PharmaSUG 2016 - Paper DS03

Results-Level Metadata: What, How, and Why

Frank Dilorio, CodeCrafters, Inc., Philadelphia PA
Jeffrey Abolafia, Rho Inc., Chapel Hill, NC USA

ABSTRACT

The power of well-designed metadata has been widely demonstrated in recent years. Organizations creating standard-compliant FDA deliverables (SDTM and ADaM datasets, define.xml) find that the creation was simplified by using data that describe the deliverables' workflow and content – that is, metadata.

One part of submission deliverables – analysis displays – hasn't received the same level of attention. This is changing. The FDA's emphasis on traceability and the recent release of the CDISC Analysis Results Metadata schema have shone a bright light on the need for using metadata for this critical piece of the submission package.

This paper is an overview of results-level metadata. It describes what is meant by results-level metadata; discusses collection techniques; illustrates how it can be used during the creation of analysis displays; and summarizes what is needed for the define-xml to be results-level compliant. The paper should give the reader an appreciation of both the scope of the metadata and its usage beyond simply creating define.xml. Used properly, it can be one of the many benefits of implementation of an end-to-end, metadata-driven system.

INTRODUCTION

One of the major goals of CDISC is to provide end-to-end standards. With the release of the results metadata specification earlier this year, this objective has finally been realized. In early 2015 CDISC published Version 1.0 of the Analysis Results Metadata (ARM) specification, which is an extension of the define-xml Version 2 standard. This extension provides a model for submitting machine-readable metadata that describe analysis results as part of the define file. With the addition of ARM, CDISC not only delivers standards from protocol to analysis results, but also provides full traceability from collected data to results.

In the sections below we describe ARM: what it is, the contents of the ARM extension to the Define-XML Version 2 standard, the business case for submitting analysis results metadata, use cases for analysis results metadata beyond submission, and recommendations for implementation.

BACKGROUND

Since its inception, CDISC standards were provided in a structured format that could be *stored* as machine readable metadata. CDISC also presented a model for *delivering* machine readable metadata to regulatory agencies. Version 1.0 of the "Case Report Tabulation Data Definition Specification" (define.xml), released in 2005, provided an XML schema for submitting machine readable metadata. While its focus was on SDTM, it could also be used for creating a define file for ADaM.

The concept of results metadata was introduced in 2006, with the publication of the CDISC Analysis Data Model (ADaM) Version 2.0. The intent was to describe key components of significant (not necessarily *all*) analysis results, providing a regulatory reviewer with text and links from a result to metadata and other resources describing the analysis. The ADaM Version 2.0 guide documented the fields to describe each result and presented an example of analysis results metadata.

At this point a metadata *model* to describe analysis results existed, but a formal *mechanism* for submission did not. In order to submit results-level metadata, a sponsor had to write their own extension to the define.xml schema. Not surprisingly, very few sponsors took this approach. In 2013 CDISC released Version 2.0 of the define.xml specification, which included updates and a re-structuring of the original specification. In 2015 CDISC published an extension to define.xml Version 2.0, which provided the long-awaited ARM schema. As a result, sponsors can now submit analysis results metadata as a part of define.xml. However it is worth noting at of this writing (2016Q1) analysis results metadata is not a required part of a submission to the FDA. The Japanese PDMA is a small step ahead of the curve, strongly encouraging, in its technical conformance guide, inclusion of results metadata and mandating it by 2020Q2.

WHAT: ANALYSIS RESULTS METADATA VERSION 1.0

Version 1.0 of the ARM specification has been released and is available at http://www.cdisc.org/adam. The release package ZIP file contains a readme file, the specification document, an ADaM-based define-xml example and the Analysis Results Metadata schema. The readme file presents an overview and a description of the contents and structure of the ZIP file.

The specification document outlines the results level metadata extension to the define.xml Version 2.0 standard. It describes the rationale for results level metadata, how in fits into a regulatory submission, and each of the components or attributes of the results metadata extension. The document also provides several examples of results level metadata, including examples of how each of the attributes is implemented. The appendix illustrates how the various attributes are rendered by the style sheet included in the package.

Not surprisingly, the “schema” folder contains schemas for: 1) the analysis results metadata extension to define.xml; 2) the current version of the CDISC ODM standard (1.3.2); and 3) the define.xml extension to the ODM standard.

The “adam”, “dummy-csr”, and “programs” folders provide an example of how results level metadata can be implemented within the define file for ADaM. The “adam” folder contains a sample Version 2.0 ADaM define.xml file, a sample XSL style sheet, as well as sample datasets and a sample reviewer’s guide. The “dummy-csr”, and “programs” folders contain a dummy Clinical Study Report and dummy programs that are used to show how external documents are referenced and can be linked to in the define file. It’s worth noting that the style sheet is a sample and its inclusion in a submission package is recommended but not required.

