Paper 105-2007

The Electronic Project: Effectively Using Metadata Throughout the Project Life Cycle

Frank DiIorio, CodeCrafters, Inc., Chapel Hill NC
Jeff Abolafia, Rho, Inc., Chapel Hill NC

INTRODUCTION

The rapid, reliable, and cost-effective production of FDA deliverables is the Holy Grail of pharmaceutical companies and Contract Research Organizations (CROs). Difficult to achieve in a "calm" environment, they become even more problematic as statisticians, programmers, and project managers edge closer to the maelstrom that is the submission date. The potential for miscommunication within the project team increases, seemingly exponentially, thus making the need for robust programming tools and practices critical.

An earlier paper by the authors (see "References," below) stated that well-constructed metadata and metadata access tools can significantly improve the creation of the datasets and documents that comprise an electronic submission. A year later, we even more emphatically believe this to be the case: metadata-driven applications and utilities can be integrated into standard business processes, speeding the production and improving the quality of deliverables.

The previous paper was heavy on theory and light on actual examples. We emphasized the benefits of metadata-driven processes over their earlier, mostly manual, counterparts. This paper switches the balance and describes the metadata tables and applications in detail. We demonstrate the ubiquity of the metadata by following its uses in the life cycle of one part of a submission package, namely, creation of industry standard (CDISC SDTM) compliant Domain datasets and their accompanying documentation, `define.XML`.

The presentation is organized into four sections:

- A summary of the key features of metadata-driven applications;
- A description of key metadata tables and content currently used at Rho, Inc.;
- A detailed discussion describing the SDTM process flow and identifies how metadata and metadata access tools are utilized at each point; and
- A summary of key points and an outline of ideas we're entertaining for further improving the scope and access to the metadata.

KEY FEATURES OF A METADATA-DRIVEN SYSTEM

Metadata is, classically, "data about data." That is, it is data stored in a programmatically accessible manner, which contains characteristics of data stores. We expand the definition here so that it describes not only data but aspects of the work environment that uses the data. That is, we create metadata that describes the work flow that surrounds the data. The broader the definition, the wider the range of applications that can be automated.

Here are the key aspects of metadata usage, as described in the earlier paper and supplemented somewhat by our experience since its publication.

- **Live by the Metadata, Die by the Metadata.** Throughout the life cycle of a project, it's key that the people entering and reviewing the metadata are well-trained (in terms of how to use the tools, what values are valid, etc.).
- **Chose Storage Carefully.** After considerable experimentation with SAS and other formats, we chose Microsoft Access as the database format for the metadata. It is a solid, reliable product that handles mutliple users without complaint and is easily accessible by the all-important SAS applications that are used throughout the project life cycle.
- **Develop Tools.** The metadata organization, while not overly complex, should be presented to its users in a straightforward, well-documented manner. If the programmers chose to use the tables directly, they can do so, but they should also have utilities at their disposal that will reliably deliver information from the tables.
- **Expect (and Welcome) Change.** The amount of metadata tables and columns must be able to grow as the range of its uses grows. This, in turn, usually means growth of the tool set that accesses the metadata: existing tools are enhanced, and entirely new ones are created.
- **Consider the Interface.** The metadata must be easily entered. The interface must be easy to navigate, should be populated programmatically when possible, and drop-down boxes should be provided when appropriate to preempt entry of inappropriate values.
- **Develop Complementary, Non-Metadata Tools.** The development time of metadata tools is greatly shortened if a library of generalized, validated utilities is available.
- **Utilize Complementary Metadata.** The SAS dictionary tables and views are an invaluable adjunct to the project-specific metadata discussed in this paper. Familiarity with these resources is essential for developing robust metadata applications (see "References," below, for examples).

- **Create Uniformity Across Projects.** Initial success created by metadata tools encourages their adoption in subsequent projects. Standardization of library names and directory structures greatly reduces, and often completely eliminates, the need for project-specific customizing.
- **Provide Documentation and Training.** The best tools in the world are of no use if they are poorly documented and not publicized. Programmer and end-user documentation must be in place for both the metadata and the tools that present the metadata to users.

CURRENT METADATA ARCHITECTURE

Figure 1, below, presents a high-level view of a project's work flow. Many tasks need to be coordinated: data must be manually or electronically captured from the CRF; the statistical analysis plan must be reviewed to determine the analysis datasets' contents; and Domain (SDTM) and analysis datasets must be programmed, validated, and prepared for submission to the FDA.

Figure 1

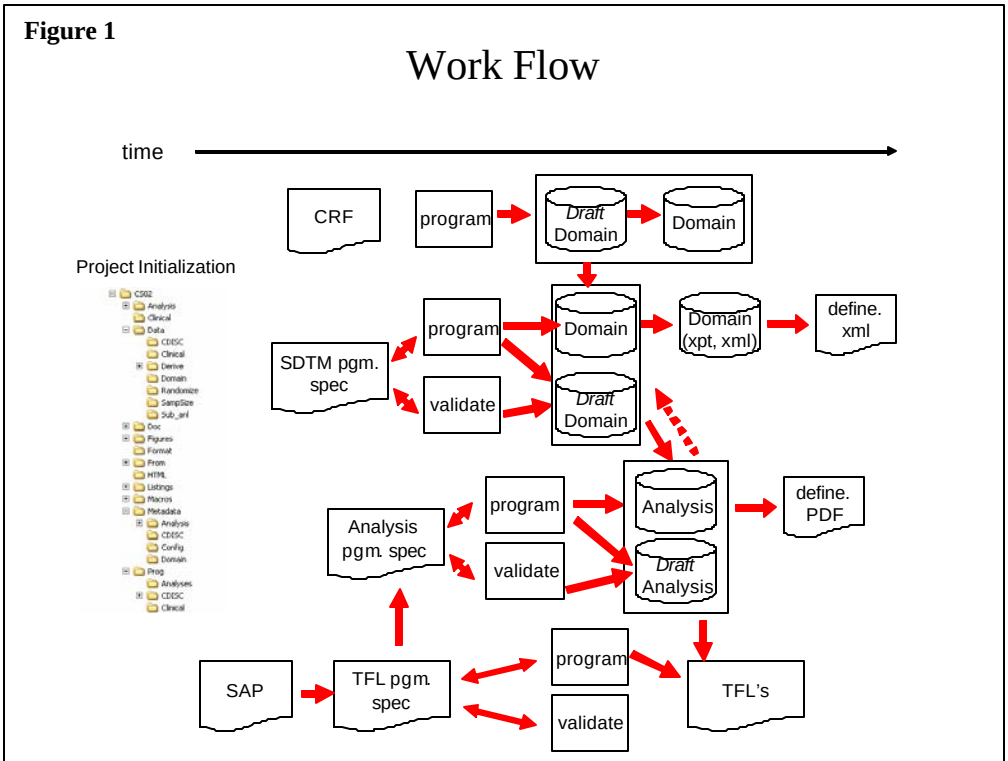


Figure 2

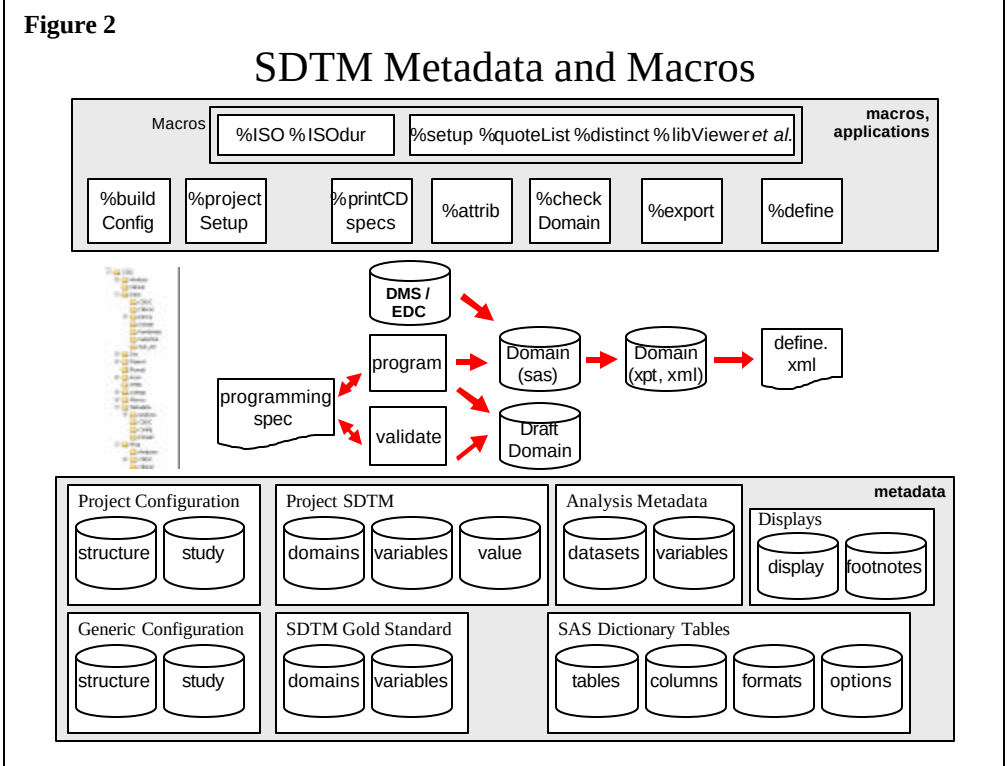


Figure 2 introduces two critical features into the work flow: metadata and metadata access tools. Due to space considerations, it focuses only on SDTM, but several points are made here that are applicable to any process described in **Figure 1**.

First, there is a considerable amount of logically separated metadata. Collectively, project configuration, SDTM and Analysis dataset, and Display (TFL) description metadata almost completely describe every piece of a project's deliverables. Also notice the inclusion of SAS Dictionary Tables. They are included here because they are used extensively by metadata access tools.

The second key feature of **Figure 2** is the macros and applications that appear at the top of the figure. Without tools that handle the metadata in a reliable and user-friendly fashion, the metadata would be rightly seen as a burden rather than as an asset. These tools, mostly macros, perform such tasks as creating fragments of `ATTRIB` and `KEEP` statements, displaying metadata contents in a PDF, and producing deliverables such as SAS transport files and define.XML. They will be discussed at length in

"Utilization in the SDTM Life Cycle," below.

