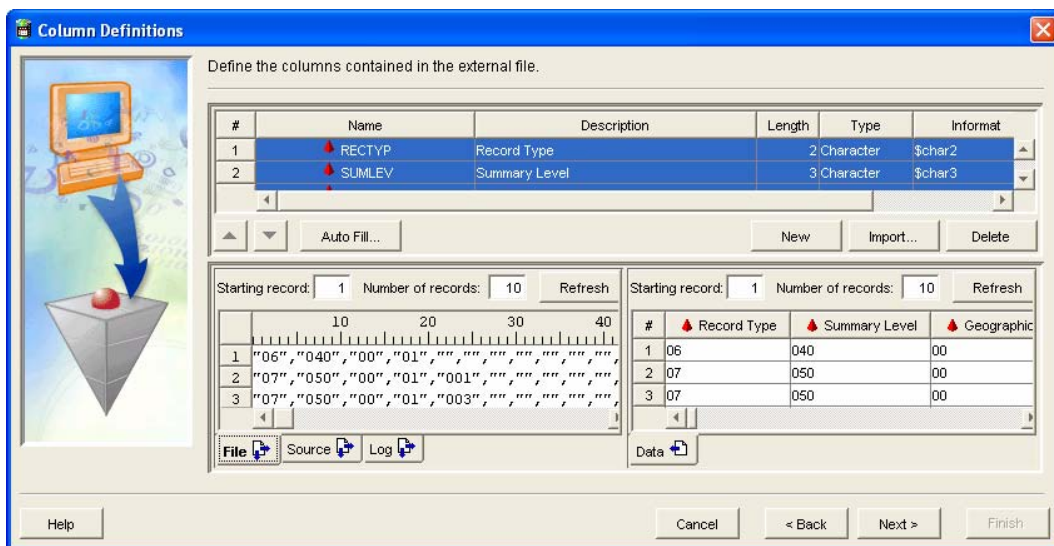


Figure 8. A Column Entry and Validation Panel

SHARING METADATA TO MEET SPECIFIC NEEDS

The process of defining ETL flows and related application settings results in the generation of metadata that is stored in a central repository. Applications such as SAS ETL Studio and SAS Management Console generate and consume this metadata for defined activities. Therefore, basic activities don't require much knowledge of what goes on in the metadata tier of an enterprise data management system.

Metadata is manipulated by various operations of SAS ETL Studio, and some of these operations can be used for more powerful ends in some cases. This section describes how these operations work and the extra benefits that can be gained.

SAS ETL Studio provides various ways that metadata can be manipulated with operations such as Export to File and Import, and Copy/Paste. These operations transform an internal metadata representation to an XML text format. This is a straightforward operation for SAS ETL Studio because the language it uses to communicate with its metadata repository is an XML query language. There will be more on this topic later; for now, let's look at XML and what you can do with it. First, some explanation of what can be done.

