Metadata 101: A Beginner's Guide to Table-Driven Applications Programming

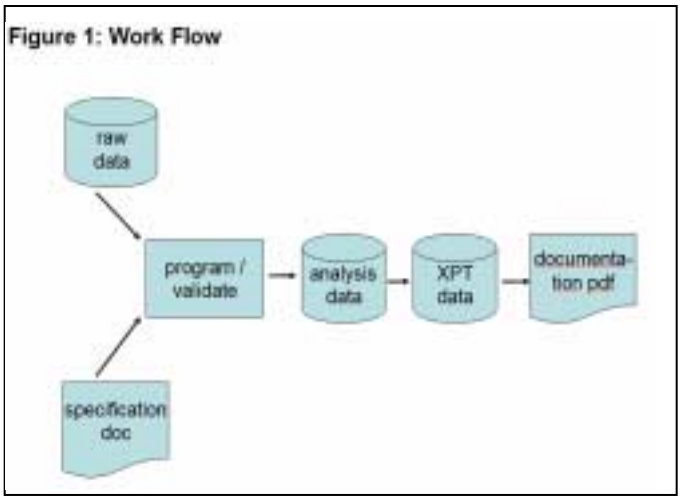
Frank DiIorio, CodeCrafters, Inc., Chapel Hill NC

Abstract. Any programmer dealing with frequent changes to program specifications is someone who has to cope well with all things frustrating and error-prone. Changes to report headers and footers, dataset contents, and other aspects of client deliverables have to be communicated effectively and implemented correctly. Standard operating procedures and good programming habits address these needs, but typically leave one thinking that there must be a better and easier way to create high-quality output.

Metadata and metadata-driven utilities are effective tools that reduce or entirely eliminate many of the programming and management problems inherent in traditional project work flows. This paper discusses the nature of metadata, some of its design criteria, and notes the need for the all-important applications that make metadata usable throughout the project life cycle. It also presents a simple case history, presenting traditional, "before" code and work flow followed by revised, metadata-driven coding. Readers should come away with an appreciation of the power of metadata-driven applications and, hopefully, ideas of how these techniques can be implemented in their workplace.

A Real World Scenario

Let's start by looking at a simplified real-world scenario. The data and output exist in a pharmaceutical environment, but the underlying issues are common to practically all of the workplaces that the SAS® System touches.



The scene depicted in **Figure 1** is deceptively simple. We read raw data for clinical trial subjects, then, using specifications written by a statistician, transform the data into a format suitable for analysis. Before sending the data to the FDA, we must convert it into an acceptable format (transport files or XML) and provide documentation showing how the datasets were created.

This appears to be straightforward, and generally speaking, it is. Most organizations undertaking this work have the workflow down to a science. Several parts of the process, however, can become problematic.

- **Entering Specifications.** In our scenario, statisticians instruct programmers on the derivation of new variables (those not taken directly from the data collection instruments). Traditionally, this has been handled using tools like spreadsheets and word processing documents. These tools are familiar and comfortable, but they come with a cost. Variable attributes such as type, length, and label must be manually transcribed by programmers. This is a tedious and error-prone process.
- **Change Management.** This being a real-world scenario, we need to remember that things change. A complex derivation can undergo numerous revisions before it is acceptable to the statistician. Even a simple specification may need revision just for the sake of standardization (e.g., using "mg/hg" in all variable labels instead of "MG/HG"). If the change affects many datasets, then many dataset creation programs are affected. The need for program changes must be communicated to the programmers, and the programmers must make the changes in all relevant programs. The possibility of incomplete or inaccurate change implementation is great, especially in a deadline-driven workplace.

These are just a few of the issues that surround creation of reliable products. It should be noted that they are present throughout the project life cycle. In other words, specification, change, and other issues dog the statistician and programmer from the collection of the data until it goes out the door to the FDA. It would be nice to have another way of handling the work flow.

"Enter the Metadata"

Not surprisingly, given the title of this paper, there is another way: using metadata and metadata tools to create application programs that *reference* values rather than *hard-code* them. This paper illustrates an important, powerful technique for moving code out of programs and into data stores that can be used to dynamically build both code and output. The technique has two underlying components: (1) creating metadata and (2) building tools to make the metadata readily available to applications throughout the project's life cycle.