WHY ANALYSIS RESULTS METADATA: THE BUSINESS CASE

Since results-level metadata is not required, why invest the time and resources to enter and use it? Although not mandatory, it can add significant value to a submission. This was demonstrated in the updated CDISC Pilot Project [(2)], which showed that results-level metadata supplies the missing piece for end-to-end traceability from data collection to the analyses required by regulatory agencies. It documents results and provides the regulatory reviewer with the means to trace results back to their source documents (programs, datasets, and statistical analysis plans). From the results-level section of the define file, a reviewer can link to the datasets, programs, and other documentation used for a given result or analysis. This has the potential to save significant time for a regulatory reviewer and thus speed time to a drug’s approval.

Enhancing the reviewer experience and saving valuable time for regulatory review are reasons enough for creating results level metadata data, but there's more! Results level metadata not only adds value for a reviewer, but also provides value to sponsors and their agents. An earlier paper by the author (see "References," below) demonstrated that well-constructed metadata and metadata access tools can significantly improve the *creation* of the datasets and documents that comprise a submission. It outlined how metadata-driven applications and utilities can be integrated into standard business processes, speeding the production and improving the quality of deliverables.

The paper looked at how dataset and variable-level metadata is utilized throughout a clinical study. First, regardless of whether the datasets being created conform to CDISC or other proprietary standards, content specification is typically stored as machine readable metadata. This is vital to ensure that the standard is maintained and readily available throughout the life cycle of the study. Second, the metadata is extended to include programming specifications and other documentation for creating the datasets. Third, the same metadata is used as input for the programs that create clinical and analysis datasets. Fourth, the datasets produced are checked against the metadata to make sure that they are standard-complaint. Fifth, the metadata is enhanced so that it includes all of the information needed to produce documentation (aka the "define file") for the submission's datasets. Finally, the metadata can be used to produce specifications or reports in a variety of formats for a variety of audiences. In summary, dataset and variable metadata is used as a single source for efficiently documenting and creating datasets, as well as providing the traceability of the data from collection to submission required by regulatory agencies.

HOW: IMPLEMENTATION STRATEGIES

We see, then, that ARM "tells the story" of a display and that there are significant benefits to collecting and assembling the pieces of the story. Let's start to look at the XML creation process, starting with a Table, shown in **Figure 1**, taken from the CDISC release package.

Figure 1: HTML Display of Results Metadata Table

Table 14-3.01 Primary Endpoint Analysis: ADAS-Cog - Summary at Week 24 - LOCF (Efficacy Population)
Dose response analysis for ADAS-Cog changes from baseline
Pairwise comparisons to placebo for ADAS-Cog changes from baseline

Analysis Results Metadata (Detail) for Study CDISC-Sample

Table 14-3.01

Display	Table 14-3.01 Primary Endpoint Analysis: ADAS-Cog - Summary at Week 24 - LOCF (Efficacy Population)
Analysis Result	**Dose response analysis for ADAS-Cog changes from baseline**
Analysis Parameter (s)	PARAMCD = "ACTOT" (Adas-Cog(11) Subscore)
Analysis Variable (s)	CHG (Change from Baseline)
Analysis Reason	SPECIFIED IN SAP
Analysis Purpose	PRIMARY OUTCOME MEASURE
Data References (incl. Selection Criteria)	ADQSADAS [PARAMCD = "ACTOT" and AVISIT = "Week 24" and EFFFL = "Y" and ANL01FL = "Y"]
Documentation	Linear model analysis of CHG for dose response; using randomized dose (0 for placebo; 54 for low dose; 81 for high dose) and site group in model. Used PROC GLM in SAS to produce p-value (from Type III SS for treatment dose). SAP Section 10.1.1
Programming Statements	[SAS version 9.2] `proc glm data = ADQSADAS;`

Define.xml, made viewable by an XSL style sheet, contains a wealth of information about the table. Its title, dataset references, data filter, documentation, and programming statements are clearly presented. Links to the display and to the relevant portion of the SAP are also provided. Everything about the Table is presented in a single place. There is no need to navigate to the TFL directory or manually open the PDF containing the SAP.

But how did we get to this point? Having seen that the display's "story" is a valuable one worth telling, we

now focus on the raw materials for the story: the metadata database, a means to populate the database, and tools to access the metadata for XML and other purposes.

THE RESULTS DATABASE

Metadata requirements for Results metadata are clearly identified in the CDISC specification document and the XML schema that is part of the CDISC release package. In this section, we summarize key portions of the specification and identify some of the design decisions that need to be made for implementation.