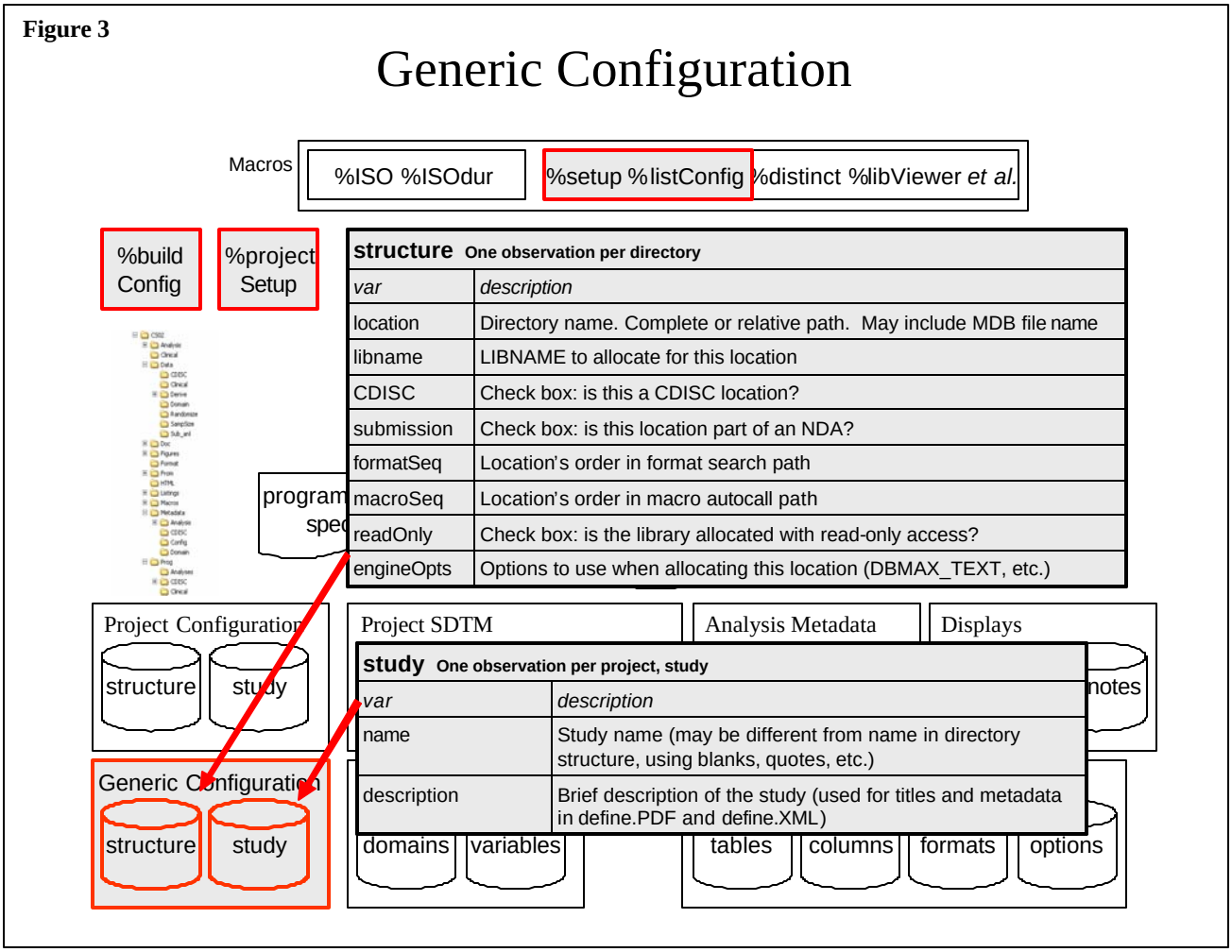
With out "Expect Change" caveat firmly in mind, here are some of the metadata databases that currently drive the systems at Rho. This section will give you a reasonably detailed view of the organization and contents of each database and most of the tables held in them. It is *not* our intent to get into the unnecessarily low-level details such as data types, lengths, keys, and the like.

CONFIGURATION (Generic)

Figure 3 describes the generic configuration metadata. It is a template, a generic version of a project's structure. It is, essentially, a description of the recommended directory structure as described in SOPs. For directories containing data stores that will be accessed by SAS, the `Structure` table also contains recommended LIBNAMEs and engine options. See "CONFIGURATION (Project)," below, for more details.

Tools Using the Metadata. `%buildConfig` copies the generic configuration metadata to a new study. Then, after optional modification of the tables, `%projectSetup` creates the directory structure and configures programs to run various utilities. These macros are demonstrated in "Utilization in the SDTM Life Cycle," below.

Figure 3

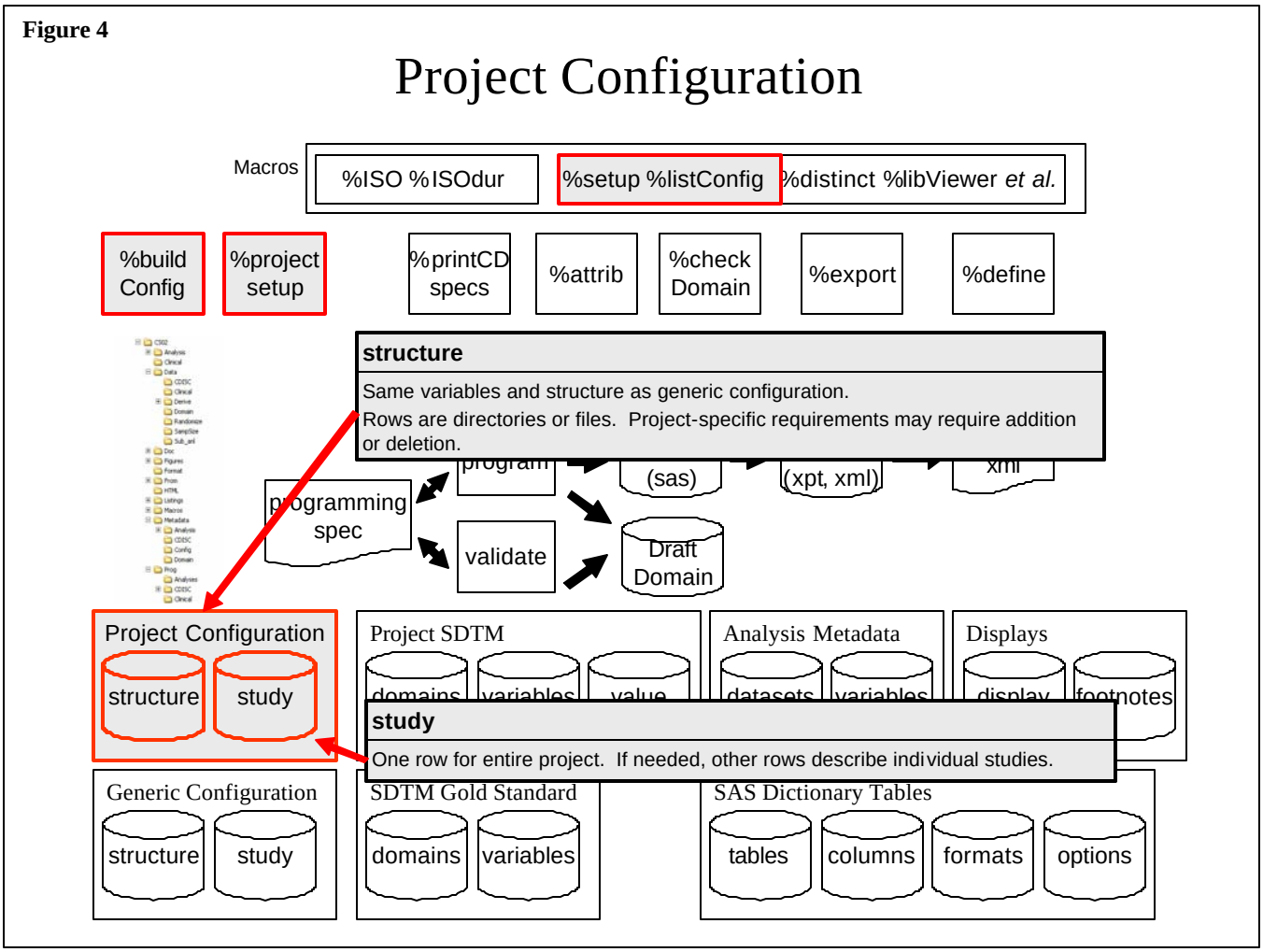


CONFIGURATION (Project)

This database, shown in **Figure 4** (next page), describes the directory structure and other key features of a project. During project setup, the generic configuration file is copied to a project-specific directory. The file may be customized to accommodate specific project needs (additional autocall or format libraries, data directories, etc.). Whatever the level of customization, the only hard and fast requirement is that the `Structure` table correctly identifies the project's root directory, since all other directories for the project are entered as relative paths.

Tools Using the Metadata. `%setup` is run at the beginning of every program in the system. Using the `Structure` table, it creates LIBNAMES, macro variables, and autocall and format search paths. Its function and output are described in detail in "Utilization in the SDTM Life Cycle," below.

Figure 4



`%listConfig` is a simple utility that creates a PDF displaying the configuration database.

SDTM (Generic, "Gold Standard")

Figure 5 (next page) describes the metadata containing the CDISC standards for SDTM domains. Collectively, these tables identify the variables needed in each domain, their data type, descriptive label, permitted values, and the like. They do not, however, contain a complete description of how to create the domains, nor do they have what an FDA reviewer requires to understand how the fields were derived. This is handled by the project-specific SDTM metadata, described below.

Tools Using the Metadata. The `%projectSetup` macro, described above, copies the Gold Standard SDTM MDB to the project.

`%printGoldstd` displays all or selected Domain information in a PDF.

SDTM (Project)

Just as the generic configuration database is customized for a project, so too is the SDTM "Gold Standard" database (**Figure 6**, next page). It contains the fields from the Gold Standard as well as fields needed for programming and other purposes. Note that some of the Gold Standard fields (Domain and variable names, label) are *not* editable for standard, predefined Domains (`AE`, `CM`, etc.) but *are* subject to change for user-defined Domains.

Many of the SDTM domains are "vertical" representations of data that is often collected in a more "horizontal" manner. For example, a traditional Vital Signs dataset may have separate variables for weight, supine systolic blood pressure, and sitting systolic blood pressure. The SDTM data is a more relational design. Rather than repeating test name, value, and so on, making the dataset "wide," the SDTM domains use variables that identify the name, value, unit, and other characteristics of a particular measurement, thus making the Domain "tall" and "skinny" (more observations, fewer variables). This verticality introduces the need for value-level metadata, in this case that describes each possible value of `VSTESTCD`. Thus we create the `VALUE` table, shown in **Figure 6**.

