PhUSE 2012

Paper TS01

Clinical Standards Toolkit 1.5 – How Do I Know My Metadata is Right?

Gene Lightfoot, SAS Inc., Cary, NC, USA

ABSTRACT

One of the key features in SAS® Clinical Standards Toolkit 1.5 (Toolkit) is the ability to validate the metadata used by the Toolkit for each supported standard. Currently when installing Toolkit, an installation qualification/output qualification (IQ/OQ) document is provided to ensure the user the product was installed properly. But as users modify files associated with each standard, add references to study-level files, and define and add their own custom standards, how can they be confident that all the component metadata is in sync for optimal Toolkit performance? This paper focuses on the two primary types of Toolkit metadata, Global Library metadata defining each standard and study-level metadata representing each study of interest, while also covering changes to the Toolkit in the form of revised metadata, new metadata, the addition of internal validation checks, and the process of running these checks.

INTRODUCTION

Prior to the release of 1.5, Toolkit relied on many assumptions that the users have expected metadata in a certain form and that metadata is consistent across multiple metadata files. However, with few exceptions, there was no systematic assessment of the validity of Toolkit metadata. The previous assessment was generally limited to:

- running the cstutil_checkds framework macro to assess the structure and content of the SASReferences data set (performed as a setup step running Toolkit sample drivers).

- requiring the specification (and existence) of various metadata files as parameters when calling selected Toolkit macros (e.g. cst_registerStandard requires the specification of valid standards and standardsasreferences data sets).

To further enhance Toolkit it needed to provide validation of the Toolkit metadata as part of the definition, registration, and maintenance of each standard and optionally as a setup step for each submitted Toolkit process. In addition, this internal validation uses the existing Toolkit validation framework used for specific standards such as CDISC-SDTM.

TOOLKIT INTERNAL VALIDATION OBJECTIVES AND USAGE SCENARIOS

The two primary objectives of the Toolkit Internal Validation functionality are:

- To make sure SAS-supplied Toolkit metadata files are consistent and correct
- To make the functionality available to users to help them validate custom standards

In developing Toolkit 1.5, the following usage scenarios were identified to describe the range of functionality desired by Toolkit Internal Validation:

- Support IQOQ assessment and reporting
- Support registration of a new standard and/or updates to an existing standard
- Assess metadata consistency across files
- Assess structural validity of a metadata file
- Assess content of a metadata file
- Validate a SASReferences data set
- Evaluate validation check metadata
- Perform a process pre-check
- Perform run-time process validation
- Evaluate code module header compliance

The expectation is that a mature Toolkit Internal Validation suite should be able to support each of the usage scenarios described above, unless information is inserted rejecting the scenario.

SUPPORT IQOQ ASSESSMENT AND REPORTING

Through the 1.4 release, the Toolkit IQOQ runs a representative set of standard-specific driver modules to illustrate that Toolkit is installed and configured satisfactorily. Toolkit Internal Validation is designed to make sure proper metadata is where

it belongs and contains what it is supposed to contain. Toolkit uses a subset of the Internal Validation tools and processes as the first step of the product IQOQ. This involves calling a series of Internal Validation tools to validate specific standards or files, and/or running a validation process against a subset of Internal Validation checks (e.g. validation_IQOQ.sas7bdat).

SUPPORT REGISTRATION OF A NEW STANDARD AND/OR UPDATES TO AN EXISTING STANDARD

When a new standard is registered, offer the option to perform Internal Validation on the metadata for that standard. Likewise, when the metadata for a registered standard is updated, optionally perform Internal Validation on the metadata for that standard. Both of these scenarios require that tools and checks be callable at the standard level.

ASSESS STRUCTURAL VALIDITY OF A METADATA FILE

Verify the structural validity of any given metadata file against a fixed reference, such as a template.

ASSESS METADATA CONSISTENCY ACROSS FILES

The interrelationship among Toolkit metadata assumes that information in certain files is consistent with information in other files. Three examples illustrate this:

- It is expected that the table.column pairs specified in the reference_columns data set have a matching record for each table in the reference_tables data set

- The metadata content in source_tables and source_columns is expected to be consistent with (though not necessarily match) the data sets found in the source data library. Toolkit would not expect, for example, a column documented in the source_columns data set that does not exist in the corresponding data set in the source library.

- Each validation check is expected to have a corresponding description in the messages data set.

Toolkit Internal Validation supports comparison of metadata between two files. A simple example is comparing dataset1.column1 against dataset2.column2. Toolkit Internal Validation also includes support of a full file comparison, such that one file may be deemed to be structurally equivalent with (or different from) another. There is no expectation that the content of one file be deemed to be equivalent to (i.e. a copy of) the content of another file.

ASSESS CONTENT OF A METADATA FILE

Toolkit checks that the content of any given metadata file meets expectations in terms of having the correct number of columns, expected number of rows, and the expected column values. This runs and reports on all metadata files, or by standard, or for any specified file.

VALIDATE A SASREFERENCES DATA SET

Toolkit has enabled this both as a runtime option (current CST 1.4 behavior using cstutil_checkds) and as a standalone option. In developing this process the following aspects were taken into consideration:

- Is the data set structurally correct? It may have optional, user-extended columns.
- Can the referenced input and output files and folders be reached?
- Does any required look-through to Global Library defaults work?
- Are all global macro variable's references valid and are values expected based on the assessment context?
- Is any derived format search path valid?
- Is any derived autocall path valid?
- Are all called macros reachable?