If you look at the Copy/Paste operation, XML enables the user to select an object (through the Copy action) and replicate it (through a Paste action). Actually, it supports single or multiple objects such as table definitions, process flows, user-defined transformations, or more basic objects such as notes or document links. In an application that runs on the Microsoft Windows platform, Copy/Paste is achieved by using the system clipboard where the XML is stored until it is pasted into the SAS ETL Studio session. A similar mechanism is available for some metadata types such as process flows, in which these objects are written and read directly from XML files.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Copy Product="ETL Studio" Version="null">
  <ObjectCount Count="1" />
  <Object Type="TextStore">
    <Metadata>
      <TextStore Desc="" MetadataCreated="02Dec2004:12:34:11" Name="Note 1"
        MetadataUpdated="18Jan2005:13:50:58" StoredText="<html> <head>
          </head> <body> <p> This is a note </p> </body> </html>" LockedBy=""
          TextType="html" TextRole="Note" Id="$1">
        <Extensions />
        <Trees />
        <ResponsibleParties />
      </TextStore>
    </Metadata>
  </Object>
</Copy>
```

This is the XML representation of the exported object, a "note," which is an HTML document that can be linked to other metadata objects as a human-readable document. This information was pasted into Windows Notepad from the Windows Clipboard, and then viewed in Microsoft Internet Explorer, which is able to nicely format XML text.

The XML representation itself isn't important to understand, but you are now looking at the metadata model that underlies SAS ETL Studio and the SAS®9 application suite. Here is a brief explanation. An object of type "TextStore" (the note itself), its attributes (when it was created or updated), and content (the StoredText) was exported. Once again, you don't need to get lost in what all this means. You just need to know that you can interact with portions of metadata that represent what is done in SAS ETL Studio.

This information can now be transported and reused in multiple ways. For example, the XML that is shown in Figure 8 can be pasted into a text editor for viewing, e-mailed to another ETL developer, pasted into another SAS ETL Studio session on another server, or saved into an off-line archive of metadata objects. Let's look into why some of these operations might be useful.

- **Exporting metadata into a text editor.** Sometimes business applications don't have rich support for features such as spell checking. Externalized metadata such as the preceding Note can be run through a spell checker in a word processor to look for typographical errors to help improve overall quality.

- **E-mailing to another ETL user.** If projects are being co-developed in distant locations, it might not always be possible to have completely current, synchronized metadata repositories at all times. If one ETL developer wants to share some interim progress with another developer (for review), this metadata can be e-mailed to the second developer, who can import or paste it into a repository for use. This mechanism isn't meant to replace official source control systems, but can provide an easy way to share small segments of metadata across a distributed team.
- **Pasting into another SAS ETL Studio session.** If a construct is needed from one project, it can be easily transported into another environment. This mechanism can be used to share the best practice transformations, which were described earlier, among a larger community of users.
- **Creating an off-line archive of metadata objects.** Metadata servers have built-in backup mechanisms (see "General Practices" below) that are the standard backup safety mechanism for entire repositories of metadata. However, there might be times when an ETL developer wants to save a specific version for personal use, or to switch in/switch out a different version. This can be accomplished if the developer can save and restore segments of metadata from a personal work area. Again, this isn't meant to replace standard backup mechanisms, but can be a more customized and/or timely approach for some types of ETL development efforts.