The paper describes what metadata is, presents some key features of successful metadata implementation, and then revisits the scenario presented earlier. The reader should gain an understanding of what metadata is, how it can be deployed for an application, and what development effort is required.

Just What *Is* Metadata?

The classic definition of metadata is "data about data." That is, it is non-operational data that describes characteristics of data stores. You have probably already seen this in SAS – the `CONTENTS` procedure displays metadata from the dataset's header: variable name, length, type, position, format, and so on. Even at this basic level you can glean the importance of metadata: it describes data that is available to your program. *Without* it, you're left to guess that `AGEYR` represents a person's age in years. *With* it, you may discover from the variable label that age is rounded to the nearest whole year. Metadata is, in a word, essential.

The definition can be extended. Metadata is "data about data and the processes that support the creation of data and related output." Viewed this way, metadata can not only describe data stores and outputs, but also the environment in which they exist. With this expanded definition, we can envision metadata tables that describe directory structures, fonts and margins to use in reports, and the like. The key point here is that the more we move into metadata, the less hard-coding is required in individual programs. The enormous benefits of this paradigm shift will be discussed at length later in this paper.

How Do We Use It?

With these definitions in hand, let's review some of the lessons learned from the author's recent development efforts.

- **Metadata Systems Are "Living Documents."** As applications and user needs grow, so do the metadata. Rows, columns, and entire tables are modified as user needs change. It's important to realize that change does not mean that the initial design was short-sighted. Rather, it usually means that usage and acceptance by the user community requires expansion of the scope of applications supported by the metadata. In the ideal world, the impetus for change comes primarily from satisfied users who engage system developers in conversations beginning with "Wouldn't it be nice if …"
- **Tools are Critical.** You can have the best-designed metadata in the world, but if you don't provide tools for users to display and use it, you're shooting yourself in the foot. Like the metadata itself, the tools evolve as new applications arise and new uses for existing metadata are discovered. We'll see many examples of tools later in the paper. For now, just bear in mind that metadata and tools to access it go hand in hand.
- **You're Now a Software Developer.** Once you develop tools, you invite changes to the tools. This means defining releases of the tools, using versioning software, and approaching the coding process in a way that's likely different than the typical datasets, tables, figures and listings paradigm.
- **The Metadata Entry Interface Is Important.** The appearance and ease of use of the interface for entering metadata is critical. If it is amateurish, locks up unpredictably or loses data, user confidence erodes, usage declines, and all the downstream benefits of metadata are, effectively, lost.
- **Carefully Consider Data Storage Options.** It is natural to think of storing the metadata as SAS datasets. Readers of this paper are, after all, SAS-centric, and the tools manipulating the metadata will likely be SAS programs (SCL or macros) The author's experience is that Microsoft Access is a better platform. Forms and table views are easy to develop, and the data are, a few documented quirks aside, directly readable from SAS. Most importantly, MS Access easily supports multiple users.

How Do We Design It?

It has been said that effective use of metadata is "part fine art [technology and aesthetic/interface considerations] and part black art [programmer intuition as to the choice of tables and variables, the data representation, and so on]". The case study presented below discusses both aspects in some detail. Here, however, we can simply say that the need for and design of metadata relies heavily on the need for abstraction of programs and processes that produce deliverables of any form (datasets, statistical summaries, graphics, etc.) If something – setting of variable attributes, identification of a library location, etc. – is hard-coded in an application (especially if the hard-coding occurs repeatedly), it can at least be considered for movement into a metadata table.

So, these are the potential benefits and attendant costs of developing a metadata-driven system. Now let's see how it's actually done by revisiting the work flow described at the beginning of the paper.

Case Study, Part 1: The Status Quo and How It Might Change

Let's look at some of the programming tasks at different stages of the deliverables' life cycle represented in **Figure 1** (page 1). We'll look at traditional, non-metadata driven program fragments. We'll also identify what might be moved into one or more metadata tables.