EVALUATE VALIDATION CHECK METADATA

The process of defining the metadata to support the running of a specific validation check in Toolkit is essential to proper functionality. The following Toolkit validation metadata issues must be correct for the check to run successfully:

- Is the checkid unique?
- Is the standard and standardversion information consistent with available, registered standards?
- Does the combination of fields that identify this as a unique check actually uniquely identify the check?
- Is the codesource (check macro) available?
- Are all utility macros and required macro variables found?
- If lookup information is provided, can the referenced data set or format catalog be found?
- Does the check status suggest the check is ready for use?
- Is any specified or referenced codelogic consistent with the requirements of the referenced check macro in codesource?

It is not expected that this assessment should be made at run time. However, it may be necessary that the calling validation process establishes the expected session "environment" (libraries allocated, format search path enabled, etc.) and that this evaluation occur for the purposes of validating each standard-specific validation_master data set before that master set of checks is used to validate study data.

PERFORM A PROCESS PRE-CHECK

This addresses the question "Are there any problems with my Toolkit process as currently defined?" That is, will my driver module run successfully? Theoretically, this can be part of an IQOQ pre-step before submission of any driver modules used as part of the IQOQ. This utility is most useful as an aid to customers building their own CST processes.

PERFORM RUN-TIME PROCESS VALIDATION

No matter how much "pre" validation has been performed, submission of any Toolkit process may include run-time changes in the code, macro calls, the "global macro environment", content of the process SASReferences data set, etc. Through Toolkit 1.4, this is the only "internal validation" performed – the specific call to cstutil_checkds which occurs as part of the call to cstutil_processsetup(). Run-time process validation will typically occur prior to and/or as a part of the call to the primary task macro (e.g. sdtm_validate). If problems are detected, default behavior is to abort the process, reporting the problems. Run-time validation can optionally be performed on any metadata files expected/required by that process.

EVALUATE CODE MODULE HEADER COMPLIANCE

Each SAS-supplied macro, tool, and driver module now conforms to an expected naming convention and commenting style in the header that conforms to a SASDOC standard so that the code interface documentation can be automated for Toolkit user documentation.

ADDITIONAL METADATA FOR TOOLKIT 1.5

To support some of the assessments listed above, several modifications were needed to existing Toolkit metadata to allow for internal validation. These modifications consist of additional required metadata. Including, but not limited to, additional information about a standard, whether a data set was considered input or output, location of key files, and Boolean style information (Y or N). The following modifications were incorporated into Toolkit to perform internal validation. Examples shown here access the default installation file paths, customer Toolkit installation paths may be different.

C:\cstGlobalLibrary\standards\cst-framework-1.5\control

STANDARDLOOKUP.SAS7BDAT

– Addition of 2 new columns *template* and *templatetype*. Template is used to identify an entity that can be used to populate the metadata for a file or data set. For example, this would name a blank data set (e.g. sasreferences) provided by Toolkit that would be used to compare the metadata against the populated Toolkit version of sasreferences to verify the metadata is correct.

Templatetype defines what kind of template this is. Current valid values are DATASET, FOLDER, CATALOG, and FILE.

STANDARDS.SAS7BDAT

– Addition of 3 new columns *studylibraryrootpath*, *controlsubfolder*, and *templatesubfolder.* Studylibraryrootpath identifies the root path for the sample study. Controlsubfolder identifies the folder relative to the standard rootpath that contains Toolkit control files (ie standardlookup, standards, etc.). Templatesubfolder identifies the template folder relative to the standard rootpath. These metadata are used to verify directories and files exist.

STANDARDMACROVARIABLES.SAS7BDAT

– New data set containing all of the global macro variables accessed by Toolkit and used to verify the existence of Toolkit global macro variables.

	Macro variable name	Macro variable label	Macro description	Is macro variable value required? (Y/N)	Is macrovalue case sensitive? (Y/N)	Value type(ANY,DISCRETE,INTEGER,DATASET)	Source file for macro
1	_cstCheckSortOrder	Specifies the order in which validation checks are to be run	This variable enables specification of the order in which the checks are to be run. The _DATA_ value indicates that checks are to be processed in the order defined in the Validation Control data set. You can specify a set of space-delimited keys from Validation Control columns (for example, checksource checkid).	N	N	ANY	validation
2	_cstColumnMetadata	Data set containing column-level metadata supporting validation	Data set that is used during processing that contains column-level metadata (derived from either the reference or study column metadata) that is used by the process.	Y	N	DATASET	initialize

STANDARDMACROVARIABLESDETAILS.SAS7BDAT

– New data set containing all of the valid values for the global macro variables accessed by Toolkit and used to verify Toolkit global macro values are correctly initiated.

	Macro variable name	Macro variable value	Macro variable value label	Is this value the default? (Y/N)
1	_cstCheckSortOrder	_DATA_	Use order defined in the Validation Control data set	Y
2	_cstColumnMetadata	work._cstcolumnmetadata		Y
3	_cstMetrics	0	Off	N
4	_cstMetrics	1	On	Y
5	_cstMetricsCntNumBadChecks	0	counter initialized to 0	Y
6	_cstMetricsCntNumChecks	0	counter initialized to 0	Y

STANDARDSASREFERENCES.SAS7BDAT

– Addition of 3 new columns *IOtype*, *filetype,* and *allowoverwrite*. IOtype defines whether or not the entity is input, output, or both. Filetype describes what the entity is and can be folder, dataset, catalog, or file. Allowoverwite informs Toolkit if the entity can be written to (Y) or not (N).

C:\cstGlobalLibrary\standards\cst-framework-1.5\messages

MESSAGES.SAS7BDAT

– Addition of new messages for handling Toolkit internal validation. These messages are prefaced with CSTV (eg cstv001).