Figure 2, below, describes metadata that needs to be populated for each display. The metadata is described in general terms and avoids XML nomenclature. Note that there are some items such as object identifiers ("OIDs") that can be programmatically generated. Since the table's focus is on fields that would have to be manually entered, auto-generated items are not included in the table.

Figure 2: Summary of ARM Requirements

Level	Item	Required?
Display	Unique display name-number	Yes
Result	Title of the Display	Yes
Result	Analysis reason	Yes
Result	Analysis purpose	Yes
Result	Description	Yes
Result	Reference to external documentation	No
Result	Description of how to join multiple datasets used	Conditional
Result	Documentation (in-line text)	No
Result	Documentation (reference to external document)	No
Result	Programming code	No
Result	Software reference (e.g. "SAS Version 9.3")	No
Result	Link to the program that created the Result	No
Result-dataset	Name of dataset used for the Result	Yes
Result-dataset	Dataset filter (WHERE clause) for PARAMCD-based Result	Conditional
Result-dataset	Variables used to create the analysis result	Conditional

At first blush, this does not appear that demanding. Items display type, number, title, and dataset would have likely been collected even in the absence of results metadata. New items such as documentation references are straightforward. Even though the collection of some of these items doesn't appear to present demands, there are a few things to consider:

- **Repetition.** The sheer number of times this information needs to be entered into the database can become problematic. A sponsor choosing to fully populate ARM for a study with "n" displays creates the scenario where this seemingly small amount of metadata is entered "n" times.
- **Hierarchy.** Items at the Display level are entered only once. Since a display can have multiple Results, anything in **Figure 2** beginning with "Result" would need to be entered for *each* Result. Consider, too, that multiple datasets can be used for multiple Results (a worst-case scenario, to be sure, but one that has to be designed for) and the metadata entry burden increases significantly.

These challenges are not unique to Results metadata. *Any* hierarchical database and *any* scenario that involves populating a high volume of metadata presents similar challenges. Repetition-based issues are addressed later, in "Interface." The remainder of this section presents some design criteria that can be applied to the hierarchical nature of the metadata.

Let's summarize **Figure 3**, emphasizing groups of data that are repeated (cardinality is indicated in boldface).

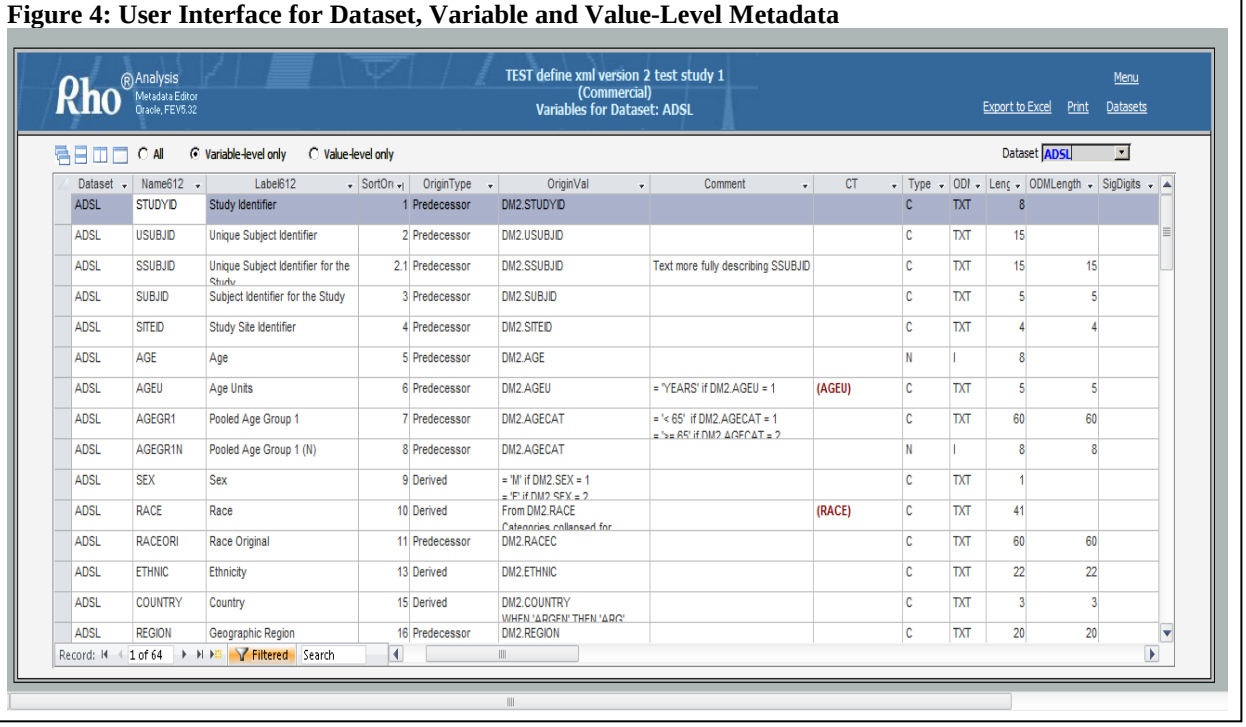
Figure 3: ARM Hierarchy and Cardinality

```
Display description, documentation [1]
      Result [1 or more]
            Description [1]
            Datasets and variables [1 or more]
            Documentation [1]
            Programming code [1]
```