Process 1 of 3: Create and Validate Datasets

To create and validate the datasets, we must, at a minimum, know the following about each variable:

- **Naming.** Dataset and variable identifiers
- **Attributes.** Type, format, label, length
- **Derivation.** This can simply be a copy of a variable in a dataset or a series of instructions for creation.

A traditional, non-metadata approach to the programming for a single variable may look something like:

```
attrib subjID length=$12 label="Subject Identifier" format=$12.;   [1]
usubjID = catt(study, '-', put(ID,z6.));   [2]
keep usubjID;    [3]
```

The variable attributes for `subjID` came from the .DOC, .XLS, or other file that contained the instructions for creating the variable. The programmer read the document, then coded the `ATTRIB` statement **[1]**; the assignment statement that created `usubjID` **[2]**; and the `KEEP` statement **[3]**. It's not even introductory rocket science, but it *is* a tedious process, especially when it is repeated for hundreds of variables.

Conversion to Metadata. What could be abstracted here? What could be generalized? What could be moved into metadata? A variable-level table (`VARIABLES`, shown in **Figure 2**, below) could have variables holding each of the attributes noted above (name, type, etc.).

Figure 2: Variable Metadata

variables : Table

	datasetName	variable	type	length	format	label	derivation
	AE	STUDY	C	5	$5.	Study Identifier	=raw.AE.study
	AE	USUBJID	C	12	$12.	Subject Identifier	Concatenate STUDY, a hyphen, and right-justified, 0-padded ID
3	AE	ID	N	8	6.	Subject ID	=raw.AE.id

We noted earlier that metadata without access tools is almost counterproductive. What would we want in the way of tools to make the metadata easily accessible to programmers?

- Create an `ATTRIB` statement
- Build `KEEP` lists

Recall the earlier characterization of metadata as a "living document." This first pass at identification of metadata tables, variables, and tools is likely to be expanded as the metadata users (statisticians and programmers) become comfortable with the tools and identify uses not considered during the initial sys-

tem design. This new functionality is not the same thing as the dread project "scope creep." Rather, it is an artifact of successful system implementation.

Process 2 of 3: CREATE TRANSPORT DATASETS

The next step in the workflow depicted in **Figure 1** (page 1) is the creation of files that are compliant with FDA guidelines. These are usually SAS Version 5 transport files, since they, unlike other formats of SAS datasets, are not in a proprietary format. As with the dataset creation step described above, the programming is fairly straightforward. Among the requirements set down by the FDA and the typical pharmaceutical client are:

- Arrange variables in a particular order: identifier and other key variables first, followed by other variables in alphabetical or another, user-specified order.
- Arrange the observations in a specific order.
- Use dataset labels.
- Exclude some variables. Not all of the variables may need to be sent to the FDA. Some filtering may be required.

A traditional, non-metadata approach to the programming for a dataset might look like:

```
libname xpt sasv5xpt "directory\ae.xpt";

proc sort data=clinical.ae out=tempAE;
     by study subjID AEseq;    [1]
run;

data xpt.ae(label="Adverse Events");  [2]
     retain study subjID AEseq ITT age race other_variables;    [3]
     set clinical.ae;
run;
```

Just as the dataset creation example was without mystery, so too is the code shown above: we create a LIBNAME, sort the dataset according to the specification document, then create the transport file using the prescribed dataset label and variable order. Also just as before, the process is clumsy. Consider the tedium involved in coding the RETAIN statement and making changes to it should the variables need rearranging.

Conversion to Metadata. What could be moved into metadata? Four items, identified above by numbers in brackets, come to mind:

- The sort sequence (item **[1]** above)
- Dataset label **[2]**
- Variable order **[3]**
- Which variables to include **[3]**

The first two items are intrinsically different from the last two. They contain information that applies to the dataset as a whole rather than individual variables within the dataset. This, in turn, means we need a new metadata dataset – DATASETS – where observations represent individual datasets. (Remember the "living documents" characterization; there is no point to forcing dataset-level information into a variable-level table. As the need for metadata grows, so does the number of variables and tables.) Variables in DATASETS will identify dataset name and sort keys.