	Result identifier	Standard version	Source of check	Record identifier used by checksource	Severity of check	Message text
82	CSTV001	***	CST	CSTV001	Warning	Multiple records detected for standard &_cstParm1
83	CSTV002	***	CST	CSTV002	Error	Standard is defined as XML but location of import XSL is missing
84	CSTV003	***	CST	CSTV003	Error	Standard is defined as XML but location of export XSL is missing
85	CSTV004	***	CST	CSTV004	Error	Standard is defined as XML but location of XML schema is missing

C:\cstGlobalLibrary\standards\cst-framework-1.5\programs

VALIDATION.PROPERTIES

– New file containing global macro variable settings for internal validation.

```
_cstTableMetadata=work._csttablemetadata
_cstColumnMetadata=work._cstcolumnmetadata
_cstCheckSortOrder=_DATA_
_cstMetrics=0
_cstMetricsDS=work._cstmetrics
_cstMetricsNumSubj=0
_cstMetricsNumRecs=0
_cstMetricsNumChecks=0
_cstMetricsNumBadChecks=0
_cstMetricsNumErrors=0
_cstMetricsNumWarnings=0
_cstMetricsNumNotes=0
_cstMetricsNumStructural=0
_cstMetricsNumContent=0
_cstMetricsCntNumSubj=0
_cstMetricsCntNumRecs=0
_cstMetricsCntNumChecks=0
_cstMetricsCntNumBadChecks=0
_cstMetricsCntNumErrors=0
_cstMetricsCntNumWarnings=0
_cstMetricsCntNumNotes=0
_cstMetricsCntNumStructural=0
_cstMetricsCntNumContent=0
_cstMetricsTimer=0
_cstStrictValidation=0
```

C:\cstGlobalLibrary\standards\cst-framework-1.5\templates

MESSAGES.SAS7BDAT

– New metadata template data set for messages data set. Contains structural content of the messages data set such as columns, label, lengths, etc., but contains no data for the messages data set (0 obs). Is used to verify structure.

STANDARDMACROVARIABLEDETAILS.SAS7BDAT
– New metadata template data set for global macro details data set. Contains structural content of the standardardmacrovariabledetails data set such as columns, label, lengths, etc., but contains no data (0 obs). Is used to verify structure.

STANDARDMACROVARIABLES.SAS7BDAT
– New metadata template data set for global macro variables data set. Contains structural content of the standardardmacrovariables data set such as columns, label, lengths, etc., but contains no data (0 obs). Is used to verify structure.

VALIDATION_STDREF.SAS7BDAT
– New metadata template data set for validation standard references data set. Contains structural content of the validation_stdref data set such as columns, label, lengths, etc., but contains no data (0 obs). Is used to verify structure.

C:\cstGlobalLibrary\standards\cst-framework-1.5\validation\control
New folder containing Toolkit internal validation validation_master data set.

VALIDATION_MASTER.SAS7BDAT
– New data set containing all of the internal validation checks for Toolkit.

	Validation check identifier	Standard model	Severity of check	Category of check	SAS macro module name	Domains/data sets to which check applies	Columns to which check applies	Code logic used within code
1	CSTV001	CST-FRAMEWORK	Warning	IQOQ	cstcheck_notunique	glmeta.standards	standard+mnemonic	
2	CSTV002	CST-FRAMEWORK	Error	IQOQ	cstcheck_columncompare	glmeta.standards	[isxmlstandard][importxsl]	data work._cstProblems;set &_cstDSName;if (upcase(&_cstColumn1) eq 'Y'' and &_cstColumn2 eq '');run;
3	CSTV003	CST-FRAMEWORK	Error	IQOQ	cstcheck_columncompare	glmeta.standards	[isxmlstandard][exportxsl]	data work._cstProblems;set &_cstDSName;if (upcase(&_cstColumn1) eq 'Y'' and &_cstColumn2 eq '');run;
4	CSTV004	CST-FRAMEWORK	Error	IQOQ	cstcheck_columncompare	glmeta.standards	[isxmlstandard][schema]	data work._cstProblems;set &_cstDSName;if (upcase(&_cstColumn1) eq 'Y'' and &_cstColumn2 eq '');run;

ADDITIONAL TOOLKIT TOOLS

In addition to new metadata, new tools were also identified and developed to help with Toolkit internal validation. These tools are all written as macros and therefore, in keeping with the core requirement of Toolkit, are open source to all customers. The following table itemizes a sampling of the new tools that generally can be run stand-alone with a single macro invocation. These tools can be called by each other or as a part of a Toolkit process that performs some larger function (e.g. validation, IQOQ). This paper was authored prior to final approval and development of the tools listed below, as such, some tools may not appear in the final product or new unlisted tools may be developed in the interim.

Tool / Description	**Code****
Structural file comparison	cstutilcomparestructure(_cstReturn=_cst_rc, _cstReturnMsg=_cst_rcmsg, _cstBaseDSName=, _cstCompDSName=, _cstResultsDS=work._cstCompareStructure)
Find a referenced file Macro confirms presence and validity of referenced external file (supported file types: FOLDER, DATASET, CATALOG, FILE)	cstutilfindvalidfile(_cstfiletype=, _cstfilepath=, _cstfileref=)
Can a file be created	cstutilcheckwriteaccess(_cstfiletype=, _cstfilepath=, _cstfileref=)
Validate SASReferences This is a replacement for current cstutil_checkds macro.	cstutilvalidatesasreferences (_cstDSName=&_cstSASRefs, _cstStandard=CST-FRAMEWORK, _cstStandardversion=1.2, _cstSASRefsGoldStd=, _cstallowoverride=, _cstResultsType=LOG, _cstPreAllocated=N, _cstVerbose=N)