The metadata that was viewed earlier has traveled out of the SAS Metadata Server through the SAS Open Metadata Interface (Figure 9), as happens when you make a metadata query against the metadata server. You see that metadata can be transported, but you need to know a little more about how this really works, because metadata is highly associative. *Associativity* means that objects usually exist in a highly-interdependent set of metadata that represents business structures. To explain, let's look at the definition of a relational table. The table is composed of other objects such as columns, primary keys, indexes, and constraints, and can have associated documents (notes) or foreign keys. In addition, it has dependencies on the library in which it is stored, which in turn has dependencies on the server that provides access to the library. To understand this fully means getting into the metadata model, which this paper won't attempt. Suffice it to say that the interdependent nature of metadata means that some of this needs to be done in a certain way to guarantee success.

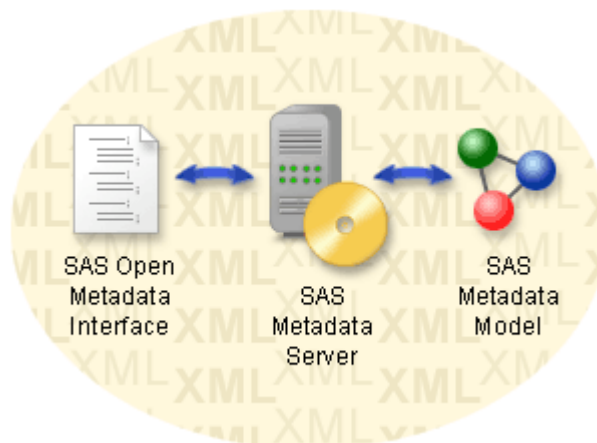


Figure 9. How the SAS Metadata Model Is Surfaced Through the Open Metadata Interface

Documentation for the SAS Metadata Model and Open Metadata Interface are available in many locations, including the SAS Web site (<http://www.sas.com>) and the online documentation for SAS products. Knowing all this, there are still some issues to consider regarding metadata transport in this way. It is important to remember that the metadata's dependence on other objects can partially constrain how it can be used through these mechanisms. For example, if a relational table definition is exported, it has a dependency on the library in which it's contained. If the library doesn't exist in the destination repository, the table won't be usable until it is re-attached to an appropriate library. The same kind of situation pertains to the export and re-importation of a process flow. The process flow itself references the tables on which it operates, but doesn't include the tables when exported. So, it expects to find them in the metadata when it's pasted back in. If the tables aren't there, the process flow won't have much value because it might not have source or target tables to operate on.

Using operations as described in this section and having some basic understanding of the SAS Metadata Model means that metadata definitions can be shared and used for other purposes. A good practice is to remain aware of the types of options that are available to teams of ETL developers and determine how and when to make use of them. Used in conjunction with standard metadata management tools, such as promotion from Development to QA/Test to Production, these options provide the flexibility to meet specific needs that can surface in ETL development environments.

BENEFITING FROM METADATA WAREHOUSING AND REPORTING

If the purpose of a typical data warehouse is to support business intelligence requirements such as providing a query and reporting system for business users, you can think of a metadata repository as a collection of information that can also support the ad hoc query and reporting system for its users. The direct users of metadata can be thought of as the IT organization, which might have as their business requirement to understand IT-oriented information. For example: What metadata really is in a system? Who “owns” metadata definitions? Is everything properly secured? When did something last change or what changed yesterday?

While these types of questions can be answered by standard applications such as SAS ETL Studio or SAS Management Console, advanced users can query the server to generate their own specific reports or provide data to answer other types of questions. The next section will briefly describe how to use the metadata interface with PROC METADATA (see the SAS online documentation or the SAS Web site for more information. Specifically, the *Open Metadata Interface: User's Guide* (for related examples) shows SAS language segments that can be incorporated into a customer transformation (see the section “Reusable Transformations”) for actual use and deployment. These segments are shown here as code sections that will be explained one piece at a time.

The first step is to specify how to connect to the metadata server. This is done through OPTION statements as shown here. These statements need to be included, even if run in a SAS ETL Studio session, because you might want to retrieve data from any metadata server in your environment. These options completely specify how to connect to the server that you want. Notice that a login and password are specified. Because the metadata server honors authentication settings, it is important to use a login.