Two additions to the variable-level metadata are also required: a sequence variable to identify a variable's position within a dataset and a check box to flag variables to include in the transport file. The metadata now looks like **Figure 3** (next page).

Also note that new metadata begets new tools. Examination of the program suggests several tools that would make the programmer's life simpler:

- Sort a dataset by the variables identified in the sort keys field in the DATASETS table.
- Create a macro variable holding the dataset label read from the DATASETS table.

- Create a list of variables to insert in the RETAIN statement. It would use the sequence variable from the VARIABLES table, and would only include variables marked for inclusion. Variables with missing order values would be arranged alphabetically.

Process 3 of 3: DOCUMENTATION

No one wants to be handed a CD filled with datasets and no documentation of how they were created. The FDA requires documentation in a specific layout, stored either as a PDF or XML file. A minimal subset of documentation is required for each. Screen shots from a PDF are shown in **Figure 4** (next page).

Recall the discussion format for the previous steps in the work flow: we began by presenting a traditional, non-metdata approach to the individual task. Here, however, it becomes difficult to think of how the documentation challenge could be met without some form of well-ordered, programmatically accessible data. The alternative is manually transcribing or cutting and pasting descriptions of hundreds of variables into a word processing document. The creation is, to be sure, a nightmare.

Equally distressing is the prospect of change: modifications to variable labels, definitions, and the like must trickle down through the dataset creation and transport file programs *and* must be reflected in the documentation file. Clearly, having the documentation components in a format accessible to a program would be vastly superior to an otherwise intensively manual process.

Conversion to Metadata. As we examine **Figure 4**, we see that we already have most of what we need. We only need to add dataset description to the DATASETS table and codes (e.g., `1 = 'Male', 2 = 'Female'`) to the VARIABLES metadata.

The major piece of implementaion here, of course, is creating the tool that processes the metadata and creates the properly formatted and hyperlinked PDF. (The design and implementation intricacies of this "black box" are beyond the scope of this paper.)

RECAP

The section above has presented traditional, non-metadata based programs and has identified parts of the programs that could be moved into metadata. It has also suggested what some of the all-important metadata tools might look like. Let's summarize the contents of the metadata tables and the functionality and benefits of the tools.

Metadata. We described two tables, DATASETS and VARIABLES:

DATASETS

Variable	Description
datasetName	Name of the dataset (8-character maximum)
label	Dataset label (40-character maximum)
description	Description of dataset contents and granularity
sortKeys	Sort order (blank-delimited list)

VARIABLES

Variable	Description
datasetName	Name of the dataset (8-character maximum)
name	Variable name (8-character maximum)
type	Variable type (C or N)
length	Variable length (maximum of 200 for character variables)
format	SAS System format (user-defined formats are not allowed)
label	Variable label (40-character maximum)
derivation	Description of where the variable was copied from or, if derived, how it was created
order	Sequence in the transport dataset. If missing, the variable will be arranged alphabetically
include	Check if the variable is to be included in the transport file
codes	Mapping of discrete values

Figure 4: Documentation

Dataset-Level

Statistical Analysis Datasets for [redacted] 1

For Internal Documentation. See ADaMIG.PDF

Dataset	Description of Dataset	Location
AE	Adverse events	CRT Datasets\[redacted]\Analysis\ae
ANAPOP	Analysis populations	CRT Datasets\[redacted]\Analysis\anapop
CONMED	Concomitant medications	CRT Datasets\[redacted]\Analysis\conmed
DEMOG	Demographics	CRT Datasets\[redacted]\Analysis\demog
DISPOS	Patient disposition	CRT Datasets\[redacted]\Analysis\dispos
ECG	ECG results	CRT Datasets\[redacted]\Analysis\ecg
EFFICACY	Efficacy parameters	CRT Datasets\[redacted]\Analysis\efficacy
ENTRY	Study entry information	CRT Datasets\[redacted]\Analysis\entry
EXPOSURE	Exposure to study drug	CRT Datasets\[redacted]\Analysis\exposure
ISITEINV	Injection site evaluated by investigator	CRT Datasets\[redacted]\Analysis\isiteinv
ISITEPAT	Injection site evaluated by patient	CRT Datasets\[redacted]\Analysis\isitepat
LAB1	Laboratory results	CRT Datasets\[redacted]\Analysis\lab1
LAB2	Laboratory results	CRT Datasets\[redacted]\Analysis\lab2