The database design needs to allow for multiple results within a display and multiple datasets within a result. From a design perspective, these requirements are most readily met by creating multiple tables: display, result, and datasets used. That said, we also know from experience that any metadata-based system requires transparent access by applications.

ARM implementation is no exception. Our Oracle tables are made available to applications by views that essentially "flatten" the tables to one row per result. The views relieve the application programmer of the

Figure 4: User Interface for Dataset, Variable and Value-Level Metadata



burden of study-level filtering, display-result-dataset table joins and other tasks. This strategy allows applications to focus on creation of compliant XML and, as we'll see later, facilitates other uses of the metadata.

POPULATING THE DATABASE

As noted earlier, the additional metadata required for even minimal ARM compliance can become burdensome if there are a significant number of displays and/or results. If metadata entry cannot be made into a *joyful* experience, it can, at least, be a *tolerable* one. This is why the user interface is critical.

Rho has used a Microsoft Access front end to an Oracle database since 2008 for entry of "traditional" metadata: dataset, variable and value-level items used for creating define.xml and other, non-eSub tasks. Both the interface and database are home-grown. The interface is intuitive, configurable, and provides numerous navigational tools. It also has the benefit of customization for company and sponsor-specific demands (most notably, we were able to create a metadata repository that allows the user to seed a study with some or all of another study's metadata). These benefits, however, impose maintenance and

development costs. **Figure 4**, above, shows the variable-level metadata for a study.

This database and front end would seem to be the logical candidates for modification to accommodate the needs of results metadata. This was not the case. Another system, one used for project management,

Figure 5: Project Tracker Database

Name	Number	Task	Status	Owner	Created	Started	Completed	Next Due Date	Instructions
TAA_DS_01	14.1.1	Create Program	Completed	kreece	08-Oct-2012 09:37:36 AM	11-Oct-2012 11:12:46 AM	11-Oct-2012 11:12:47 AM		
TAA_DS_01	14.1.1	Create Validation Program	Completed	pnguyen	08-Oct-2012 09:37:36 AM	16-Oct-2012 04:24:23 PM	16-Oct-2012 04:24:23 PM		
TAA_DS_01	14.1.1	Validate Cosmetics	Completed (Pass)	rwoolson	08-Oct-2012 09:37:36 AM	06-Dec-2012 09:50:05 AM	06-Dec-2012 09:50:18 AM		
TAA_DS_01	14.1.1	Validate Statistics	Completed (Pass)	pnguyen	08-Oct-2012 09:37:36 AM	19-Oct-2012 10:53:05 AM	19-Oct-2012 03:52:05 PM		
TAA_DS_01	14.1.1	QC	Superseded		08-Oct-2012 09:37:36 AM				
TAA_DS_01	14.1.1	Revise Program(1)	Completed	kreece	21-Nov-2012 12:39:13 PM	04-Dec-2012 09:42:05 AM	04-Dec-2012 09:42:05 AM		Update so that all subjects in the ‘Not Randomized’ column show as discontinued. For all columns, calculate reason for discon % using discontinued as denominator
TAA_DS_01	14.1.1	Verify Revision(1)	Completed (Pass)	pnguyen	21-Nov-2012 12:39:13 PM	06-Dec-2012 09:43:05 AM	06-Dec-2012 09:43:05 AM		Update so that all subjects in the ‘Not Randomized’ column show as discontinued. For all columns, calculate reason for discon % using discontinued as denominator
TAA_DS_01	14.1.1	Validate Cosmetics(1)	Completed (Pass)	rwoolson	21-Nov-2012 12:39:13 PM	06-Dec-2012 09:50:05 AM	06-Dec-2012 09:50:18 AM		Update so that all subjects in the ‘Not Randomized’ column show as discontinued. For all columns, calculate reason for discon % using discontinued as denominator
TAA_DS_01	14.1.1	Validate Statistics(1)	Completed (Pass)	pnguyen	21-Nov-2012 12:39:13 PM	06-Dec-2012 09:43:08 AM	06-Dec-2012 09:43:08 AM		Update so that all subjects in the ‘Not Randomized’ column show as discontinued. For all columns, calculate reason for discon % using discontinued as denominator
TAA_DS_01	14.1.1	QC(1)	Completed (Pass)	rwoolson	21-Nov-2012 12:39:13 PM	06-Dec-2012 09:50:28 AM	06-Dec-2012 09:50:36 AM		Update so that all subjects in the ‘Not Randomized’ column show as discontinued. For all columns,