Tool / Description	Code**
Evaluate global macro variables	cstutilcheckglobalmvars(filepath=, fileref=, macrovar=)
Assess metadata file content Metadata file has the correct number of columns, expected number of rows, and the expected column values.	cstutilevaluatefilecontent(filepath=, fileref=, targetds=, referenceds=)
Evaluate check metadata Evaluate a single check metadata data set (e.g. validation_master) record	cstutilevaluatecheckmetadata(_cststd=CDISC-SDTM, _cststdver=3.1.2, _cstcheckds=, _cstchkid=)
Is xml file valid Input of this macro would be an XML file and a schema that the XML file should validate against.	cstutilcheckxmlfile(xmlfilepath=, xmlfileref=, schemapath=, schemaref=)
Parse SAS Log for problems	cstutilparselog(logtype=Current|File, filepath=, fileref=)
Summarize a CST results data set No errors or warnings noted in the results data set.	cstutilsummarizeresults(dstype=Current|File, resultsds=)
Process pre-check “Are there any problems with my CST process as currently defined?” That is, will my driver module run successfully?	cstutilchecksubmittedprocess(driverpath=, driveref=, checktype=Log|Results|Both)
Report installation status Generate a report that itemizes installation status, by standard, and provides an overall product status	cstutilreportinstallationstatus(_cststd=CDISC-SDTM, _cststdver=3.1.2)
Evaluate code module compliance	cstutilparsecodeheader(filepath=, fileref=)
This macro is called in the internal validation driver programs and builds a job stream for all registered standards passed to cstutilbuildstdvalidationcode() in the _cstStdDS parameter data set.	cstutilbuildstdvalidationcode(_cstStdDS=work._cstStandardsforIV, _cstSampleRootPath=_DEFAULT_, _cstSampleSASRefDSPath=_DEFAULT_, _cstSampleSASRefDSName=_DEFAULT_)
Builds the framework reference_tables and reference_columns data sets from a SASReferences data set .	cstutilbuildmetadatafromsasrefs(cstSRefsDS=work.stdvalidation_sasrefs, cstSrcTabDS=work.source_tables, cstSrcColDS=work.source_columns)

**macro names and signatures may change

THE TOOLKIT “INTERNAL VALIDATION” STANDARD

The Toolkit framework for 1.5 is handled as a standard that supports validation. It is registered and contains all of the metadata one would associate with any Toolkit standard undergoing validation and includes a sample library providing users with examples of data and sample programs. Notably absent In the section above describing additions and modifications to the Toolkit global standard library, is the metadata folder that stores the reference_tables and reference_columns data sets. Since internal validation is dynamic in the sense it can be run or modified at any time, and as new standards, studies, etc. are added by the customer, most of the metadata are generated into temporary work files when needed. The generated temporary tables include reference_tables, source_tables, reference_columns, and source_columns. Below is an example of a generated reference_tables data set.

VIEWTABLE: Work.Reference_tables

	sasref	table	label	path
1	CSTCNTL	STDVALIDATION_SASREFS	SASReferences: Validation of standard	&studyRootPath/control
2	CSTLKUP	STANDARDLOOKUP	CST Metadata Lookup	&_cstGRoot./standards/cst-framework-1.5/control
3	CSTMETA	STANDARDLOOKUP	CST Metadata Lookup	&_cstGRoot./standards/cst-framework-1.5/control
4	CSTMETA	STANDARDS	CST-FRAMEWORK standard metadata	&_cstGRoot./standards/cst-framework-1.5/control
5	CSTMETA	STANDARDSASREFERENCES	CST-FRAMEWORK standard sasreferences	&_cstGRoot./standards/cst-framework-1.5/control
6	CSTMSG	MESSAGES	CST messages	&_cstGRoot./standards/cst-framework-1.5/messages
7	CSTRCNTL	VALIDATION_MASTER	Validation Checks, Master Superset	&_cstGRoot./standards/cst-framework-1.5/validation/control
8	GLMETA	STANDARDS		&_cstGRoot/metadata
9	GLMETA	STANDARDSASREFERENCES		&_cstGRoot/metadata
10	LOOKUP	STANDARDLOOKUP	Standard-specific Metadata Lookup	&_cstGRoot./standards/cdisc-adam-2.1-1.5/control
11	MESSAGES	MESSAGES	ADAM messages	&_cstGRoot./standards/cdisc-adam-2.1-1.5/messages
12	REFCNTL	VALIDATION_MASTER	Validation Checks, Master Superset	&_cstGRoot./standards/cdisc-adam-2.1-1.5/validation/control
13	REFMETA	CLASS_COLUMNS	Reference Standard Class Column Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/metadata
14	REFMETA	CLASS_TABLES	Reference Standard Class Table Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/metadata
15	REFMETA	REFERENCE_COLUMNS	Reference Standard Column Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/metadata
16	REFMETA	REFERENCE_TABLES	Reference Standard Table Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/metadata
17	SRCCNTL	STDVALIDATION_SASREFS	SASReferences: Validation of standard	&studyRootPath/control
18	SRCMETA	SOURCE_COLUMNS		&studyRootPath/metadata
19	SRCMETA	SOURCE_TABLES	Source Table Metadata	&studyRootPath/metadata
20	STDMETA	STANDARDLOOKUP	Standard-specific Metadata Lookup	&_cstGRoot./standards/cdisc-adam-2.1-1.5/control
21	STDMETA	STANDARDMACROVARIABLEDETAIL	Global Macro Variable Details Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/control
22	STDMETA	STANDARDMACROVARIABLES	Global Macro Variable Metadata	&_cstGRoot./standards/cdisc-adam-2.1-1.5/control
23	STDMETA	STANDARDS		&_cstGRoot./standards/cdisc-adam-2.1-1.5/control
24	STDMETA	STANDARDSASREFERENCES	Standard sasreferences for standard	&_cstGRoot./standards/cdisc-adam-2.1-1.5/control

In addition to being registered as a Toolkit standard, internal validation also has its own Toolkit Sample library. In these folders, users will find the validation_control data sets. Multiple validation control data sets have been created to anticipate different types of internal validations – Framework IQOQ, Standard IQOQ, and Standard metadata. Because different metadata exists in different areas within the Toolkit, internal validation specific sasreferences data sets are also included. New to Toolkit is the introduction of SAS views accessing the validation_control data set to subset out the different internal validation checks based on type.