```
option metarepository='Foundation';

option metaserver='MyServer.sas.com'
        metaport=8561
        metaprotocol=bridge
        metauser='domain\login'
        metapass='myPassword'
```

With these options set, code can be run that will access the metadata server to obtain the requested results.

In the next section, you'll construct a metadata request. It closely resembles the XML shown earlier describing the “Note”, but has no values in it because it is a request (GetMetadataObjects) that wants objects of the type “Change.” In addition, it requests metadata attributes for the change, which tables (PhysicalTables) were checked in, and who the Person is that performed the activity. The following code is written into a text file as input for PROC METADATA, which will execute the query request and return results to another text file. Infile CARDS4 syntax is used to construct the input text file. If this request will be re-executed, the following step can be performed just one time to generate the request file and, subsequently, referenced for the call to PROC METADATA.

```
/* Build the GetMetadata request. */
filename request 'c:\progs\work\request.xml';
data _null_;
  file request;
  infile cards4;
  length long $256;
  input;
  long=_infile_;
  put long ' ';
  cards4;
<GetMetadataObjects>
  <ReposId> $METAREPOSITORY </ReposId>
  <Type> Change </Type>
```

```

<Objects/>
<ns>SAS</ns>
<Flags>260</Flags>
<Options>
  <Templates>
    <Change MetadataUpdated="" Desc="" >
      <ChangeIdentity/>
      <Objects/>
    </Change>
  <PhysicalTable Name="" />
  <Person Name="" />
</Templates>
</Options>
</GetMetadataObjects>
;;;
run;

```

When executed, the preceding code constructs the file "request.xml," which serves as input to PROC METADATA. This is really a query, much in the way that communication to a DBMS might use SQL syntax. Because the interface to the metadata server is XML-oriented, this is an analogous request. Here is some information about the query itself. The GetMetadataObjects request expects a set of parameters, such as the type of object(s) requested, an important number in the FLAGS option, and, in this case, TEMPLATES. The FLAGS setting indicates how much metadata to retrieve, ranging from only what was specifically requested, to the setting of using a template to determine how much metadata is needed (this is the case here), to a different FLAG, which indicates that all attributes and related metadata are to be retrieved.

Templates are a useful way to control which attributes and associated objects should be retrieved. In this case, the request is for a subset of attributes of a Change: MetadataUpdated (when the change occurred), its Description, and the Objects that are associated with the Change. Of the associated objects, the request is for PhysicalTables (metadata terminology for a table) and Persons (who performed the action).

This request is executed by PROC METADATA as shown next. The request file and response file are specified as options in the procedure.

```

/* Issue the request. */
filename response 'c:\progs\work\response.xml' lrecl=1024;
proc metadata
  in=request
  out=response;
run;

```

The output of PROC METADATA is collected in the response file, which is included in the following code as an example of what the response looks like. The structure of the response is similar to the request, but with the response has attributes filled in and other object information provided.

```

<GetMetadataObjects>
  <ReposId>A0000001.A5ROJDT5</ReposId>
  <Type>Change</Type>
  <Objects>
    <Change Id="A5ROJDT5.AS0001JL" Name="Check in table changes Version 1.1"
      MetadataUpdated="18Jan2005:20:20:00" Desc="">
      <ChangeIdentity>
        <Person Id="A5KI3PIS.AN0000B5" Name="sasdemo" />
      </ChangeIdentity>
      <Objects>
        <Column Id="A5ROJDT5.AC0000RT" />
        <Column Id="A5ROJDT5.AC0000RU" />
        <Column Id="A5ROJDT5.AC0000RV" />
        <Column Id="A5ROJDT5.AC0000RW" />
        <Column Id="A5ROJDT5.AC0000RX" />
        <PhysicalTable Id="A5ROJDT5.AB0000RT" Name="RETAIL" />
      </Objects>
    </Change>
  </Objects>
</GetMetadataObjects>

```

```

        <Role Id="A5ROJDT5.AD0000RT" />
        <Property Id="A5ROJDT5.AF0000RT" />
        <Property Id="A5ROJDT5.AF0000RU" />
        <PropertySet Id="A5ROJDT5.AE0000RT" />
    </Objects>
</Change>
</Objects>
<ns>SAS</ns>
<Flags>260</Flags>
<Options>
    <Templates>
        <Change MetadataUpdated="" Desc="">
            <ChangeIdentity />
            <Objects />
        </Change>
        <PhysicalTable Name="" />
        <Person Name="" />
    </Templates>
</Options>
</GetMetadataObjects>

```

In the preceding results, you see that this repository has only a single checked-in object (the PhysicalTable RETAIL), and you see the related objects that were also checked in at the same time. Because you're really only interested in Tables for this example, you'll not propagate information about Properties, Columns, Roles, or other types of objects that could be included here. It is also worth noting that it is a learned skill to query and get returned only the metadata that is actually needed. This is important because the metadata server and receiving client program have more work to do to process extraneous metadata. Therefore, performance requirements suggest making the metadata queries as efficient as possible. This means that the metadata server doesn't have to look up the details of the extra objects, the results don't need to be transported over the network, and consuming applications don't need to parse a lot of XML content that's not needed.

The format of this information isn't completely useful yet, so you'll apply a map to re-style it into a format that can be directly read by the SAS XML engine. Here is the way you can do this.

```

/* Build the XML Map file to parse the response. */
filename map 'c:\progs\work\map.xml';
data _null_;
    file map;
    infile cards4;
    length long $256;
    input;
    long=_infile_;
    put long ' ';
    cards4;
<?xml version="1.0" ?>
    <SXLEMAP version="1.2">
        <TABLE name="Changes">
            <TABLE-PATH syntax="xpath">/GetMetadataObjects/Objects/Change</TABLE-PATH>

            <COLUMN name="Change" retain="YES">
                <PATH>/GetMetadataObjects/Objects/Change@Name</PATH>
                <TYPE>character</TYPE>
                <DATATYPE>STRING</DATATYPE>
                <LENGTH>60</LENGTH>
            </COLUMN>