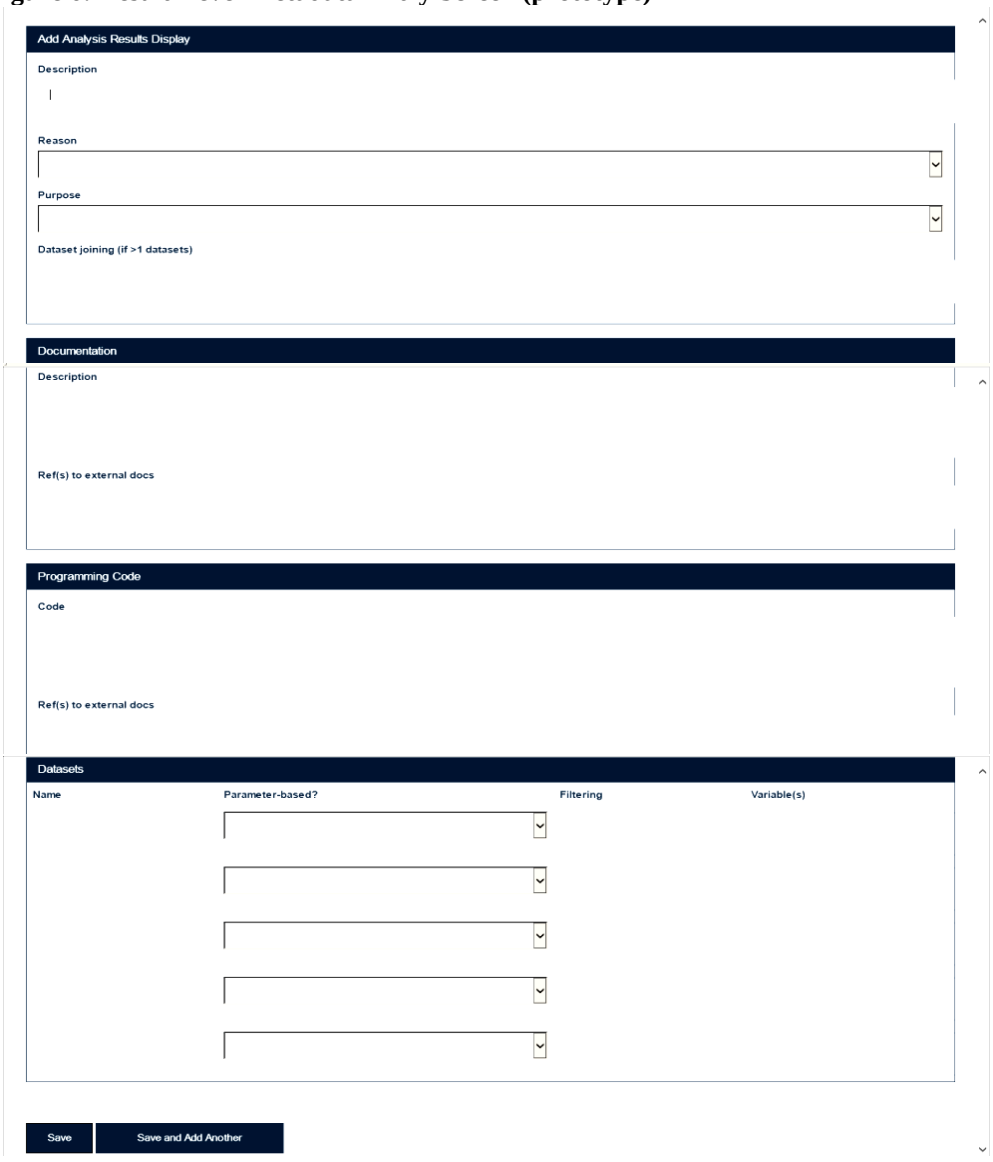
was already being used and contained some of the items needed for ARM compliance. Some of the items for table-specific task tracking are shown in **Figure 5**, above.