Figure 5

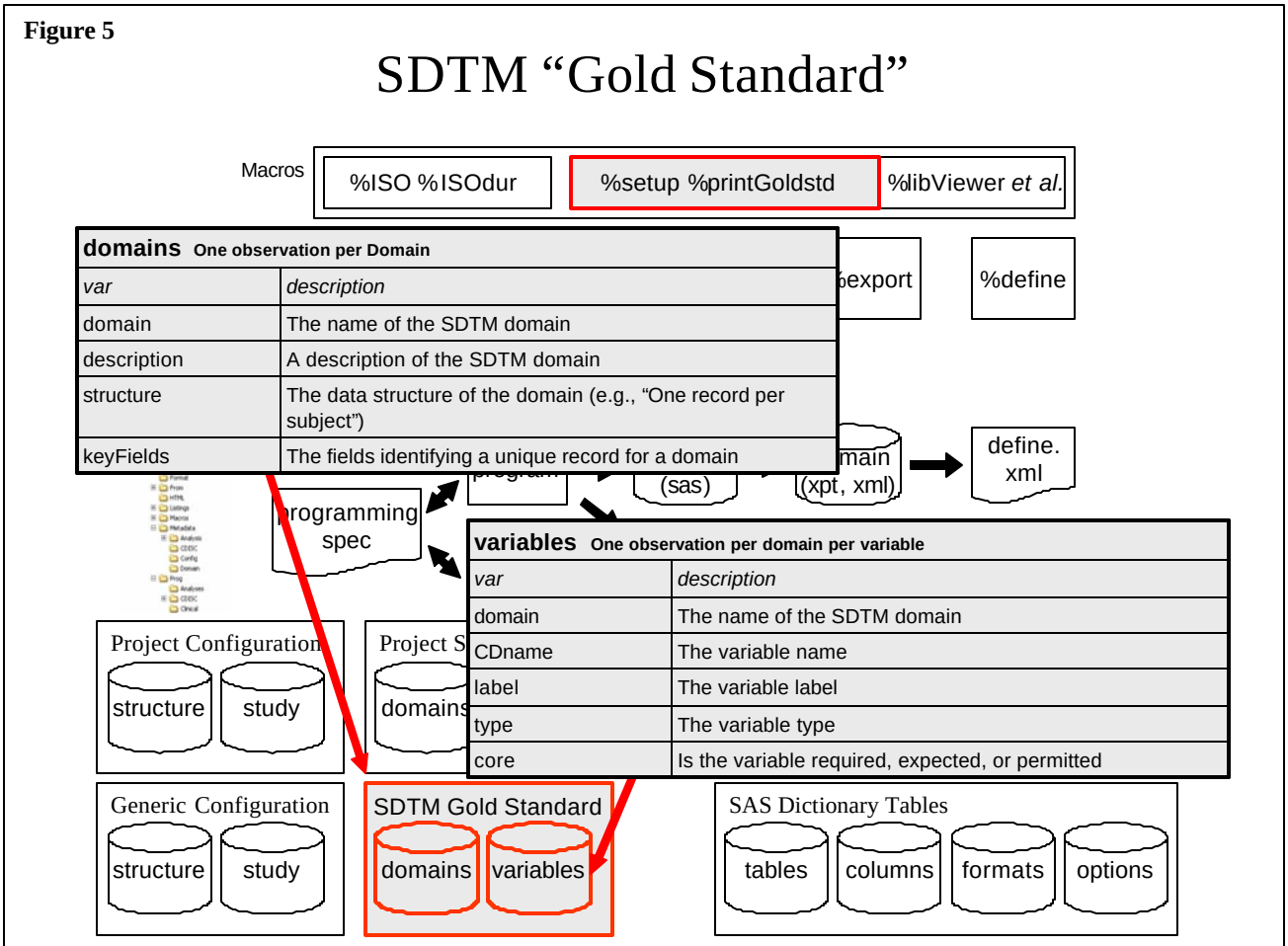
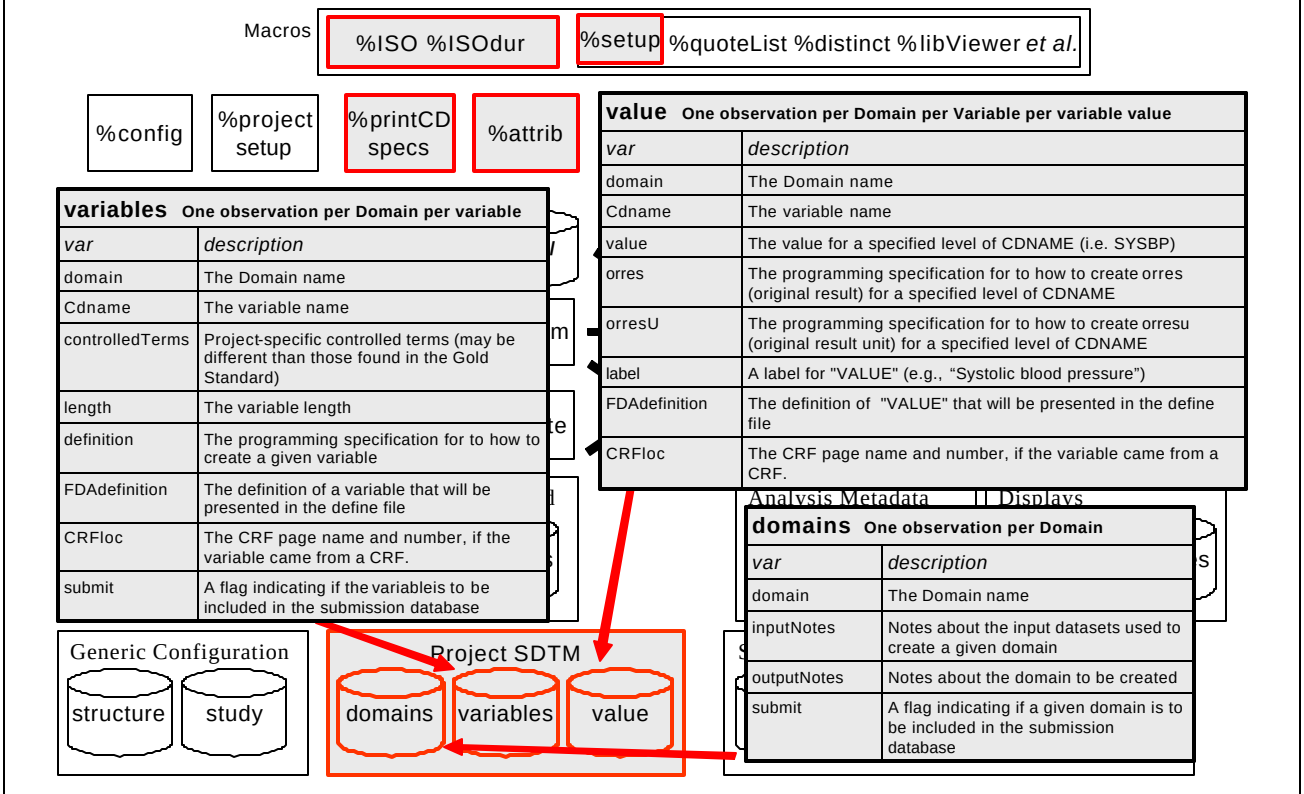


Figure 6



Pre-SDTM domains were often non-relational. The SDTM/relational approach for Vital Signs data, for example, looks like:

CDname	VStestcd	orres	orresu
VSTESTCD	Systolic BP	90	mg/hg
VSTESTCD	Diastolic BP	110	mg/hg

A non-relational representation of this data combines everything into one observation:

sysbp	sysbpunit	diasBP	diasBPunit
90	mg/hg	110	mg/hg

This is significant because the SDTM approach introduces the need for value-level metadata. Simply displaying VSTESTCD in the define file is inadequate. Its values and their meaning require additional metadata and, in turn, additional programming so that `Define.XML` can not only describe the variables' attributes, but also the possible values of the variable that have been, in effect, transposed.

Tools Using the Metadata. `%printCDspecs` displays all related metadata for a Domain in a PDF.

`%attrib` uses the VARIABLES table to create KEEP list and ATTRIB statements for a Domain. Since the macro reads from the metadata, the programmer is assured that the generated code is accurate.

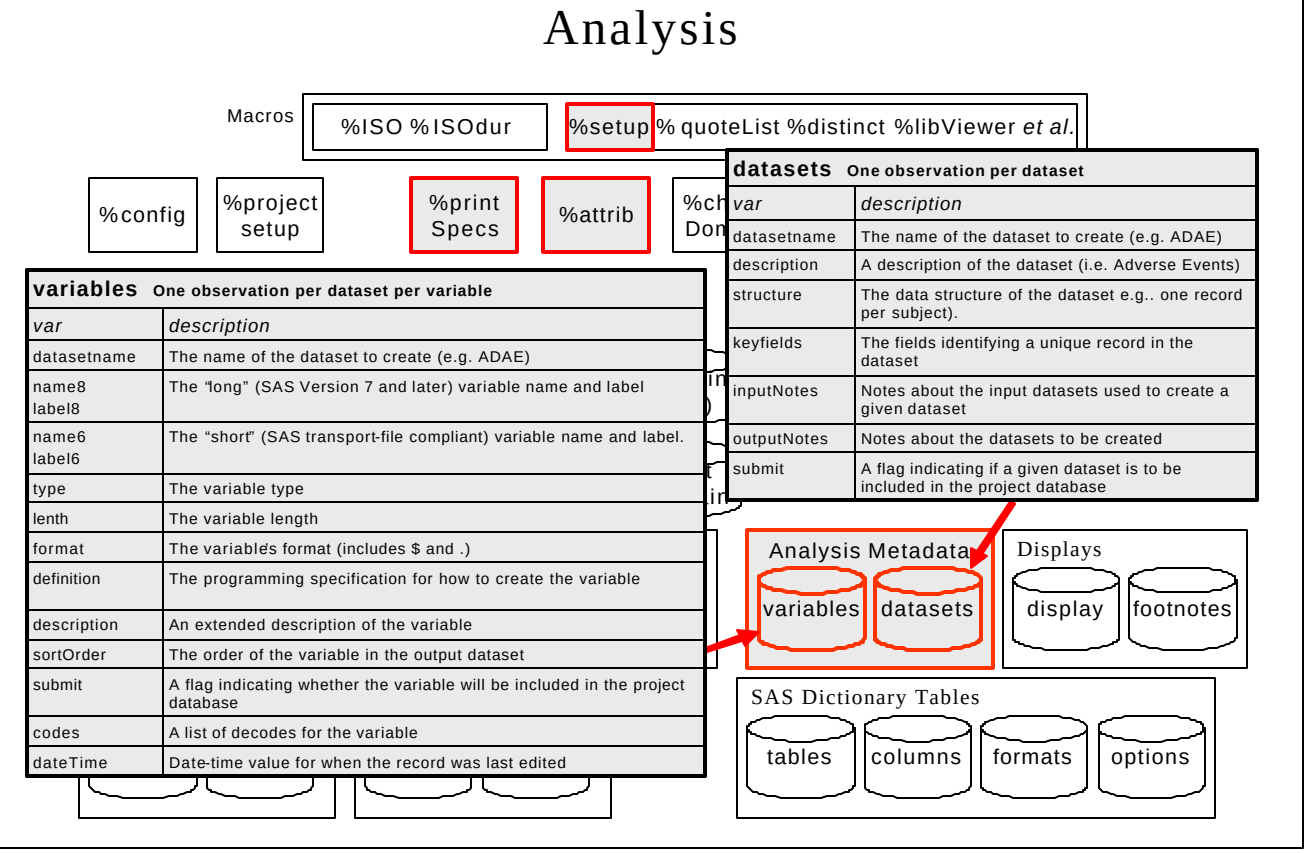
`%checkDomain` evaluates a created Domain for consistency with the variables described in the project-level tables and similar tables in the Gold Standard.