Variable-Level

[redacted]: Variables for Table AE 5

Description: Adverse events
Structure: One record per adverse event

Return to TOC

Variable	Label	Type	Codes	Comments
AERES_DD	Date of recovery (Day)	num		=RAW.AE_CODED.AECE_DD
AERES_DM	Date of recovery (Month)	num		=RAW.AE_CODED.AECE_DM
AERES_DT	Date and time of recovery	num		=dhms(DERIVE.AE.AERES_D, hour(DERIVE.AE.AERES_T), minute(DERIVE.AE.AERES_T), 0) if DERIVE.AE.AERES_D and DERIVE.AE.AERES_T not missing
AERES_DY	Date of recovery (Year)	num		=RAW.AE_CODED.AECE_DY
AERES_T	Time of recovery	num		=RAW.AE_CODED.AECE_T
AESER	Serious	num		=RAW.AE_CODED.AESER
AESER_	Serious Label	char		='No' if DERIVE.AE.AESER=0; ='Yes' if DERIVE.AE.AESER=1
AEST_DD	Date of onset (Day)	num		=RAW.AE_CODED.AEST_DD
AEST_DM	Date of onset (Month)	num		=RAW.AE_CODED.AEST_DM
AEST_DT	Date and time of onset	num		=dhms(DERIVE.AE.AEST_D, hour(DERIVE.AE.AEST_T), minute(DERIVE.AE.AEST_T), 0) if DERIVE.AE.AEST_D and DERIVE.AE.AEST_T not missing
AEST_DY	Date of onset (Year)	num		=RAW.AE_CODED.AEST_DY
AEST_T	Time of onset	num		=RAW.AE_CODED.AEST_T
AGE	Age (years)	num		=DERIVE.DEMOG.AGE
BMI	BMI (kg/m/m)	num		=DERIVE.DEMOG.BMI

Tools. Recall, yet again, that *without tools the metadata are close to useless*. Notice the tool type – the tools that access the metadata are implemented as SAS macros. Some database coding (Microsoft Access Visual Basic, for example) may also be required or helpful. The tools are displayed by work flow process. Note, however, that there may be other tools that could be used in more than one process.

Process 1: Create Datasets

Macro Name	Metadata Used	Description / Benefits
`%buildAttrib`	`variables`	Create a macro variable holding `ATTRIBUTE` statements variables in a dataset. Programmer is relieved of manual coding and is guaranteed to receive most recent attribute values.
`%buildKeep`	`variables`	Create a macro variable holding a list of variables suitable for inclusion in a `KEEP` statement or `KEEP` dataset option. Programmer is guaranteed that the variable list is consistent with the metadata.