New ARM-related tables and views were added to the underlying Oracle database. The user interface was also modified. This was more labor-intensive than expected due to the hierarchical nature of the metadata. The project tracker design needed to be expanded to accommodate multiple instances of results and datasets per display. A screen for the enhanced interface is shown below, in **Figure 6**, next page.

It’s notable that the look and feel of the screen, as well as navigation between screens, differs from the interface used for building SDTM and ADaM datasets. This dissonance is dampened somewhat by the history of both systems’ usage. Statisticians and statistical programmers were familiar with both tools; the addition of fields and tabs in the project tracker was just viewed as another enhancement, and with proper training was not problematic (this is not to say that the additional entry of results metadata was met with unbridled enthusiasm!). A “bulk loading” method to populate the Oracle tables from other, sometimes client-supplied sources is also available.

The evolution of the dual-interface system at Rho has, so far, meant that the result-level metadata is missing one popular piece of functionality that is in the dataset/variable system: a repository. Conceptually, it would be similar to what is being used for datasets and variables. The user could import Result-level metadata in either of two ways. First, the user could seed a study's metadata by importing from another study. This approach makes sense when “study current” is nearly identical to “study previous.” Second, the user could seed from a “standard results” library. This is similar to Rho's TFL library, which is a collection of program shells for standard TFLs. The standard results library would work in concert with the TFL library: when the user selects a TFL shell, there would be an option to populate that study's results-level metadata with values for that display. Display type, a generic title, subpopulations, ADaM dataset name and other fields are candidates for seeding.

Figure 6: Result-Level Metadata Entry Screen (prototype)



BUILDING THE XML

We've discussed how the metadata are structured and the importance of a solid, extensible user interface to use when populating the database. The remaining task is to construct the XML. This section describes some strategies for building define.xml. It briefly describes XML syntax and structure and then focuses on how to construct a file that is compliant with the CDISC ARM schema.

XML Basics

XML is stored in a plain-text file with the file extension XML. It is an ordered collection of items bounded by angle brackets (< and >). One of its strengths is that XML is that it is pure content, uncluttered by the markup tags found in HTML. Just as a SAS® dataset can be transformed into different reports and graphs, so too can an XML file. XML file content is just that – content, with no mechanism for controlling display. The transformation into HTML is performed by XSL style sheets, which will be discussed later.

Let's look at **Figure 7**, a snippet of XML code taken from the CDISC release package:

Figure 7: XML Describing Result Analysis Datasets

```
<arm:AnalysisDatasets def:CommentOID="COM.JOIN-ADSL-ADAE">
  <arm:AnalysisDataset ItemGroupOID="IG.ADAE">
    <def:WhereClauseRef WhereClauseOID="WC.Table_14-5.02.R.1.ADAE" />
    <arm:AnalysisVariable ItemOID="IT.ADAE.AEBODSYS"/>
    <arm:AnalysisVariable ItemOID="IT.ADAE.AEDECOD"/>
  </arm:AnalysisDataset>
  <arm:AnalysisDataset ItemGroupOID="IG.ADSL">
    <def:WhereClauseRef WhereClauseOID="WC.Table_14-5.02.R.1.ADSL"/>
  </arm:AnalysisDataset>
</arm:AnalysisDatasets>
```

This is where the similarity to SAS datasets ends. While a SAS dataset is rectangular, a typical reasonably complex XML file is a **hierarchy**, where **elements** (items within the <>), some of which have **attributes** (items denoted by name="value") are nested within each other. The following are important to keep in mind when building *any* XML file:

- **White space doesn't matter.** Blanks are, with few exceptions, ignored unless they are part of an attribute value. This allows the XML writing application to insert seemingly extraneous blanks in order to indent elements. Since define.xml is constructed as a hierarchy of elements, white space can be effectively used to highlight the hierarchy. While this visual emphasis isn't required, it is easy to do and is very helpful during the inevitable debugging. Likewise, comments in the XML are just that – items that don't contribute to the information in the XML. But they are extremely useful for debugging: our standard practice when writing a group of elements is to insert comments that identify where in the define program the element was written. This is invaluable when trying to locate the source of a malformed or otherwise problematic element.
- **Case matters.** <arm:AnalysisDatasets> and <arm:analysisdatasets> are almost identical, but there are few places where "almost" counts for as little as it does in XML. Case must exactly match what is defined in the schema. Anything less results in a file that cannot be validated.
- **Elements must start and end.** Most browsers allow lack of closure of an HTML command (e.g., <p>). This isn't the case with XML; elements must be ended clearly. <arm:AnalysisDatasets> must eventually be balanced out with </arm:AnalysisDatasets>.
- **Element order matters.** Elements must be formed correctly and must appear in the correct location within the XML. In the example, <arm:AnalysisDatasets> precedes <arm:AnalysisDataset> and <def:WhereClauseRef>. Element termination must reflect the order in which they were defined (elements "a", "b", and "c" must be terminated in order "c", "b", and "a"). The correct order can be gleaned from a number of sources, among them the CDISC define-xml version 2 reference and the schema files.