TOOLKIT INTERNAL VALIDATION VALIDATION

During development, internal validation of Toolkit was broken into 3 areas:

- Global Library and Sample Library
- Standard reference metadata
- Standard source metadata

Global Library checks attempt to assure that the definition of standards within the CST Global Library is consistent. Two points of emphasis:

1) That the structure and content of the Global Library metadata folder standards, standardsasreferences, and standardlookup data sets is correct
2) That the metadata for each standard within the Global Library is complete and consistent with expectations.

Three values were added to the checktype variable in the validation_master data set. These allow the user to subset the validation_master based on the type of internal validation required. The three values are:

GLMETA – pertains to validation checks that are specific in nature to the Toolkit global library. Considered an IQOQ check.
STDIQOQ – Standard level IQOQ checks
STD – Standard level checks that are not considered part of an IQOQ.

Internal validation runs like any other Toolkit validation process. It uses the driver programs supplied with the product, but it also demonstrates the flexibility of Toolkit. This flexibility is seen in the creation of multiple driver programs to handle different types of validation (ie Global Library versus a Standard), multiple validation control views and/or data sets representing various subsets of the validation_master data set. Several internal validation driver programs are listed below.

VALIDATION DRIVER MODULES

In Toolkit, all driver modules are example SAS programs provided to illustrate Toolkit functionality. For Internal Validation, these driver modules are located in the <cstSampleLibrary>/cst-framework-1.5/programs folder. The following examples show actual code snippets and explain the workflow that is used by each driver.

VALIDATE_STANDARD.SAS

The objective of this driver module is to perform all standard-specific validation checks defined in validation_master where checktype NE 'GLMETA'.

The current driver workflow is described in the following steps:

a) Choose the standard(s) of interest and populate work._cstStandardsforIV with these standard(s):

```
*****************************************************************;
* User defines standard(s) of interest in the following data step *;
* Note exclusion of CST-FRAMEWORK standard.                       *;
*****************************************************************;

%cst_getRegisteredStandards(_cstOutputDS=work._cstAllStandards);

data work._cstStandardsforIV;
  set work._cstAllStandards (where=(
       (upcase(standard) = 'CDISC-ADAM'        and standardversion='2.1')
    or (upcase(standard) = 'CDISC-CRTDDS'      and standardversion='1.0')
/*
    or (upcase(standard) = 'CDISC-ODM'         and standardversion='1.3.0')
    or (upcase(standard) = 'CDISC-ODM'         and standardversion='1.3.1')
    or (upcase(standard) = 'CDISC-SDTM'        and standardversion='3.1.1')
    or (upcase(standard) = 'CDISC-SDTM'        and standardversion='3.1.2')
    or (upcase(standard) = 'CDISC-SEND'        and standardversion='3.0')
    or (upcase(standard) = 'CDISC-TERMINOLOGY' and standardversion='NCI_THESAURUS')
*/
 ));
 run;
```

b) Modify the standard validation SASReferences to point to the validation_control ***view*** of interest. SAS views are being used to make definition of the various check subsets more dynamic. If problems arise using SAS SQL views within or external to Toolkit (e.g., within CDI or SDD), Toolkit can revert to the use of physical SAS data sets.

```
*************************************************************************;
* Modify the sample SASReferences data set to point to the run-time     *;
* validation_control data set identifying the validation checks of interest. *;
*************************************************************************;

* libname below accesses c:/cstSampleLibrary/cst-framework-1.5/control *;
libname _cstTemp "&studyrootpath/control";

data work.stdvalidation_sasrefs;
  set _cstTemp.stdvalidation_sasrefs;
  if type='control' and subtype='validation' then
  do;
    filetype='view';
    memname='validation_control_std.sas7bvew';
  end;
run;
```

The view validation_control_std is one of several provided with Toolkit 1.5. The development code used to create these views is listed below:

```
*****************************************************************************;
*  Create the various internal validation validation_control data sets/views  *;
*****************************************************************************;

proc sql;
  create table cstcntl.validation_control
    as select *
    from refcntl.validation_master;

  create view cstcntl.validation_control_fw
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype)="GLMETA";

  create view cstcntl.validation_control_std
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype) in ("STD","STDIQOQ");

  create view cstcntl.validation_control_stdiqoq
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype) in ("STDIQOQ");
quit;
```

c) Call the process setup macro to perform all file and library allocations. The returned &_cstSASRefs data set contains fully resolved paths and memnames.

```
%cstutil_processsetup(
    _cstSASReferencesLocation=&workpath,
    _cstSASReferencesName=stdvalidation_sasrefs
    );
```

d) Re-create work.stdvalidation_sasrefs, setting _srcfile='FWVAL':

```
data work.stdvalidation_sasrefs;
  set &_cstSASRefs;
    attrib _srcfile format=$8. label='File source for record';

    ********************************************************************;
    * Framework validation sasreferences: cstcntl.stdvalidation_sasrefs *;
    ********************************************************************;
    _srcfile='FWVAL';
run;
```

e) Call the code-generator macro to build the job stream for each standard:

```
filename incCode CATALOG "work._cstCode.stds.source" LRECL=255;

%cstutilbuildstdvalidationcode(
    _cstStdDS=work._cstStandardsforIV,
    _cstSampleRootPath=_DEFAULT_,
    _cstSampleSASRefDSPath=_DEFAULT_,
    _cstSampleSASRefDSName=_DEFAULT_
    );
```

SAS Code is generated by the macro above for each standard that is specified in the _cstStdDS parameter. In this example the 2 standards are ADaM and CRT-DDS. Once this code is created it is then %included and the following workflow is persisted.