Process 2: Create Transport Files

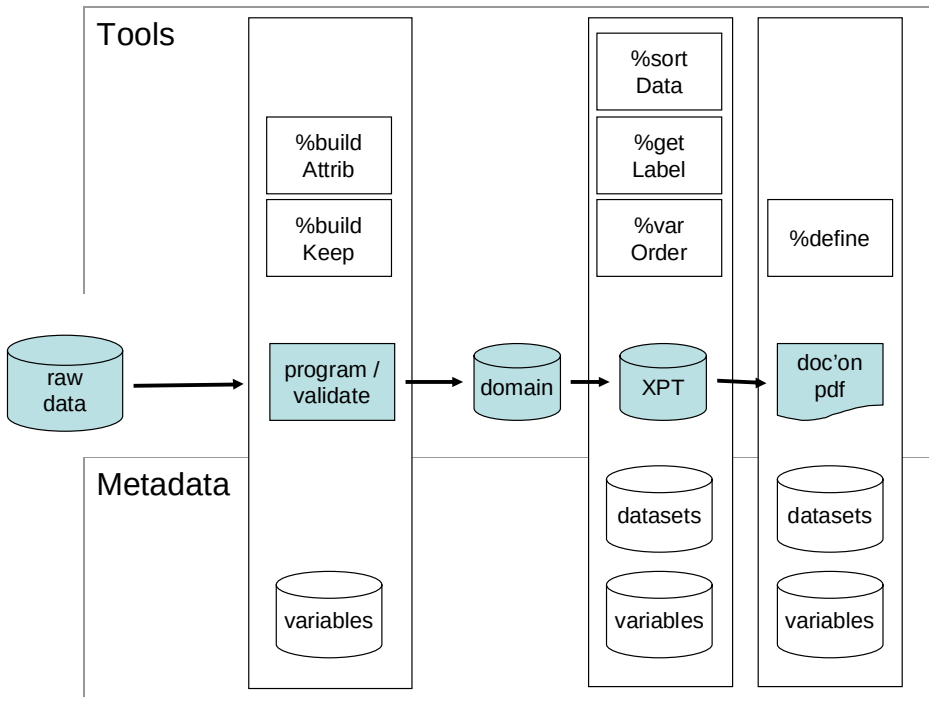
Macro Name	Metadata Used	Description / Benefits
`%sortData`	`datasets`	Sort a dataset by the sort keys specified in the `SORTKEYS` variable. Programmer does not have to copy variable names from specification document.
`%getLabel`	`datasets`	Place the label for a dataset in a macro variable. Programmer does not have to copy the dataset label from specification document.
`%varOrder`	`variables`	Create a macro variable containing variables in a dataset in the order specified by the `ORDER` variable. Only variables for which `INCLUDE` was checked are included in the list. Programmer is relieved of coding the `RETAIN` statement, and is guaranteed that the list will accurately reflect current metadata.

Process 3: Documentation

Macro Name	Metadata Used	Description / Benefits
`%define`	`datasets` `variables`	Reads the metadata tables and creates the PDF containing the documention. Combining of dataset and variable-level metadata is handled automatically, as are formatting issues required for FDA compliance.

These metadata tables and tools are displayed in **Figure 5** (next page). Based on **Figure 1**, it overlays the tables and tools over each phase of the work flow. We see that metadata usage is found throughout the process. The distaff view might be that this is a clutterered version of **Figure 1**, inserting extra layers of code and data on a process that did not really seem that broken. The benefits of metadata-oriented design become tangible when we see how the programming changes. This is addressed in the next section.

Figure 5: Work Flow, Using Metadata and Tools



Case Study, Part 2: Redesign Using Metadata

Before discussing the revised metadata-driven programs, we need to state some assumptions and caveats. First, realize that the program fragments are just that, pieces of programs and not complete applications. Second, we assume that the data libraries, macro libraries and other options have been established prior to the program excerpts (indeed, this is something that could be accomplished with different metadata and macros). Third, we assume the macros are production-quality. This means that they have been validated and implies that they thoroughly check parameters, issue messages to the SAS Log, and have the other hallmarks of high-quality tools. With these notes in mind, let's use the metadata and tools described on pages 5 and 7 and revisit the programs from Part 1.

Dataset Creation and Validation. The following program fragment demonstrates several key concepts surrounding the use of metadata and metadata access tools.

```
%buildAttrib(data=ae)   /* Creates &attrib */     [1]
%buildKeep(data=ae)     /* Creates &keepList */   [2]

data clinical.ae;
     set raw.dmAE;
     &attrib.                [3]
     assignmentStatements
     keep &keepList.;        [4]
run;
```

We call macros `buildAttrib` **[1]** and `buildKeep` **[2]**. Then, in the `DATA` step, we use the macro variables that the macros created. `ATTRIB` **[3]** is a complete `ATTRIBUTE` statement, while `KEEPLIST` **[4]** is simply a list of variables, and so must be enclosed in a `KEEP` statement.

Note that the programmer does *not* have to know the structure (granularity, variable naming) of the metadata – this is handled by %buildAttrib. What the programmer *does* have to know is how to use the macro; that is, he/she must know what is expected as input and what is created as output. In this example, we have to specify the DATA parameter and we must know that the global macro variable ATTRIB is the macro's output. All that is required of the programmer is to read the macro's documentation.