These tools are described in more detail in " Utilization in the SDTM Life Cycle," below.

ANALYSIS

Figure 7 describes the datasets and variables in the Analysis metadata. The contents of these data stores differs from their SDTM counterparts. They contain not only CRF variables found in the SDTM Domains, but also have derived variables (e.g., change from baseline). This difference, plus the absence of a "gold standard" to mandate data type and order, results in metadata that bears only some resemblance to the SDTM tables.

Figure 7



Tools. `%printSpecs` creates a programming spec PDF. Macro options give the user control over the sort order of the variables, whether to display only recently edited entries, and so on.

`%attrib` reads the variables table and creates `KEEP` and `ATTRIB` statements for a dataset. This relieves the programmer of the need to enter this information manually. It also ensures that the metadata and the output datasets will be in sync (provided, of course, that the programmer reruns the program creating the dataset once the metadata has been changed).

DISPLAYS

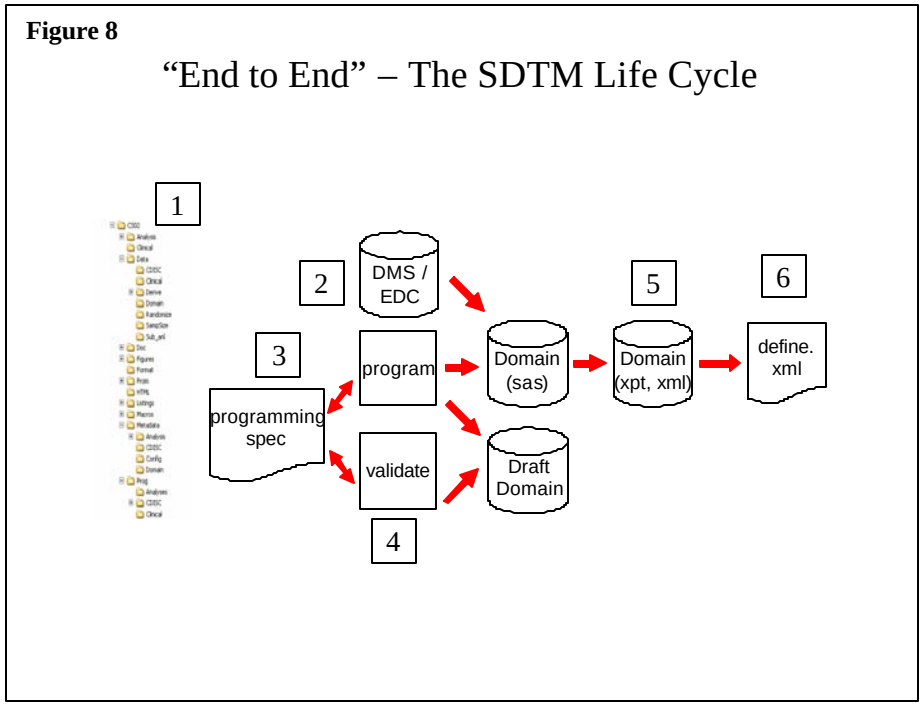
The creation of data displays – tables, figures, and listings – is well-suited for improvement using well-designed metadata and tools. The metadata can exploit similarities among a group of related displays. For example, the same table design might be used in 12 distinct displays, varying only by subpopulations (safety population, gender, etc.). Rather than write essentially the same program 12 times with only minor variations, we can create metadata, use macros to extract the metadata for a given display, and write a single, generalized program. This is a loaded statement, for it implies the need not only for new metadata tables and macros, but also for a change in the way displays are programmed. There is a learning curve, to be sure, just as there are there are also huge productivity benefits to be gained.

UTILIZATION IN THE SDTM LIFE CYCLE

The power and elegance of a metadata-driven system are best appreciated when seen in action. In this section, we'll illustrate the use of metadata during the creation of SDTM Domains and their accompanying documentation, Define.XML. For each phase, we will:

- Describe the work being performed, comparing "before and after" metadata usage efficiencies and issues;
- Identify the metadata tables that are read, written, or updated; and
- Describe some of the metadata tools that facilitate use of the metadata and then present program and output examples.

Figure 8



Overview

Figure 8 presents a highly simplified view of the SDTM life cycle. There is little here that is surprising to any one who has been involved in the production of clinical or analysis datasets and their documentation.

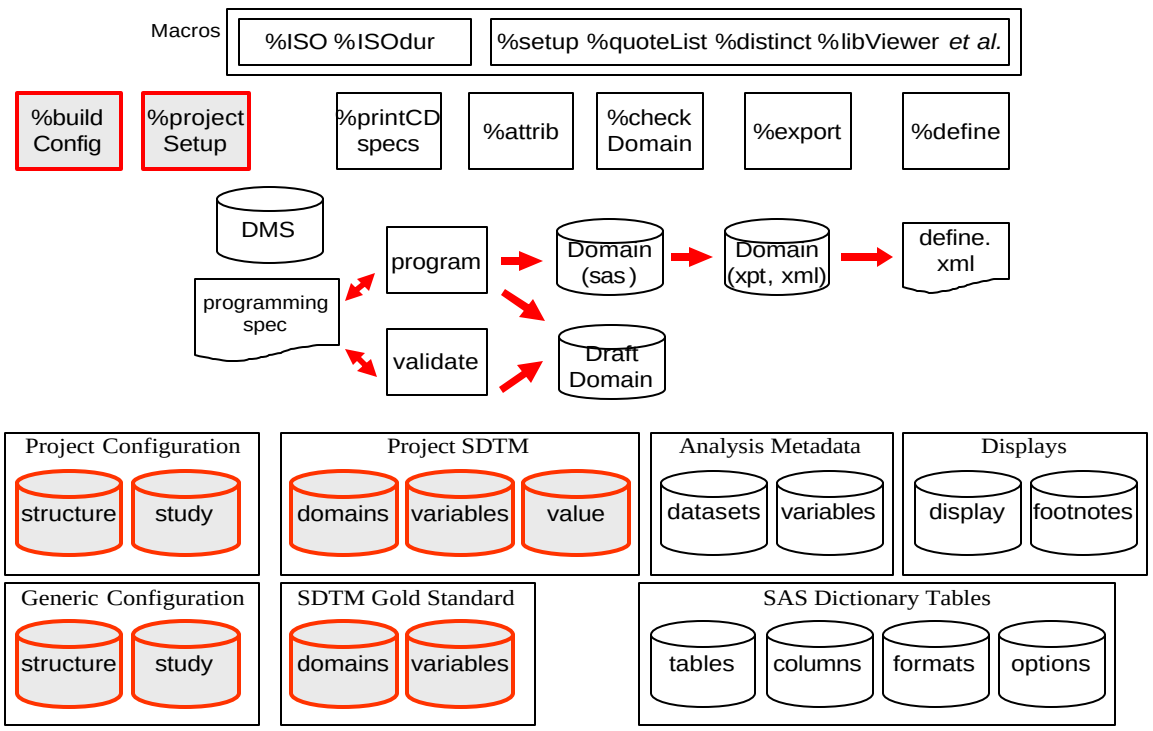
1. The project is initialized. A directory structure is created that complies with internal SOPs, and databases that will house configuration and other metadata are seeded with corporate-wide default values.
2. An in-house or outsourced data management system collects data from case report forms (CRFs) and other sources such as laboratory measurements.
3. Statisticians and support groups develop specifications for statistical programmers. Despite the presence of many predefined Domains, there are still many questions to be addressed. The spec writers need to consider: mapping of CRF variables to Domains; variables to use for conversion to ISO 8601 date-time values; what to put in supplemental Domains versus entirely new Domains; and so on.
4. Programming and validation for a Domain begin once the specifications are complete. The process is iterative: the datasets created by the primary and validation programmers may not be identical. The lack of agreement could be due to programming errors; it could also be caused by vague or incorrect specifications. Eventually, exchanges between the statistician, primary programmer, and validation programmer result in a valid, SDTM-compliant dataset.
5. The dataset is converted into a format acceptable to the FDA: either a SAS Transport file or an XML file.
6. The last stop in this simplified tour is the creation of the Domain dataset documentation. We create the "define" file as a PDF or, more typically, an XML file.

With this somewhat idealized view of the process in mind, let's take a detailed look at the process.

Project Initialization

One of the great advantages of a table-driven, metadata-oriented environment is that it promotes standardization. Perhaps nowhere in the system is this better seen than during project initialization, depicted in **Figure 9** (next page). This is a two-step

Figure 9



process. The user invokes a SAS macro (`%buildConfig`) to copy a generic configuration file to the project root directory. The user passes the project root directory location to the macro and, if creating a multiple-study project, specifies a list of study names and descriptions. A simple invocation looks like:

```
%buildConfig(root=d:\comm\bigPharma\cholBgone)
```

The macro copies the generic configuration MDB to the project directory, using the root directory to change path values in the project database. The macro also creates a PDF that shows what is essentially the directory structure, LIBNAME assignment, and macro and format path sequence that will be used each time the metadata tools will be used. Once the output is reviewed and the user edits the MDB, the directory structure can be populated by the `%projectSetup` macro.

```
%projectSetup(root=d:\comm\bigPharma\cholBgone)
```

`%projectSetup` reads the configuration MDB created by `%buildConfig` and:

- Creates directories underneath the project root directory;
- Copies generic SDTM and other metadata to the newly-created directories;
- Copies generic `RUN_xxx.SAS` programs to project directories. These programs run some of the metadata tools, and are populated with typical default macro parameters suitable for the study. The following is an example of the generated program `Run_printSpecs.sas`. Note that the `%setup` call is prefilled with the correct `drive` and `project` parameters.

```
/* The program header has a copy of the source macro's header.  It contains a description
   of input and output datasets, macro parameters, examples of use, and other usage
   notes. */
%inc 's:\submissions\macros\util\setup.sas' / nosource2 ;
%setup(drive=s,drive=S, project=RHO\Sponsor\Project\PR01)
%printSpecs(order=alpha, use=_all_, skip=)
```

- Writes a PDF showing the directory tree that was created. Other diagnostics in the SAS Log identify the metadata tables (SDTM and others) that were seeded and the sample programs that were copied to the project directory.