            <COLUMN name="Time" retain="YES">
                <PATH>/GetMetadataObjects/Objects/Change@MetadataUpdated</PATH>
                <TYPE>date</TYPE>
                <DATATYPE>TIME</DATATYPE>
                <LENGTH>20</LENGTH>
            </COLUMN>

```

```

<COLUMN name="Description" retain="YES">
  <PATH>/GetMetadataObjects/Objects/Change@Desc</PATH>
  <TYPE>character</TYPE>
  <DATATYPE>STRING</DATATYPE>
  <LENGTH>60</LENGTH>
</COLUMN>

<COLUMN name="Who" retain="YES">
  <PATH>/GetMetadataObjects/Objects/Change/ChangeIdentity/Person@Name</PATH>
  <TYPE>character</TYPE>
  <DATATYPE>STRING</DATATYPE>
  <LENGTH>60</LENGTH>
</COLUMN>

<COLUMN name="Table" retain="YES">
  <PATH>/GetMetadataObjects/Objects/Change/Objects/PhysicalTable@Name</PATH>
  <TYPE>character</TYPE>
  <DATATYPE>STRING</DATATYPE>
  <LENGTH>60</LENGTH>
</COLUMN>
</TABLE>
</SXLEMAP>
;;;
run;

```

Again, this looks like a complex example, but there is extensive reference information available, so there is no need to worry about it. The main point is to look for XML paths in the response file and map it to a different structure. The resulting structure is a table with columns of information named in the XML in the preceding program. This is done through the XML LIBNAME engine; you'll copy the results from here to a SAS data set for later use or reporting.

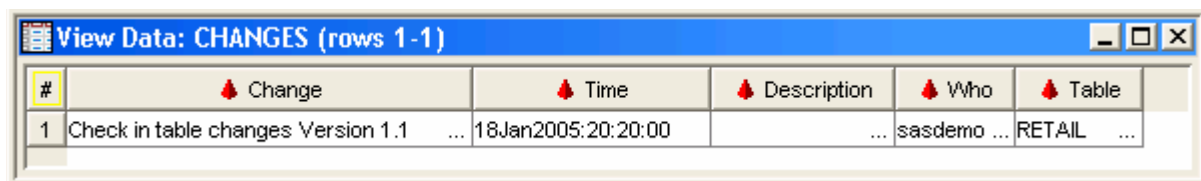
```

/* Parse the response with the XML library engine and PROC SQL. */
libname response xml xmlmap=map;