1. Initialize work._cstTempSASRefDS to accumulate SASReferences records from all standards of interest for later use by cstvalidate().

2. Look for standard-specific standardSASReferences data set from the Global Library. If found, run cstutil_processsetup() using this file.
3. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STD'.
4. Look for standard-specific sdtvalidation_sasrefs data set from the Sample Library. If found, run cstutil_processsetup() using this file.
5. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STUDY'.
6. Remove any duplicate records from work._cstTempSASRefDS (keys=standard standardversion type subtype).
7. Run %cstutilbuildmetadatafromsasrefs(cstSRefsDS=work._cstTempSASRefDS, cstSrcTabDS=work.source_tables, cstSrcColDS=work.source_columns). This macro dynamically builds reference_tables and reference_columns data sets from a SASReferences data set.
8. Set _cstSASRefs=work._cstTempSASRefDS – the cumulative ready-to-go SASReferences data set
9. Call cstvalidate(), which uses the validation_control_std view from b) above
10. Remove standard-specific records from work._cstTempSASRefDS, in anticipation of appending new records for the next standard to the framework records that remain.

f) Repeat steps 1-10 as described above for *each* standard selected in a) above, in this case ADaM and CRT-DDS.

g) Results are collated in cstrslt.validation_results. An excerpt of process results is provided in the following figure:

	resultid	checkid	resultseq	seqno	srcdata	message	resultseverity
5	CST0200		1	0	CDISC-ADAM 2.1	PROCESS WORKFLOW: Validating CDISC-ADAM 2.1	Info
6	CST0102		1	1	CST_CREATEDSFROMTEMPLATE	work._cstTempStdSASRefDS was created as requested	Info
7	CST0200		1	1	CSTUPDATESTANDARDSASREFS	SASReferences data set was successfully validated	Info
8	CST0200		1	2	CSTUTIL_ALLOCATESASREFERENCE	SASReferences data set was successfully validated	Info
9	CST0108		1	1	CST_SETPROPERTIES	The properties were processed from the PATH c:/cstGlobalLibrary/standards/cdisc-adam-2.1-1.5/programs/initialize.properties	Info
10	CST0108		1	1	CST_SETPROPERTIES	The properties were processed from the PATH c:/cstGlobalLibrary/standards/cdisc-adam-2.1-1.5/programs/report.properties	Info
11	CST0108		1	1	CST_SETPROPERTIES	The properties were processed from the PATH c:/cstGlobalLibrary/standards/cdisc-adam-2.1-1.5/programs/validation.properties	Info
12	CST0200		1	1	CST_INSERTSTANDARDSASREFS	SASReferences data set was successfully validated	Info
13	CST0200		1	2	CSTUTIL_ALLOCATESASREFERENCE	SASReferences data set was successfully validated	Info
14	CST0108		1	1	CST_SETPROPERTIES	The properties were processed from the PATH c:/cstGlobalLibrary/standards/cdisc-adam-2.1-1.5/programs/initialize.properties	Info
15	CST0108		1	1	CST_SETPROPERTIES	The properties were processed from the PATH c:/cstSampleLibrary/cdisc-adam-2.1-1.5/sascstdemodata/programs/validation.properties	Info
16	CST0200		1	3	CSTUTILBUILDMETADATAFROMSAS	Reference metadata was successfully derived from work._cstTempSASRefDS	Info
17	CST0200		1	1	CSTVALIDATE	PROCESS STANDARD: CST-FRAMEWORK	Info
18	CST0200		1	2	CSTVALIDATE	PROCESS STANDARDVERSION: 1.2	Info
19	CST0200		1	3	CSTVALIDATE	PROCESS DRIVER: validate_standardmetadata.sas	Info
20	CST0200		1	4	CSTVALIDATE	PROCESS DATE: 2012-08-28T16:28:17	Info
21	CST0200		1	5	CSTVALIDATE	PROCESS TYPE: VALIDATION	Info
22	CST0200		1	6	CSTVALIDATE	PROCESS SASREFERENCES: c:/cstSampleLibrary/cdisc-adam-2.1-1.5/sascstdemodata/control/stdvalidation_sasrefs.sas7bdat	Info
23	CST0200		1	7	CSTVALIDATE	PROCESS VALIDATION CONTROL DATA SET: c:/cstSampleLibrary/cdisc-adam-2.1-1.5/sascstdemodata/control/validation_control_std.sas7bvew	Info
24	CST0200		1	8	CSTVALIDATE	PROCESS STUDYROOTPATH: c:/cstSampleLibrary/cdisc-adam-2.1-1.5/sascstdemodata	Info
25	CST0200		1	9	CSTVALIDATE	PROCESS GLOBALLIBRARY: c:/cstGlobalLibrary	Info
26	CST0200		1	10	CSTVALIDATE	PROCESS STUDYLIBRARY: c:/cstSampleLibrary	Info
27	CST0200		1	11	CSTVALIDATE	PROCESS CSTVERSION: 1.5	Info
28	CST0200		1	10	CSTVALIDATE	PROCESS CONTROLLED TERMINOLOGY SOURCE: c:/cstGlobalLibrary/standards/cdisc-terminology-1.5/cdisc-adam/current/formats/cterms	Info
29	CST0100	CSTV260	1	1	REFMETA.CLASS_TABLES	No errors detected in REFMETA.CLASS_TABLES	Info
30	CST0100	CSTV260	1	2	REFMETA.REFERENCE_TABLES	No errors detected in REFMETA.REFERENCE_TABLES	Info
31	CST0100	CSTV260	2	1	REFMETA.CLASS_COLUMNS	No errors detected in REFMETA.CLASS_COLUMNS	Info
32	CST0100	CSTV260	2	2	REFMETA.REFERENCE_COLUMNS	No errors detected in REFMETA.REFERENCE_COLUMNS	Info
33	CSTV260	CSTV260	3	1	STDMETA.STANDARDS	Required column cannot be null	Error
34	CSTV260	CSTV260	3	2	STDMETA.STANDARDS	Required column cannot be null	Error
35	CST0100	CSTV260	4	1	STDMETA.STANDARDLOOKUP	No errors detected in STDMETA.STANDARDLOOKUP	Info
36	CSTV260	CSTV260	5	1	STDMETA.STANDARDSASREFERENC	Required column cannot be null	Error
37	CSTV260	CSTV260	5	2	STDMETA.STANDARDSASREFERENC	Required column cannot be null	Error
38	CSTV260	CSTV260	5	3	STDMETA.STANDARDSASREFERENC	Required column cannot be null	Error
39	CSTV260	CSTV260	5	4	STDMETA.STANDARDSASREFERENC	Required column cannot be null	Error
40	CST0100	CSTV260	6	1	REFCNTL.VALIDATION_MASTER	No errors detected in REFCNTL.VALIDATION_MASTER	Info
41	CST0100	CSTV260	7	1	MESSAGES.MESSAGES	No errors detected in MESSAGES.MESSAGES	Info
42	CST0200		1	0	CDISC-CRTDDS 1.0	PROCESS WORKFLOW: Validating CDISC-CRTDDS 1.0	Info