The advantages of this approach compared to a manual one are significant.

- LIBNAMEs and directory locations for common data stores (raw data, derived data, macro libraries, etc.) are stored as data that will be accessed by a setup macro. Thus there is no need for a manually-created initialization program and, assuming no LIBNAMEs were changed, the new project uses the standard, corporate-wide names found in the SOPs.
- The sample programs don't require any editing except for parameter changing. The documentation header for each macro called is included with each `RUN_` program.

Once the directory structure is created and the metadata databases are seeded, the domain creation process can begin. First, though, let's take a look at the macro that is called at the beginning of every program in the system. `%setup`, as the name suggests, reads the configuration MDB and assigns LIBNAMEs, creates option and format search paths, and performs other activities based primarily on the `STRUCTURE` table. The call is simple:

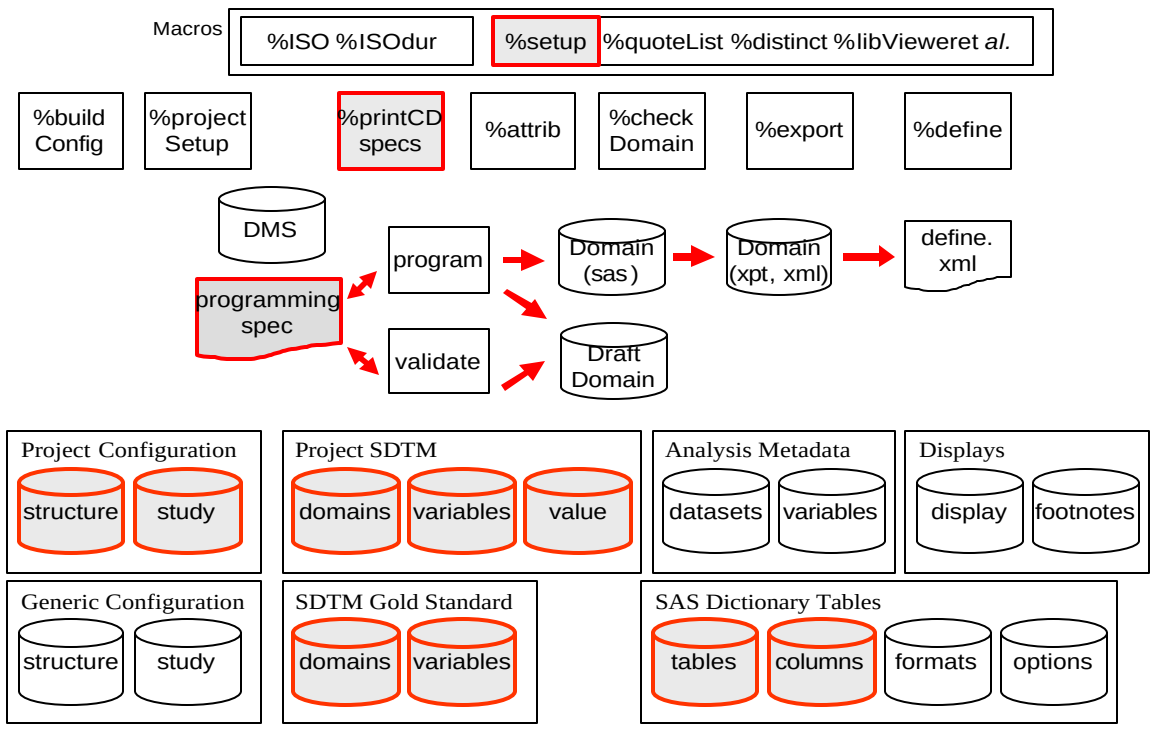
```
%setup(project=root_directory)
```

Diagnostic output is written to the SAS Log and to `setup.txt`. Typical Log output follows:

```
SETUP-> Begin ------------------------------------------------------------------------------
SETUP-> SETUP Ver 3. For help, rerun with HELP=y, or see s:\submissions\docs\setupMacro.doc
SETUP-> Run time [Friday, February 23, 2007 1:42 PM]  SAS Version [9]
SETUP-> Project [RHO\Sponsor\Project\PR01]  config [metadata\config\config.mdb]
SETUP-> Project drive [S]  Prefix []  Generic drive [S]
SETUP-> Configuration database [S:\RHO\Sponsor\Project\PR01\metadata\config\config.mdb]
SETUP-> LIBNAME AMETAANL had this engine option value: dbmax_text=10000
SETUP-> LIBNAME AMETACD had this engine option value: dbmax_text=10000
SETUP-> LIBNAME ACONFIG had this engine option value: dbmax_text=500
SETUP-> ODS path: template.templates sasuser.templat(update) sashelp.tmplmst(read)
SETUP-> See S:\RHO\Sponsor\Project\PR01\Prog\Derive\setup.txt for diagnostics
SETUP-> Processing required 1.3 seconds.
SETUP-> End --------------------------------------------------------------------------------
```

The `setup.txt` file noted in the Log is a thorough description of all the LIBNAMEs, macro variables, and other settings that were created by the macro. Here, and throughout the system, the benefits of having clear, thorough diagnostics are two-fold.

Figure 10



First, they provides information to the user in an easily readable format. Input and run-time errors are caught and reported by the macros so that the program fails gracefully (that is, we don't let SAS generate pages of run-time errors). This leads to the

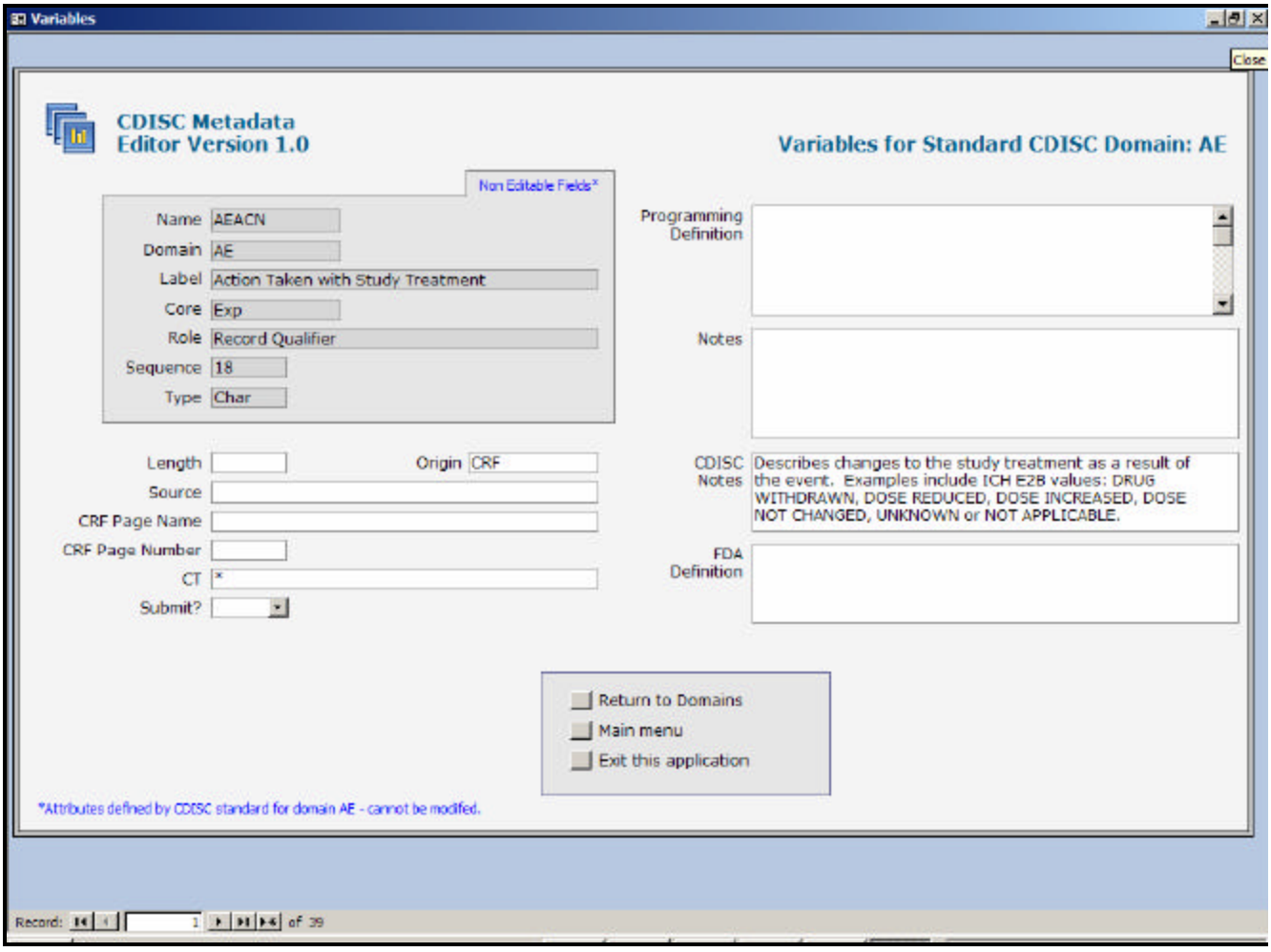
next benefit of the system design: ease of use and good diagnostics encourage both continued usage and suggestions for enhancements and new tools. This is a back door approach, if you will, to acceptance of the standardized library names and directory structures encouraged by the SOPs.

Programing Specs

For years, the specifications for creating datasets were held in word processing formats such as Microsoft Word. Although this was a familiar and easy-to-use tool, it required considerable transcription by programmers. Dataset and variables specifics such as lengths, labels, and formats had to be copied into SAS programs. The process was tedious and error-prone. Moving the specifications out of a document and into a database (**Figure 10**, previous page) created significant opportunities. Now not only could a spec document be produced (duplicating and improving on the look and feel of the Word documents), but the document, since it is a table conceptually identical to a SAS dataset, could be used to generate SAS `FORMAT`, `LENGTH`, and other statements. This section describes generation of the spec document. We'll see other programming tools later.

Recall that one of our points in the "Lessons Learned" section was that metadata are complex and so require a user-friendly interface. **Figure 11**, below, shows a one of the data entry prototypes. The screens are simple and intuitive. This makes data entry easier and results in higher initial metadata quality (and, not to be discounted, happier statisticians).

Figure 11 CDISC Metadata Entry



A statistician or programmer can use the SDTM metadata to produce a spec document at any point during the metadata entry process. The `%printCDspecs` macro processes the metadata and produces a PDF that not only displays the metadata but also performs some metadata quality checks (note that some checks were already performed by the GUI and that other, more thorough checking is done later, by the `%domainCheck` macro). The macro displays dataset-level metadata for a Domain and, if found, its supplemental qualifier ("suppqual") dataset, followed by variable-level and value-level data for the Domain.