Constructing the XML File

There are many viable and robust ways to build the XML. If you already create define.xml, the toolset currently in use is likely going to be the one used to create the ARM elements. If creating define.xml is new to your organization, your choices are more wide open.

Rho has been building define.xml for its clients since 2006. The date is important. As was the case with the database and user interface, there were few off-the-shelf tools at the time to meet our needs. Being a CRO, our requirements for flexibility and responsiveness to clients places even more demands on the XML tools (e.g., be able to link from dataset to variable-level tables from the dataset *description* rather than the dataset *name*). The home-grown XML generator was, and remains, a suite of SAS macros that massage the metadata and then use DATA steps and PUT statements to write the XML. It's an Old School solution, to be sure. It is also completely flexible, able to easily react to both client requests and changes to the XML schema.

Regardless of the tool used to write the XML, there are some software-agnostic points to be made about the process:

- **Consistent OID formation is critical.** Some results metadata elements refer to other locations in define.xml. In **Figure 7**, above, for example, element arm:AnalysisVariable refers to element ItemOID IT.value ADQSADAS.CHG. The use of OIDs (Object IDentifiers) is pervasive in the define file, and is used by the style sheet to navigate between elements. Their naming and casing are

arbitrary, but must be used consistently (you could assign an OID of Item.ds1.chg rather than IT.ADQSADAS.CHG, for example). Whatever the naming convention, it must be used consistently throughout the XML in order for the style sheet to transform the XML and for the XML to validate.

- **Exploit commonality.** Some elements are written from multiple locations in the XML program. The Description element, for example, can: define a TFL tile; describe a Result's contents; and identify a variable label. Similar scenarios where one or more elements can be written by generalized code are: comment definitions, “leaf” identifiers (references to external documents), and WHERE clause references. Rather than duplicate the often tedious coding in every location from which they are written, it's far easier to abstract the code and (assuming we're in the SAS world) write a parameterized macro to do the work.

Making the XML Viewable: The Style Sheet

A valid define.xml is simply a collection of elements and attributes recognized by the schema and populated with values from the metadata database. Without a means of making it easily readable, the file's worth is greatly diminished. The resource that transforms the XML into navigable HTML is define.xsl (E**x**tensible **S**tylesheet **L**anguage). The XSL searches for elements (study-level, dataset, variable, value-level, results, and others) and inserts their values into HTML tags. Suppose define.xml contains this line near the top of the file:

<?xml-stylesheet type=”text/xsl” href=”define.xsl” ?>

When define.xml is double-clicked, the typical default action is to open it in a browser. The browser will detect the presence of the style sheet and display the HTML that it generates. A discussion of XSL intricacies is beyond the scope of this paper, but two comments warrant inclusion here:

- An XSL file is, itself, formatted as an XML document. It is a member of a family of XML-related sublanguages that facilitate XML element-attribute identification (XPath) and transformations to complex file formats (XSL-FO). XSL is a functional (versus the more familiar imperative or object-oriented) language (think Lisp, SNOBOL). It's a lot to learn, and *strengthens the case for using the style sheet supplied by CDISC*. Sheets are available both for converting XML into browser-friendly HTML and into a PDF.
- With a little knowledge of the XPath sublanguage, the XSL can be a valuable resource for understanding the structure of the XML. This is especially helpful when trying to understand which element or attribute contributed to the HTML.

Did You Get It Right? Validation

Building define.xml using requires knowledge of XML syntax, the ARM schema, and the tools to create a file that is valid both syntactically (“are the elements in the correct order?”) and semantically (“is the content correct?”). *Syntax* is easy to check: badly formed XML results in errors when being opened in a browser or XML editor. *Content* is another matter. Validation requires that the elements cross-reference correctly (e.g., the variable OID in a result dataset element group is defined in dataset and variable-level elements, that use of controlled terminology is correct, and so on).

The validation tool of choice is the Pinnacle 21 validator (formerly, “OpenCDISC”). The “Community” version is open-source software distributed without charge by Pinnacle 21 LLC. Its suite of SDTM, ADaM and define.xml checks are endorsed by the FDA. Testing for valid define.xml includes: element consistency and presence/absence, cross-reference validity, and correct use of MedDRA and other external dictionaries. Current plans are for validation of results-related elements by quarter 3 of 2016.