VALIDATE_IQOQ.SAS

This driver is designed to provide a workflow that enables the user to run ***all*** internal validation checks identified as supporting Toolkit IQOQ.

The current driver workflow is described in the following steps:

Choose the standard(s) of interest and populate work._cstStandardsforIV with these standard(s). See a) in the validate_standard.sas example listed above for a code excerpt.

a) Modify the standard validation SASReferences to point to the validation_control *view* of interest.

```
*******************************************************************************;
* The validation_control_fw view of the validation_master data set includes just those *;
* core framework checks that look only within the <cstGlobalLibrary>/metadata folder.  *;
*******************************************************************************;
libname _cstTemp "&studyrootpath/control";

data work.stdvalidation_sasrefs;
  set _cstTemp.stdvalidation_sasrefs;
  if type='control' and subtype='validation' then
  do;
    filetype='view';
    memname='validation_control_fw.sas7bvew';
  end;
run;
```

The view validation_control_fw is one of several provided with Toolkit 1.5. The development code used to create this view is listed below.

```
proc sql;
  create view cstcntl.validation_control_fw
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype)="GLMETA";
quit;
```

b) Call the process setup macro to perform all file and library allocations. The returned &_cstSASRefs data set contains fully-resolved paths and memnames.

```
%cstutil_processsetup(_cstSASReferencesLocation=&workpath,
                      _cstSASReferencesName=stdvalidation_sasrefs);
```

c) Re-create work.stdvalidation_sasrefs, setting _srcfile='FWVAL':

```
data work.stdvalidation_sasrefs;
  set &_cstSASRefs;
  attrib _srcfile format=$8. label='File source for record';
  ***************************************************************;
  * Framework validation sasreferences: cstcntl.stdvalidation_sasrefs *;
  ***************************************************************;
  _srcfile='FWVAL';
run;
```

d) (New step) Dynamically build reference_tables and reference_columns data sets from a SASReferences data set and call cstvalidate(), which uses the validation_control_fw view.

```
%cstutilbuildmetadatafromsasrefs(
     cstSRefsDS=work.stdvalidation_sasrefs,
     cstSrcTabDS=work.source_tables,
     cstSrcColDS=work.source_columns
     );

%cstvalidate;
```

e) (New step) Modify the standard validation SASReferences to point to the validation_control_stdiqoq view for any standard specific processing requested in step a).

The view validation_control_stdiqoq is one of several provided with Toolkit 1.5. The code used to create this view is listed below:

```
proc sql;
  create view cstcntl.validation_control_stdiqoq
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype)=" STDIQOQ";
quit;
```

f) Call the code-generator macro to build the job stream for each standard:

```
filename incCode CATALOG "work._cstCode.stds.source" LRECL=255;

%cstutilbuildstdvalidationcode(
    _cstStdDS=work._cstStandardsforIV,
    _cstSampleRootPath=_DEFAULT_,
    _cstSampleSASRefDSPath=_DEFAULT_,
    _cstSampleSASRefDSName=_DEFAULT_
    );
```

SAS Code is generated by the macro above for each standard that is specified in the _cstStdDS parameter. In this example the 2 standards are ADaM and CRT-DDS. Once this code is created it is then %included and the following workflow is persisted.

1. Initialize work._cstTempSASRefDS to accumulate SASReferences records from all standards of interest for later use by cstvalidate().
2. Look for standard-specific standardSASReferences data set from the Global Library. If found, run cstutil_processsetup() using this file.
3. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STD'.
4. Look for standard-specific sdtvalidation_sasrefs data set from the Sample Library. If found, run cstutil_processsetup() using this file.
5. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STUDY'.
6. Remove any duplicate records from work._cstTempSASRefDS (keys=standard standardversion type subtype).
7. Run %cstutilbuildmetadatafromsasrefs(cstSRefsDS=work._cstTempSASRefDS, cstSrcTabDS=work.source_tables, cstSrcColDS=work.source_columns). This macro dynamically builds reference_tables and reference_columns data sets from a SASReferences data set.
8. Set _cstSASRefs=work._cstTempSASRefDS – the cumulative ready-to-go SASReferences data set.
9. Call cstvalidate(), which uses the validation_control_stdiqoq view from e) above.
10. Remove standard-specific records from work._cstTempSASRefDS, in anticipation of appending new records for the next standard to the framework records that remain.

g) Repeat steps 1-10 as described above for *each* standard selected in a) above.

h) Results are collated in cstrslt.validation_results.