Suppose a programmer wants to review the spec for Domain AE. The complete program follows:

```
%include 'utility_directory\setup.sas' / nosource2;
%setup(project=project_root_directory)
```

```
%printCDspecs(use=ae)
```

Messages to the Log describe the macro's activity, noting among other items the complete path of the output PDF. Other parameters to the macro control the order of the variables (alphabetical within Domain, position within Domain), Domains *not* to process, and so on. **Figure 12**, below, shows partial screen capture for a Domain. The output displays not only the metadata information but also other features that are sometimes helpful. Among these: location of the Gold Standard metadata; key variables from the `Domains` table; location of the program generating the display; and highlighting of Required or Expected variables.

Figure 12 %printSpecs Output

Variable Specs for Domain AE 2

Gold Standard [S:\sasbu\invivo\metadata\SDTM31.mdb] Study [redacted] MetaData\CDISC\SDTM.mdb Mod Date 13FEB07:14:21]
Key vars from DOMAINS table: STUDYID USUBJID AETERM AESTDTC
Variables are arranged by their position in the output Domain. Only Expected/Required and SUBMITDB=Yes Variables Are Displayed

							CRF Page		
CDname [2]	Label [1]	Type [1]	Len	Core [1]	sub mit DB	Source	Name	#	Comments
STUDYID	Study Identifier	Char	12	Req	Yes	CRF	DEMOGRAPHICS AND REPRODUCTIVE STATUS	1	[ProgDef] ='E7974A00110Z'
DOMAIN	Domain Abbreviation	Char	2	Req	Yes	DER			[ProgDef] ='AE'
USUBJID	Unique Subject Identifier	Char	24	Req	Yes	DER			[ProgDef] Concatenate RAW.DEMO.PROTOCOL and three-digit RAW.DEMO.ID separated by '-'
AESEQ	Sequence Number	Num	4	Req	Yes	MC			[ProgDef] =RAW.AECDISC.AESEQ
AETERM	Reported Term for the Adverse Event	Char	70	Req	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AE
AEDECOD	Dictionary-Derived Term	Char	200	Req	Yes	DER			[ProgDef] =RAW.AECDISC.PT
AEBODSYS	Body System or Organ Class	Char	200	Exp	Yes	DER			[ProgDef] =RAW.AECDISC.SOCT
AESEV	Severity/Intensity	Char	8	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =put(RAW.AECDISC.AE_SEV, AESEVF.)
AESER	Serious Event	Char	3	Exp	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =put(RAW.AECDISC.SERIOUS, NYF.)
AEACN	Action Taken with Study Treatment	Char	32	Exp	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =put(RAW.AECDISC.ACTION, ACTF.)
AEACNOTH	Other Action Taken	Char	100	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AEACNOTH
AEREL	Causality	Char	18	Exp	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =put(RAW.AECDISC.AE_REL, AERELF.)
AEOUT	Outcome of Adverse Event	Char	25	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =put(RAW.AECDISC.OUTCOME, AEOUTF.)
AESCONG	Congenital Anomaly or Birth Defect	Char	1	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AESCONG
AESDISAB	Persist or Signif Disability/Incapacity	Char	1	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AESDISAB
AESDTH	Results in Death	Char	1	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AESDTH
AESHOSP	Requires or Prolongs Hospitalization	Char	1	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AESHOSP
AESLIFE	Is Life Threatening	Char	1	Perm	Yes	CRF	ADVERSE EXPERIENCES	88	[ProgDef] =RAW.AECDISC.AESLIFE

[1] Red background indicates a Gold Standard that was changed at the study level. The value displayed is the Study value.
[2] Blue background indicates Key variable (in DOMAIN variable KEYS)
Table entries: [1] Required/Expected variables and/or [2] CDNAME found in project metadata
Shaded rows indicate Required or Expected variables (CORE='Req' or 'Exp')

Run date-time: Wednesday, February 14, 2007 2:19 PM
Program: S:[redacted]Prog\CDISC\run_printALLspecs.sas

Programming Tools

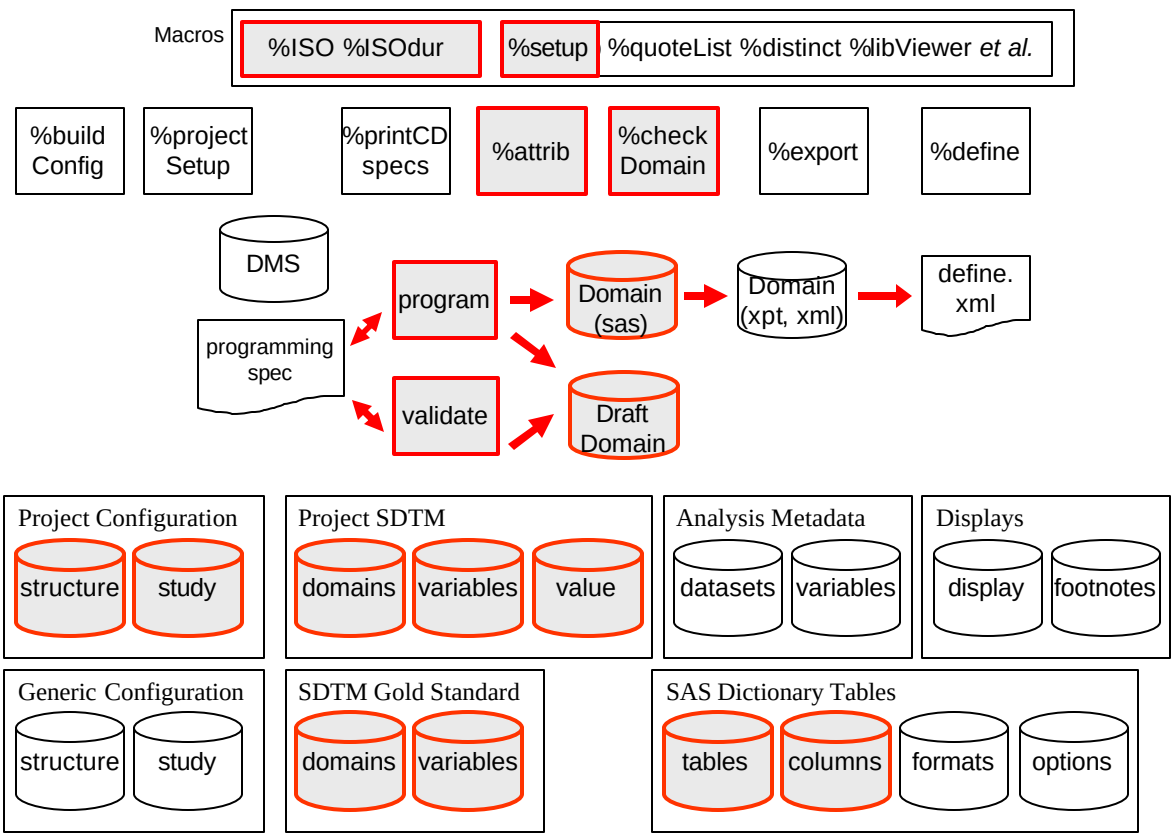
The creation of the Domains, as we noted earlier, requires interplay among the primary programmer, validation programmer, and the statistician who wrote the specification. The Domain dataset is FDA-ready only when all three agree that the specification is correct and that the two programmers independently created a dataset that represents the spec's intentions. **Figure 13** (next page) presents an overview of the process.

Conforming to ISO 8601: %ISO and %ISOdur

Date and time-related values and durations are represented in SDTM using the ISO 8601 standard. The ISO specification is thoroughly documented, a tad arcane, and for most programmers, unfamiliar. We created several macros that offload the programming burden. (The spec writer does not necessarily have to know *anything* about ISO standards other than entering "ISO 8601" in the definition, along with the variables containing the date or date-time components.)

%ISO The programmer passes the name of the variable to be created, whether it is building a date, datetime, or time variable, and the component/input variables. The output variable is an ISO 8601-compliant date-time in extended format (as required by the SDTM guidance). The macro call depends on the type of input variable being passed. In the simplest case, where a SAS date-time variable is being converted, the call looks like:

Figure 13




```
data cdisc.AE;
     set AEtemp;
     %iso(dateTime=AEstdtm, outVar=AEstdtc)
```

The macro looks for parameter conflicts, writes messages to the SAS Log as needed, and creates character variable `AEstdtc` as an ISO 8601-compliant date.

%ISOdur Conforming to the ISO 8601 representation of event duration is, to put it mildly, a challenge. Values can be represented in a number of forms: duration, duration/end date-time, start date-time/duration, or start date-time/end date-time.

Here, as with `%ISO`, we developed a tool that allows the Domain dataset programmer to pass a minimal set of parameters, knowing that errors and inconsistencies will be identified and that the duration value that is created will be ISO 8601-compliant. A call to the macro looks like:

```
%ISOdur(unit=dt, start=AEstdtc, end=AEendtc, out=AEdur, outType=d)
```

This call to the macro creates `AEdur`, which holds the duration between date-time variables `AEstdtc` and `AEendtc`. The value stored in AEdur contains duration in the form `PyyYmmMddD`. For example, if the duration between the two events was 1 year, 5 months, and 2 days, `AEdur` would equal `P1Y5M2D`.

Using Metadata to Build Code: %ATTRIB

One of the most onerous tasks of the pre-metadata programming days was manually moving the dataset specification's variable names, formats, and other attributes into the text of a SAS program. Once the metadata were moved into a format that was programmatically accessible, issues of accuracy and timeliness disappeared, and programmers were happier. The `%ATTRIB` macro reads variable-level metadata for a Domain and builds portions of `KEEP` and `ATTRIB` statements. Since the macro reads directly from the metadata, there is no possibility of specs and programs being out of sync – if a label changes in the metadata for table X, the next time `BUILD_X.SAS` runs, it will pick up the change. The key point here is that the

programmer does not have to change the program. Each time the program is rerun, %ATTRIB picks up the variable attributes (variable name, label, length, type, and format) currently specified in the metadata.