EXTENDING RESULTS METADATA USAGE

Building the infrastructure to create define.xml with result-level metadata is challenging. The cost is more than offset by the ease of review and, in turn, the likely reduction of time to approval. As is the case with *all* metadata used by define.xml, there are other, “upstream” non-eSub uses. We have described other uses of result-level in earlier papers (see “References,” below) and will summarize them here.

The workflow for use of ARM for non-eSub tasks is similar to that used for define.xml: identify places

where the metadata can be used, create new, complementary fields if needed, and develop tools such as SAS macros to provide easy access to the metadata. One of the most obvious and easily implemented uses of result-level metadata is creating titles and footnotes for the TFLs. Display type, number, description, population and other subsetting criteria can be accessed via a macro and stored in one or more macro variables for use by the calling program. Likewise, footnotes (not part of the ARM specification), can be added to the metadata and accessed in a manner similar to titles.

Also, as noted earlier in "Populating the Database," we were able to add tables and fields for ARM compliance to an existing application. The Project Tracker is used to manage the production of deliverables such as datasets and TFLs. Progress toward item (dataset/TFL) completion can be monitored and commented upon. Some ARM fields were present in the Tracker prior to the 2015 ARM specification. Enhanced results metadata in the Tracker means the project manager can view more TFL detail. The interface enhancements allow the project manager to view summary or detailed levels of metadata for each display.

A quick glance at the history of our usage of other, non-ARM metadata suggests another way to repurpose the metadata resource and further strengthen the case for its adoption. Rho's proprietary metadata management system has been used for over 1,100 studies. The dataset, variable and related tables are a rich resource, and one that can be used across studies. It is not unusual for a project to have many studies that closely resemble each other. Enhancements to the system make it easy for a statistician tasked with populating study "next" with metadata from study "previous," thus saving a huge amount of metadata entry time.

It isn't a conceptual challenge to envision a similar repository for results metadata. A new study's ARM tables could be populated using a mechanism similar to that used for datasets, variables and other metadata found in the define-xml. The concept could be extended even further, with the source metadata being independent of a particular study. A common Adverse Events table, for example, could have its ARM tables populated from a study-independent repository. Regardless of its source, this bulk populating of results tables can create significant resource savings. As is the case with other re-use scenarios described above, all that is required is an institutional commitment to developing and maintaining the metadata infrastructure and the programming tools to access it.

CONCLUSION

The 2015 release of the Analysis Results Metadata (ARM) specification by CDISC finally provides the missing link for end-to-end standards and full traceability from data collection to analysis results. This benefits a number of stakeholders. Regulatory reviewers who used the CDISC Pilot Project define.xml and style sheet recognized the value of being able to easily trace results back to their source and to link to a wealth of additional result-related information. Results-level metadata also can deliver tremendous value to sponsors. It can be stored, queried, populated, and re-used for tasks throughout the life cycle of a project similar to the more familiar datasets and variables metadata.

At this point in time a commercial off-the-shelf product to manage and process results level metadata does not exist. However, organizations with experience building software to create define.xml should be able to apply the same development methodology to construct deliverables compliant with results level metadata ODM extension.

This paper has presented a summary of our experiences with database, interface and tool-building for ARM implementation. It is our belief results-level metadata provides an asset whose benefits far outweigh it design and development costs.

REFERENCES

(1) "Analysis Results Metadata" 2015. Online at http://www.CDISC.org/ADaM

(2) "Updated SDTM/ADaM pilot Package".2013. Online at www.cdisc.org/sdtm-adam-pilot-project

(3) DiIorio, Frank and Jeff Abolafia, "The Design and Use of Metadata: Part Fine Art, Part Black Art". 2006. Proceedings of the Annual SAS® Users Group International Conference.

(4) DiIorio, Frank and Jeff Abolafia, “From CRF Data to Define.XML: Going “End to End with Metadata”. 2007. Proceedings of the Pharmaceutical SAS Users Group Conference.

(5) Abolafia, Jeff and Frank DiIorio,“Managing The Change And Growth Of A Metadata-Based System”. 2008. Proceedings of the SAS Global Forum.

(6) Abolafia, Jeff, “The Missing Link: Results Level Metadata”. 2013. Proceedings of the Annual PhUSE Conference.

(7) Abolafia, Jeff, “Effective Use of Metadata in Analysis Reporting”. 2014. Proceedings of the Pharmaceutical SAS Users Group Conference.

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

Frank DiIorio
CodeCrafters, Inc.
Frank@CodeCraftersInc.com

Jeffrey Abolafia
Rho, Inc.
Jeff_Abolafia@rhoworld.com