Consider, too, some of the other benefits of the metadata-driven approach. If the metadata changes, the tools pick up the changes and present the modifications to the program. *The program itself does not have to change.* If the project statistician modifies variable labels for 15 variables and unchecks the inclusion box for several others, the program statements for attribute declarations (&attrib) and KEEP statement coding (&keepList) are unchanged.

Finally, note the benefits of using the macros from a system and validation perspective. The tools will be validated before being used by the programmers. The tools are generalized, and can be used by any project that uses properly formatted variable-level metadata. If properly constructed, the tools will also perform checks, issue messages, and in general be more thorough than if similar functionality were embedded within an individual study's program.

Transport Files. Similar code reduction and reliability improvements are gained when creating transport files.

```
libname xpt sasv5xpt "directory\ae.xpt";

%sortData(in=clinical.ae, out=tempAE)  [1]

%getLabel(data=ae)     [2]
%varOrder(data=ae)     [3]
data xpt.ae(label="&datasetLabel.");  [4]
     retain &varList.   [5]
     set clinical.ae;
run;
```

We call %sortData **[1]**, creating dataset tempAE. Calls to %getLabel **[2]** and %varOrder **[3]** create global macro variables that will be used in the DATA **[4]** and RETAIN **[5]** statements. The benefits of using metadata and metadata access tools are clear: the macros read the appropriate metadata tables and return datasets and macro variables. All the programmer needs to do is read the documentation and know the macro parameters and outputs. As with the dataset creation example, if the metadata changes, the program does not need to change; changes to variable inclusion and order are picked up automatically. Transcription of .DOC file instructions happily becomes a thing of the past.

Documentation. The process could be as simple as a macro call:

```
%doc(out=define.PDF)
```

Everything the macro needs to know regarding content and layout is found in the metadata and the FDA guidelines. The %doc macro reads the metadata and creates a compliant PDF. Along the way, it draws heavily not only on metadata-related macros but also on general-purpose utilities that greatly reduce the amount of code in the macro.

Case Study, Part 3: Coping with Change

Any system designed for general use, particularly one servicing an environment characterized by rapidly changing needs, must be extensible. That is, its structure must be amenable to growth and modification without affecting existing functionality and methods of usage. In this part of the case study, we trace the impact of a change to the metadata. Focus here less on *what* is being changed and more on *how many* changes are required. This section may be considered a discussion of entry cost into the metadata-oriented world.

The Change. Recall the purpose of the derivation variable in the VARIABLES table: it contains programming instructions in the form of a narrative rather than actual code. Suppose the manipulation is sufficiently straightforward that actual code could be entered. We would still want the narrative (for documentation purposes), so the derivation variable should not be changed. An additional field – CALCULATION – could be added to the metadata. It would either be blank or contain the complete programming statement(s) to create the variable. In our earlier example, the definition of SUBJID was:

```
Concatenate STUDY, a hyphen, and right-justified, 0-padded ID
```

By contrast, the entry in CALCULATION is the complete assignment statement::

```
subjID = catt(study, '-', put(ID,z6.));
```

Multiple statements could also be entered:

```
if ^missing(ID) then subjID = catt(study, '-', put(ID,z6.));
   else do;
        putLog "Missing ID: " _n_;
        subjID = catt(study, '-??????');
        end;
```

The Implementation. The advantages of moving code into the metadata are obvious. What is less clear is how to implement the change. Metadata-driven applications have more "moving parts" than traditional programming. Each part needs to be altered and tested before the addition is ready to use. We list them here to bring attention to the inherent complexity of this style of programming.