/* save the results to a SAS data set */
proc copy in=response out=work; select changes; run;

```

The results show what you were interested in finding out; who checked in which tables in the repository and when did they do it? As you can see in Figure 10, only a single table, RETAIL, has been checked into this repository. It shows the user who performed the update and when it occurred. The change information is included, but the person neglected to enter a full-text description of their activity.



#	Change	Time	Description	Who	Table
1	Check in table changes Version 1.1 ...	18Jan2005:20:20:00	...	sasdemo ...	RETAIL ...

Figure 10. Metadata Source Information Makes Its Way into a Warehouse Table

The example included in this section is an example of the type of metadata request and results processing that can be used in various areas of the metadata model to provide many kinds of metadata reporting capabilities that might be needed. The SAS product suite will continue to incorporate more of these types of abilities, but a Best Practice is to remain able to supplement what is available in products with this type of request. Developing additional support will require a study of the metadata model and various topics such as metadata templates and flags, but most things are handled very similarly in these requests and responses.

Bringing this together with the first point that's discussed in this paper—constructing a library of custom transformations to meet specific metadata reporting needs—is definitely a Best Practice.

OTHER PRACTICES

While not specifically Best Practices, this section includes some standard practices that are generally useful and have been shown to show significant benefit when using SAS ETL Server products.

Be aware of available documentation. As has been mentioned in several earlier places, there is a lot of useful documentation in the SAS online documentation, on the SAS Customer Support Web site, and in various documents that accompany the shipped product. Particularly useful are the various configuration and administration documents that are vital for you to be familiar with when deploying a SAS@9 application.

Take good care of your metadata. Be aware of standard utilities like %OMABAKUP, which can be used to perform a backup operation on a metadata server. It is important to remember that, although every effort is made to have reliable tools to construct and use metadata, it is very valuable to have the actual record of what has been done during an ETL project.

Understand and use the standard metadata promotion tools. SAS ETL Server works well in a Development/Testing/Production environment, and standard ways of performing these functions are available to metadata administrators. The incremental metadata movement topic described earlier (using metadata import/export operations), while helpful, do *not* replace a standard methodology for working with these three levels.

Think about the needed Load Technique. The Loader has a set of supported options that are appropriate for different types of storage. One mode, Truncate, works well for DBMS tables. Storage settings and constraints are retained, but all rows are deleted to create an empty table to which new rows are appended. Another mode, Drop, works well for SAS storage. It is an efficient operation for SAS data sets to delete them and create new ones to receive new table rows. Using these incorrectly can lead to complications: dropping a DBMS table isn't usually desirable because this can be a costly operation, and some constraints or other system values might not be created in the same way afterwards. For SAS data sets, truncation doesn't actually delete rows from the physical data set. Although the number of rows afterwards is shown as 0, the data set file itself doesn't return to an empty state on disk.

Remember case sensitivity for DBMS schemas. In some databases, schema names are case-sensitive. The author speaks from personal experience as to how difficult it can be to diagnose this problem when it occurs.

Use the Source Editor in SAS ETL Studio to validate code. Although it is possible to take code that has been developed in a SAS Display Manager session (DMS), there are some considerations about running code in a client/server configuration such as SAS ETL Studio. One notable consideration involves network drives when running on a Windows-based application server. In client/server configuration, the SAS application server doesn't run in the same way as a Windows console session. One difference is that mapped network drives (for example, a server drive mapped to a drive letter such as M:) probably won't be available to the application server session. Instead, a Universal Naming Convention (UNC) path should be used (such as \\servername\sharedvolumename) that can always be connected. A related topic is to understand whether a Windows domain user account or a local user account is being used. Some default login definitions for SAS@9 configuration default to local user accounts that probably don't have access to network drives, even if they are specified with UNC notation. Validating code in the SAS ETL Studio Source Editor, which also provides Log and Output windows, executes SAS programs on an application server and is a realistic way to validate that new code will work as expected.

Consider the size of process flows. While it might be tempting to create long logical flows in SAS ETL Studio, it's a good idea to think about the performance cost of doing this. In deployment through LSF Scheduler (The Load Scheduling Facility from Platform Computing) that's provided with SAS ETL Server, the scheduler can do a good job of running collections of processes in parallel. The more modular and numerous the component flows scheduled with LSF Scheduler, the better job it can do in efficiently running these process flows on available hardware.

CONCLUSION

This paper explains a number of ways to work better in enterprise data management under the heading of Best Practices. Although this is a wide-ranging discussion, all areas are subject to review to determine whether reusable practices will lead to generating more documented practices. The considerations that are presented in this paper are, by no means, a complete set but they provide a good starting point when new data management projects are undertaken.

It is important to remember that transformational standards can be achieved through the use of generated transforms, which can even include logic to meet metadata management and reporting needs. Reusability of external flat-file

descriptions leads to more reliable import processes. Incremental metadata migration can be used for a range of needs. The best way to really benefit from Best Practices is to start developing and sharing information about your own.

REFERENCES

Mehler, Gary, 2004, "Next Generation Warehousing with Version 9", *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference*, Orlando, FL,.

SAS Institute Inc. (2004), *SAS 9.1 Open Metadata Interface: User's Guide*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. SAS Publishing, SAS OnlineDoc. Available <http://support.sas.com/publishing/cdrom/index.html>.

RESOURCES

SAS Institute Inc. (2004), *SAS 9.1.3 Open Metadata Architecture: Best Practices Guide*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004), *SAS 9.1 Open Metadata Interface*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004), *SAS 9.1 Open Metadata Interface: Reference*, Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Gary Mehler
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000
E-mail: Gary.Mehler@SAS.com
www.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.