The macro is simple to use. It creates macro variables _ATTRIB and _KEEPLIST. In the simplest case, all the programmer needs to know to use this tool are the names of the input metadata and Domain.

```
%attrib(in=meta.variables, domain=DM)
data CDISC.DM;
     %unquote(&_ATTRIB)
     other SAS statements
     keep &_keeplist;
     run;
```

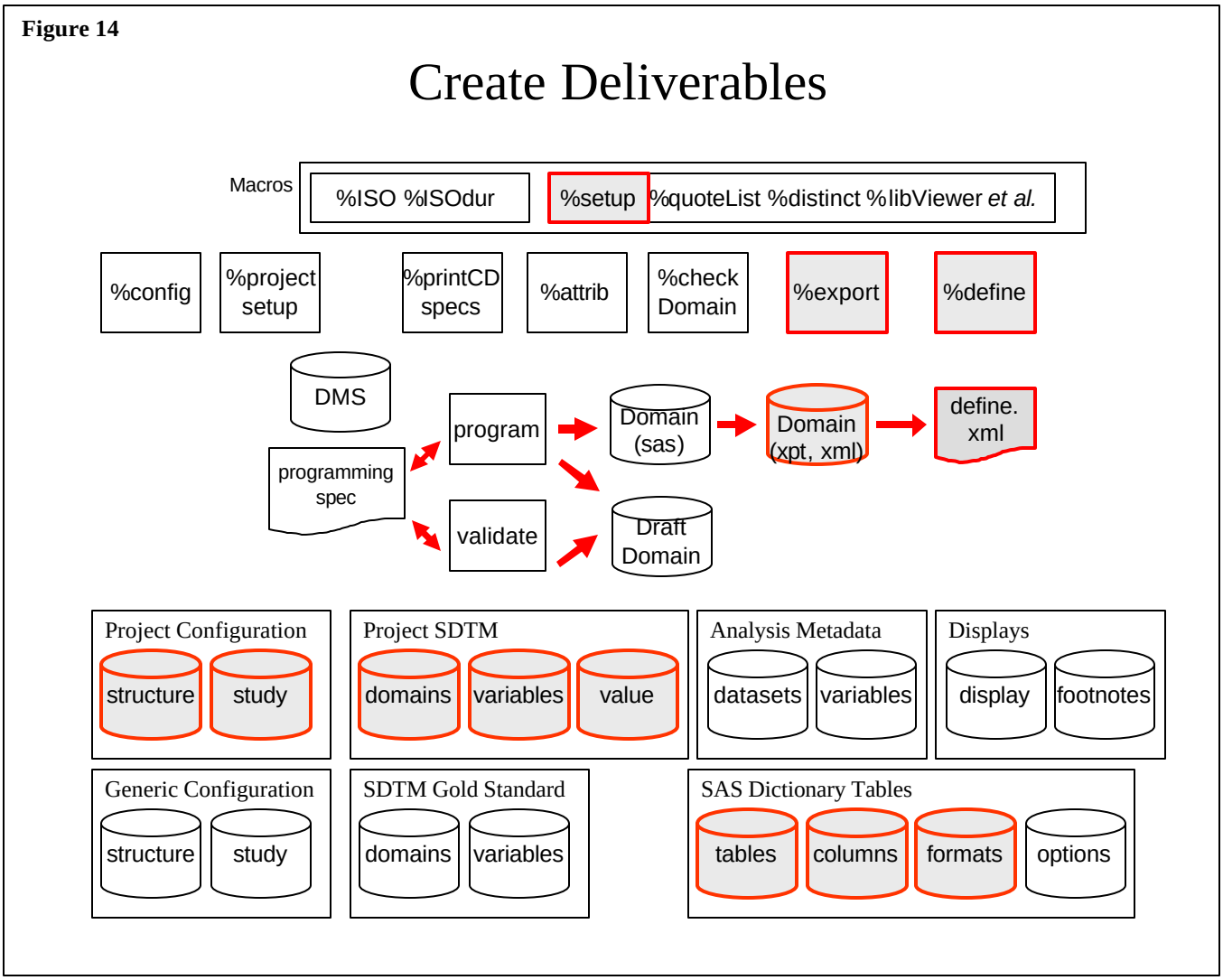
Making Sure We Have What We Need: %checkDomain

Two programmers producing the same results only validates that they understood the specification the same way. The %checkDomain macro performs additional QA by comparing what was produced (the Domain dataset) to the SDTM standard for the Domain. It creates a report summarizing errors and possible inconsistencies. The macro says, in effect, "Let's compare the attributes of the Domain datasets to those in the project SDTM and the Gold Standard metadata." Just *what* constitutes an error is sometimes a matter of judgment, but many of the criteria are taken from the current guidance from CDISC. (A complete list of checks currently being used is found in

Creating Deliverables: %export and %define

The final phase of the SDTM life cycle is the creation of data and documentation deliverables. This is depicted in **Figure 14**, below.

Figure 14



%export

Conversion of the Domain datasets from the proprietary SAS dataset format (Windows files with a SAS7BDAT extension) to a public domain format acceptable by the FDA (transport or XML) is straightforward. The `%export` macro utilizes the SAS dictionary tables to create a list of datasets in the CDISC domain directory, then reads the configuration metadata to identify the target directory. The macro call is as simple as its functionality. A complete exporting program follows:

```
%include 'utility_directory\setup.sas' / nosource2;
%setup(project=project_root_directory)
%export(type=xpt)
```

%define

Things get more complicated when creating `define.XML`. This happens for several reasons. First, recall the discussion of value-level metadata ("SDTM (Project)," above). Simply displaying the name of a transposed variable in the define file is inadequate. Its values and their meaning require additional metadata and, in turn, additional programming. `Define.XML` must not only describe the variables' attributes, but also the possible values of the variable that have been transposed.

The second complexity introduced by `Define.XML` is the additional hyperlinking. Continuing the VSTESTCD example, there must be a link between the *variable*-level VSTESTCD and the display of its *value*-level metadata. The sample CDISC `define.xml` and `define.xslt` files demonstrate this and other internal hyperlinking. These are only the starting point for truly robust and usable files – modifications are needed to the XSLT and, to a lesser degree, the XML file so that the screen and printed versions of the files will be accurate as well as aesthetically pleasing.

The `%define` macro heavily utilizes the project CDISC and configuration metadata to create `Define.xml`. It also uses numerous general-purpose macros (discussed in "General Purpose Tools," below) to unload user-written format libraries, create variable and dataset lists, and so on. The call, however, can be deceptively simple:

```
%defineXML
```

The macro adheres to an important design principle: as the number of "moving parts" of a macro increases, so should its default diagnostic output. The macro's Log messages tell the story of its execution. If all goes well, the story is a routine list of tasks attempted and successfully completed. If there is a problem, the macro should fail gracefully, explaining the problem and shutting down without letting SAS create needless and, frequently, puzzling error messages. (The `%defineXML` Log excerpt in DiIorio and Abolafia, 2007, Appendix A, illustrates this principle.) The "Log as execution narrative" approach requires more coding, but is worth the effort. Consider, too, that if a general-purpose macro utility library is available, the coding effort can be significantly reduced.

General-Purpose Tools

If metadata without tools to access them is difficult, then access tools built without the benefit of a general-purpose utility library is almost impossible. We have developed a collection of well-documented macros that perform tasks ranging from the small-scoped and tightly defined to complete, standalone applications. The use of this library accrues several benefits:

- **Reliability.** The utilities only provide well-defined outputs (reports, macro variables, *et al.*). Additionally, the processing performed by the macro is likely more thorough than an application program's "quick and dirty" coding.
- **Code Reduction.** A one-line call to a macro, possibly followed by another line that tests a return code or similar status check, is much more compact than writing code in-line. The fewer "moving parts" in a program, the better.

In the remainder of this section we discuss how some of these tools are used, either in conjunction with metadata tools or as individual applications.

Case 1: Use In a Metadata Application

This code fragment demonstrates how a metadata access application can use generalized tools from a macro library. Don't focus here on how the macro is called and what it produces. Rather, consider how the application is not burdened with tedious SQL coding, quoting, and the like. That burden is offloaded to the macro, allowing the application programmer can focus on the program's core functionality.

The macro uses variable-level metadata to create a list of datasets marked for FDA submission. For each of dataset, it looks for identifier variables `UNIQUEID, PATNO,` and `SCREENNO` (not all variables are necessarily in each Domain). Then it lists any of the located observations from the Domain.

```
%macro listIDs;
       %distinct(data=domMeta.variables, var=domain, where=%str(submit=1))
       %do i = 1 %to &nLevels.;
           %let token = %scan(&levels., &i.)
           %varCheck(data=domain.&token., varList=uniqueid patno screenno)
           %if &found. ^= %then %do;
               proc report data=domain.&token. panels=5;
                    columns &found.;
                    title "Identifier Variables for Domain &token."
```

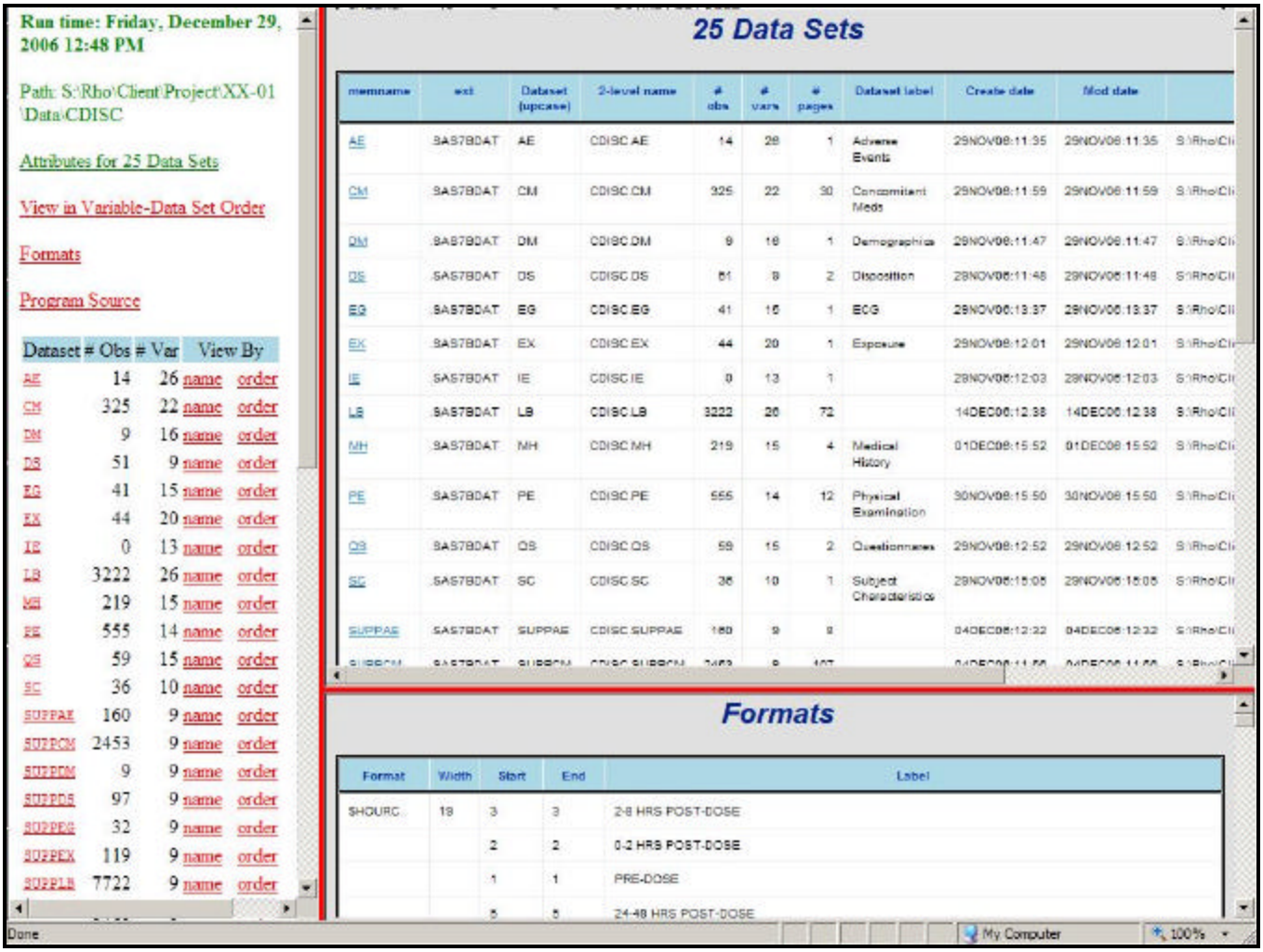
```
                run;
                %end;
                %else %put No identifier variables were found for domain &token.;
        %end;
%mend listIDs;
```

The code is compact and reliable, and will fail gracefully if identifier variables cannot be located. `%distinct` and `%varCheck` create lists and macro variables, held in `&nLevels`, `&levels`, and `&found`. The program is made both shorter and more reliable by their use. All that is required on the part of the programmer is to know that they are available and how to use them.