- **Metadata Changes.** The VARIABLES table needs to be altered. Queries that refer to the table may also need modification.
- **Form Changes.** If the metadata are entered with forms, the CALCULATION field must be added to each form.
- **Changes to Generic Tools.** If there is a standard tool that produces HTML or a PDF of dataset and variable specifications, it must be modified to add CALCULATION to the output. Since not all projects' metadata may have the new field, the program must determine whether the variable exists in a particular table and act accordingly. Otherwise, assuming that CALCULATION exists in all VARIABLES tables will generate an error when processing earlier studies.
- **New tools.** A new tool – %getCalcs – has to be developed to fully exploit the benefits of the calculation field. It would accept a dataset name as a parameter and create a global macro variable containing all the non-blank values of CALCULATION for the dataset. The macro must be validated, documented, and its availability made known to the programmers.
- **Additional Application Coding.** Programs creating datasets call the macro by and use its outputs.

The program that we saw in the second part of the case study can be rewritten to take advantage of the CALCULATION variable and %getCalcs (changes shown in boldface):

```
%buildAttrib(data=ae)
%buildKeep(data=ae)
%getCalcs(data=ae)

data clinical.ae;
     set raw.dmAE;
     &attrib.
     &calcs.    /* Created by %getCalcs */
     other assignment statements
     keep &keepList.;
run;
```

All of the routine transformations can now be moved into the metadata. The only assignment statements that actually have to be written by the programmers are those that are complex, have timing dependencies, or require passes through multiple datasets. Application programs become smaller thanks to the use of standardized, validated tools.

Conclusion

Metadata-oriented systems enjoy a number of advantages over traditional programming methods:

- **Data are single-entry, mutliple use.** Dataset and variable characteristics and other metadata are held in a database format that can be easily accessed programmatically. Rather than specifying values in a word processing document and requiring the programmer to copy this information to potentially many program locations, values stored as data need to be entered only once. Access tools allow the data to be used any number of times without the possibility of incorrect transcription from the specification source.
- **Modifications propagate correctly.** If the access tools are used, modifications to a metadata value will be picked up automatically in all appropriate locations. For example, a change to a variable label will be propagated during dataset creation (by the `%buildAttrib` macro) and when writing the documentation PDF (by the `%doc` macro). Consistent use of the metadata access tools ensures that a change that needs to occur "n" times does, indeed, occur "n" times. This cannot always be said of a traditional or even partly automated programming process.
- **Manual processes are greatly reduced.** Building KEEP lists and other labor-intensive tasks is greatly simplified. Lists and entire statements are replaced by a single, *unchanging* reference to a macro variable that accurately reflects the current contents of the metadata.
- **Production programs become more reliable.** The size of programs decreases due to the use of validated tools. This makes intial programming, documentation, validation, and maintenance more straightforward.
- **Happier staff.** Data is entered using a friendly interface. Well-documented and validated tools relieve programmers of tedious, error-prone coding. Output is produced more quickly thanks to the tools. Interface and tool developers enjoy interesting, challenging work. It's a win all around.

While these and other advantages will create enthusiasm for building metadata and metadata tools, you should also be aware of the startup and ongoing costs that they impose:

- **Interface development.** Building an attractive, functional front end for metadata entry can be time-consuming. It also requires a skill set that may not be currently available in a programming department's collective skill set (MS Access, Visual Basic, interface design). Once built, however, maintenance-level operations such as adding fields and modifying queries has relatively small cost.
- **Tool development.** Designing and coding tools to make the metadata readily accessible to programmers can be costly. Here, as in interface development, however, if the tools are properly designed, they can be modified to accommodate new variables, new options, etc. relatively easily.
- **Training.** The metadata-driven paradigm is new to some programmers. Training (some would say proselytizing) is vital. The programmer must be convinced that current processes, often as comfortable as they are tedious, can be effectively replaced. Likewise, statisticians and others entering metadata must be trained in the use of the database interface.

This paper has presented only a small fraction of the possible uses of metadata and metadata tools. Tables and figures have components that can readily be abstracted into metadata. Even system-level items such as library locations, macro and format autocall paths, and system options can be represented as metadata. The endpoint for these and all metadata-oriented applications is reliable, easily accessible retrieval of data and process characteristics throughout a project's life cycle.

Comments

Visit www.CodeCraftersInc.com for other papers on this topic.

Your comments and questions are welcomed and encouraged. Contact the author at Frank@CodeCraftersInc.com.