VALIDATE_FRAMEWORK.SAS

This driver is designed to provide a workflow that enables the user to run ***all*** internal validation checks pertaining to the Toolkit Global Library metadata folder (<cstGlobalLibrary>/metadata).

The current driver workflow is described in the following steps:

a) Choose the standard(s) of interest and populate work._cstStandardsforIV with these standard(s). See a) in the validate_standard.sas example listed above for a code excerpt.

b) Modify the standard validation SASReferences to point to the validation_control *view* of interest.

```
*******************************************************************************;
* The validation_control_fw view of the validation_master data set includes just those *;
* core framework checks that look only within the <cstGlobalLibrary>/metadata folder.  *;
*******************************************************************************;
libname _cstTemp "&studyrootpath/control";

data work.stdvalidation_sasrefs;
```

```
  set _cstTemp.stdvalidation_sasrefs;
  if type='control' and subtype='validation' then
  do;
    filetype='view';
    memname='validation_control_fw.sas7bvew';
  end;
run;
```

The view validation_control_fw is one of several provided with Toolkit 1.5. The code used to create this view is listed below:

```
proc sql;
  create view cstcntl.validation_control_fw
    as select *
    from cstrcntl.validation_master as a
    where upcase(a.checktype)="GLMETA";
quit;
```

c) Call the process setup macro to perform all file and library allocations. The returned &_cstSASRefs data set contains fully resolved paths and memnames.

```
%cstutil_processsetup(
    _cstSASReferencesLocation=&workpath,
    _cstSASReferencesName=stdvalidation_sasrefs
   );
```

d) Re-create work.stdvalidation_sasrefs, setting _srcfile='FWVAL':

```
data work.stdvalidation_sasrefs;
  set &_cstSASRefs;
  attrib _srcfile format=$8. label='File source for record';
  ********************************************************************;
  * Framework validation sasreferences:  cstcntl.stdvalidation_sasrefs *;
  ********************************************************************;
  _srcfile='FWVAL';
run;
```

e) (New step) Dynamically build reference_tables and reference_columns data sets from a SASReferences data set and call cstvalidate(), which uses the validation_control_fw view.

```
%cstutilbuildmetadatafromsasrefs(
    cstSRefsDS=work.stdvalidation_sasrefs,
    cstSrcTabDS=work.source_tables,
    cstSrcColDS=work.source_columns
    );

%cstvalidate;
```

f) (New step) Modify the standard validation SASReferences to point to the validation_control_std for any standard specific processing requested in step a).

g) Call the code-generator macro to build the job stream for each standard:

```
filename incCode CATALOG "work._cstCode.stds.source" LRECL=255;

%cstutilbuildstdvalidationcode(
    _cstStdDS=work._cstStandardsforIV,
    _cstSampleRootPath=_DEFAULT_,
    _cstSampleSASRefDSPath=_DEFAULT_,
    _cstSampleSASRefDSName=_DEFAULT_
    );
```

SAS Code is generated by the macro above for each standard that is specified in the _cstStdDS parameter. In this example the 2 standards are ADaM and CRT-DDS. Once this code is created it is then %included and the following workflow is persisted.

1. Initialize work._cstTempSASRefDS to accumulate SASReferences records from all standards of interest for later use by cstvalidate().
2. Look for standard-specific standardSASReferences data set from the Global Library. If found, run cstutil_processsetup() using this file.
3. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STD'.
4. Look for standard-specific sdtvalidation_sasrefs data set from the Sample Library. If found, run cstutil_processsetup() using this file.
5. Append the fully-resolved work._cstSASRefs to the work._cstTempSASRefDS created in 1. above, setting _srcfile='STUDY'.
6. Remove any duplicate records from work._cstTempSASRefDS (keys=standard standardversion type subtype).
7. Run %cstutilbuildmetadatafromsasrefs(cstSRefsDS=work._cstTempSASRefDS, cstSrcTabDS=work.source_tables, cstSrcColDS=work.source_columns). This macro dynamically builds reference_tables and reference_columns data sets from a SASReferences data set.
8. Set _cstSASRefs=work._cstTempSASRefDS – the cumulative ready-to-go SASReferences data set.
9. Call cstvalidate(), which uses the validation_control_fw view from b) above
10. Remove standard-specific records from work._cstTempSASRefDS, in anticipation of appending new records for the next standard to the framework records that remain.

h) Repeat steps 1-10 as described above for *each* standard selected in a) above.

Results are collated in cstrslt.validation_results.

CONCLUSION

In the end, when a user is running any validation software, it is important that the user has a high level of confidence that any supplied standards metadata is not only installed correctly but contains accurate information. This paper described the process designed by SAS for its Clinical Standards Toolkit product to ensure the product is installed properly and all required metadata is where it should be and accurate. In addition to verifying standards shipped with the product, the user can also verify their customized standards.

MORE INFORMATION

SAS Clinical Standards Toolkit information can be found at: http://support.sas.com/rnd/base/cdisc/cst/index.html

CONTACT INFORMATION

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

Gene Lightfoot
SAS Institute Inc.
SAS Campus Drive R4318
Cary, North Carolina 27513 USA
+1 (919) 677-8000
+1 (919) 677-4444 (Fax)
gene.lightfoot@sas.com
www.sas.com