Case 2: Standalone Application

Some generalized, recurring requirements can become standalone applications that are useful diagnostic or display tools for a project. `%libViewer` is an example of this class of support tool (**Figure 15**, below). Unlike `%distinct` and `%varCheck`, shown in the previous example, `%libViewer` is a full-fledged, standalone application. It heavily utilizes the macro library to read the SAS Dictionary tables for an input library and create an HTML frameset. The application displays dataset and variable-level information about the library's datasets. It also unloads any user-written formats currently allocated and builds links from the variable-level tables to a frame containing the contents of the individual formats. The HTML is useful when

Figure 15 %libViewer Output



questions about variable types, formats, and the like arise. The program is relatively short (489 lines, 280 statements). Its design points out the power of a good utility library: it would be dramatically longer and likely less bulletproofed if it did not effectively use the utilities.The invocation is simple:

```
%libViewer(lib=domain)
```

Ad Hoc Programming

It would be foolish to believe that we could ever have a comprehensive, all-inclusive set of metadata and metadata tools that could address all data, display, and submission needs of all our projects. This is especially true in the CRO environment,

where corporate agility is key; often data may come from in-house systems or one or more EDC vendors. Add to this the diverse data, documentation and other standards of multiple clients, and you can easily see why flexibility (and "hand me the Maalox") becomes part of the programming mantra.

The unpredictable demands of this environment can be handled to some degree by knowing the metadata and generic utility resources that are available. We present two scenarios here, and then comment on the solutions.

Case 1: Use Project Metadata

Scenario. Analysis dataset metadata for a project checked out correctly – variable types, formats, lengths, and definitions were scanned by our generic metadata validation tools. The sharp-eyed statistician assigned to the project noticed a discrepancy in the wording of variable labels between two variables residing in different datasets but with different names.

Rather than manually review the metadata in Access, we used the analysis metadata as input to a program that read the metadata, built a string containing various attributes, then compared the string for like-named variables, reporting instances where there were multiple values (i.e., differing characteristics). The core section of the program is shown below:

```
proc sql noprint;
     create table attribProblems as
     select varName, dsName, type, length, format, label,
            catx('#', type, put(length,5.), format, label) as compare
     from metaANL.variables(keep=dsName varName type length format label)
     group by varName
     having count(distinct compare) > 1
     order by varName, datasetName
     ;
quit;
```

Once dataset `attribProblems` dataset is created, it's a simple matter to use a reporting procedure such as `PRINT` or `REPORT` to display the results.

Case 2: Use Generic Utilities

Scenario. Visual review of early drafts of CRT datasets that were to be input to SDTM domains identified several variables that contained only missing values. The reviewer wanted a way to identify all missing variables that wasn't as error-prone and tedious as manual inspection. It was a fairly simple matter to use several general-purpose macros to quickly produce the desired output. The program is shown in its entirety below (comments have been removed for the sake of brevity):

```
%inc 'directory\setup.sas' / nosource2 ;
%setup(project=d:\comm\bigPharma\cholBgone) /* Allocates LIBNAME DOMAIN */

%macro allMiss(libname=);
       %memlist(lib=&libname.)
       %let path = %sysfunc(pathname(&libname.));
       %put; %put LIBNAME=&libname. Path=&path.~Datetime: &dateTime.)
       %do i = 1 %to &_nmemList_.;
           %let temp = %scan(&_memList_., &i.);
           %nLevels(data=&libname..&temp, out=temp_&temp.)

           %dsetAttribs(data=&libname..&temp)

           %put; %put [&i. of &_nmemlist_.]  Dataset [&libname..&temp.];
           %put # obs: &nobs.  # vars: &nVar. (&nchar. char, &nnum. num)  Updated: &modC.;
           %put &nMissing. variables have all missing values: ;
           %put &missing.;
       %end;
%mend;
%allMiss(libname=domain)
```

A partial listing of the Log output is shown below:

```
LIBNAME=sderive Path=S:\Client\Project\XX-01\Data\Derived
Run date-time: Friday, February 23, 2007 11:09 AM
Datasets [AE DEMOG PROTDEV RANDLIST]
[1 of 4]  Dataset [sderive.AE]
# obs: 203  # vars: 31 (20 character, 11 numeric)  Last updated: 23FEB07:11:01
No variables had all missing values
[2 of 4]  Dataset [sderive.DEMOG]
# obs: 82  # vars: 16 (9 character, 7 numeric)  Last updated: 23FEB07:11:01
No variables had all missing values
[3 of 4]  Dataset [sderive.PROTDEV]
```

```
# obs: 109  # vars: 12 (11 character, 1 numeric)  Last updated: 23FEB07:11:02
1 variable with all missing values: D_COMMENTS
[4 of 4]  Dataset [sderive.RANDLIST]
# obs: 82  # vars: 8 (6 character, 2 numeric)  Last updated: 23FEB07:11:02
3 variables with all missing values: D_AEACT D_AEACT_C D_DAY
allMissing-> End
```

Scenario Comments

A review of how these ad hoc programs work, rather than what they produced, reveals the following important points:

- The programs will always supplement, not replace, manual processes. Some parts of the project life cycle are best done manually. The role of the ad hoc programs shown here is to help team members exercise their good judgment without forcing them to stay up through the wee hours of the night.
- The programs are compact and were coded quickly because we were aware of the resources at our disposal, namely, the metadata databases and utility library macros. The attribute-checking program required knowledge of the metadata tables. The missing value program required knowledge of the workings of several general-purpose macros (`%nLevels`, `%memList`, and `%dsetAttribs`).
- They started as ad hoc, but these programs can be integrated into larger, more formal and validated programs. The attribute inconsistency program could, for example, become a standard Analysis metadata quality assurance check. Its output would be enhanced in a production-quality tool (for example, we could highlight or display only the conflicting attributes). A formal, production-quality version of the program would also implement parameter and run-time error trapping (test for valid formation of the passed LIBNAME, see if the library had been allocated, and so on).

CONCLUSION

Pharmaceutical projects are typically complex, involving organizing hundreds of files in wide variety of formats. Furthermore, as the submissions date nears, there is an increased need for quick production of deliverables without sacrificing quality. Moving to a metadata-driven system at Rho has lead to more efficient processes at each phase in the project life cycle, and has improved the overall quality of deliverables. By utilizing a metadata based system:

1) Studies are now set up and configured by a standard program. This ensures a uniform setup across projects and allows studies to be setup instantaneously by a wide variety of personnel.Also by storing setup information in a database, it can be utilized by all subsequent programs.
2) Specifications for datasets as well as displays are now held in a database instead of in a document. This has lead to considerable savings in the time to produce datasets and displays and has led to higher quality datasets and displays. Furthermore, the data comprising these specifications can be repurposed downstream when producing transport and define files.
3) Once the dataset and display specifications have been entered, standard push button programs can be created to produce submission deliverables such as the SAS transport files and the Define file.
4) Quality control can be automated and made more robust. Programs can be developed to check specifications and to check the consistency between the metadata and the data. Also, by using metadata as input to programs, programmer error can be decreased significantly.
5) Metadata created at the early stages of a project can be utilized throughout the life cycle of a project, thereby increasing efficiency and decreasing the amount of time and resources needed for project deliverables. For example, metadata created at study setup time can be used at all subsequent stages of a project.

ACKNOWLEDGEMENTS

Russ Helms and **Ron Helms** of Rho, Inc. provided the impetus for metadata use at Rho. They have also created an environment at Rho that encourages experimentation, rewards success, and is sympathetic when ideas don't pan out.

April Sansom provided her typically thorough copyediting input, and is probably wondering why, with all those years of Catholic school education, the authors punctuate in a way that can only be described as whimsical.

REFERENCES

Previous papers by the authors, focusing on metadata and tool development:

Abolafia, Jeff, "What Would I Do Without PROC SQL And The Macro Language," SAS Institute Inc. 2005. Proceedings of the Thirtieth Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.

DiIorio, Frank, “Controlling Macro Output or, ‘What Happens in the Macro, Stays in the Macro’,” SAS Institute Inc. 2006. Proceedings of the Fourteenth Annual SouthEast SAS® Users Group Conference. Cary, NC: SAS Institute Inc.

DiIorio, Frank and Jeff Abolafia, “From CRF Data to Define.XML: Going “End to End” with Metadata,” SAS Institute Inc. 2007. Proceedings of the Ninth Pharmaceutical SAS Users Group Conference. Cary, NC: SAS Institute Inc.

DiIorio, Frank and Jeff Abolafia, “The Design and Use of Metadata: Part Fine Art, Part Black Art,” SAS Institute Inc. 2006. Proceedings of the Thirty-first Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.

DiIorio, Frank, “Rules for Tools: The SAS Utility Primer,” SAS Institute Inc. 2005. Proceedings of the Thirtieth Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.

DiIorio, Frank and Jeff Abolafia, “Dictionary Tables and Views: Essential Tools for Serious Applications,” SAS Institute Inc. 2004. Proceedings of the Twenty-ninth Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.

Also see http://www.CodeCraftersInc.com for other papers and resources not directly related to the current paper.

CDISC resources:

define.XML http://www.cdisc.org/models/def/index.html

SDTM Version 3.1 http://www.cdisc.org/models/sds/v3.1/