MEDICAL WRITING

CLINICAL PROGRAMMING

BIOSTATISTICS

DATA MANAGEMENT

REGULATORY AFFAIRS

MEDICAL INFORMATION

DRUG SAFETY

CLINICAL DEVELOPMENT SUPPORT

GMP AND GLP SERVICES

6880 Commerce Blvd.
Canton, MI. 48187
Phone: 734.245.0310
Fax: 734.245.0320
www.mmsholdings.com

Prove It!

Importance and Methodologies of Validation in Clinical Trials Reporting

Michigan SAS Users Group
18OCT2012

Chris Hurley, Senthil Kumar

ISO 9001:2008 Certified

Agenda

- What is validation
- Why is validation needed
- How do you approach validation
- General techniques facilitate validation
- Validation tool (OpenCDISC Validation)

What is Validation

- In 1987 the FDA published a document entitled: '*FDA Guidelines on General Principles of Process Validation'.*

- *Process validation is establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes.*

- **Note:** This definition indicates that validation applies to our computer programs and that if the validation is not documented then there is no proof that it occurred.

What is Validation

- The General Principles of Software Validation (FDA 2002)
- **Validation -** "*Confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled."*
- **Verification** - *"Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase."*
- Doing the right thing, doing it right, consistent repeatable results

Why is Validation Needed

Analogy from steel/airline industry

- Standard validation not exactly followed
- Deviations from consistent quality
  - Bolts incorrectly heat treated - 60% strength
- Deviations from consistent process
  - Partnair Flight 394 take-off checklist; power sources
- Structural failure in tail with tragic results

Why is Validation Needed

Pharmaceutical Example - Therac-25 incident

- Large radiotherapy device
- Consistent software quality?
  - Hi-power beam spreader for X-rays
    - Mechanical interlocks replaced with software
    - Safety algorithm error occurred when the operator changed the setup too quickly
- Deviations from consistent process
    - Used old software which relied on hardware interlocks
    - Error messages not understood
- Operator over-rides had tragic results

How do you approach validation

- Reporting accuracy in clinical trials is crucial because these data represent the people within the clinical trial
- Validation is a regulatory requirement by FDA
  - *Data and records being produced and maintained by computer systems must be complete and accurate, and the data integrity is the responsibility of the user or owner of the data*
- Delivering consistent quality develops a positive relationship with clients

How do you approach validation

- Start with all the information
- Have a validation plan
- Make the code do the work
- Be proactive
- Ask questions
- Validating early saves time
- It’s never too early for quality

Validation Methods

1. Basic review of SAS code
    - Second programmer goes through source programmer's SAS program and checks logic, looks for correct dataset names, verifies merging, etc.

2. Extensive review of SAS code
    - Combination of code review and programming
    - Second programmer checks source programmer's code for logic as done during basic review
    - Can also program certain pieces to recreate output and check that same output is obtained

3. Double programming
    - Second programmer only looks at specifications and attempts to create entire output
    - Should not look at source programmer's code during this process
    - Upon completion, the project lead will designate someone to test the results and inform programmers of discrepancies

Limitations of Each Method

- Basic review of SAS code
  - Reviewer must understand task and study thoroughly to perform a useful review. (True for all three validation types.)
  - Code functionality is not always understood easily
  - Easy to miss or overlook a mistake
- Extensive review of SAS code
  - Still possible to overlook errors since not everything is checked
  - Second programmer decides what to check
  - No clear “stopping point” when programmer can be certain validation is complete
  - Data can be refreshed prompting further review
- Double programming
  - This method occupies the largest amount of resources.
  - For this method to be successful, specifications must be very detailed/clear and there should be a designated resource for making updates
  - Must plan for the time needed to fully complete validation

Double Programming

- One of the standard validation methods in which two independent programmers create independent programs and then compare output

Principles:

- This process requires at least two programmers, a source programmer who will produce the final deliverable and a validation programmer who will produce a second output to be used solely for checking that source output is correct.
  - A third party (possibly a programmer or a lead) may be needed to compare results

- Any deliverable that is programmed is a candidate for double programming validation, in particular SDTM or ADaM datasets, tables, listings and figures.

- This is particularly valuable for items that serve as predecessors for later items, such as ADaM datasets that will be used to generate TLG's

Advantages of Double Programming

- Significant increase in quality in final deliverable as almost all mistakes will be addressed and worked out, whether large or small
- Compare output easy to reproduce and provides documentation that validation was done.
- Forces the specifications to be correct in such a way that two independent programmers can obtain the same results
- If done correctly, this validation strategy allows for a great amount of confidence in final deliverables
- With a minimal amount of effort, minor changes or updates, including refreshed data, can be verified quickly and confidently
- Comparisons can often be made through an automated technique, such as PROC COMPARE, which eliminates human error and fatigue
- When dealing with predecessors like SDTM and ADaM that are used in TLGs, this method limits mistakes early in the process so as to avoid updating or correcting something that was already “final”

Dataset Double Programming

- Two programmers will use the specifications to *independently* create a given dataset
  - Independence is backed up by math. The probability that two people make a given mistake is significantly lower than one person.
  - For example, under true independent circumstances, if the probability of making a given mistake is 10%, then there is a 10% chance Programmer A makes it, but there is only a 1% (10% x 10%) chance both Programmers A and B make it, which is a 10-fold reduction in the probability of that mistake

- Therefore, during development, the source and validation programmers should not communicate their logic and should ***never*** view each other's code
  - Additionally, the source or validation programmer should never rely on the other programmer to have it right – that is, they should strive to get it right on their own, even if this causes a delay. Relying on others can lead to inaccurate results

- When each programmer is confident that he or she is complete, the project lead will designate someone to perform an initial comparison

- The designee should run a PROC COMPARE on the source and validation datasets

PROC COMPARE output:

1. Number of records (N)
2. Sort order
3. Individual variables

- Validation logs and e-mail can be used to communicate issues to the programmers. There should be a record of the steps that were taken during validation and there *must* be a record that validation was done
- If the datasets match the first time, then something is probably wrong! This process can take many iterations, depending on complexity of the datasets and clarity of the specifications. So it will likely take a lot of communication back-and-forth between all parties before this validation step is complete

Dataset Compare: Step1

Step 1: Number of Records (N):

This should be the same in both datasets. If not, this discrepancy should be reported to the programmers immediately. There is no value in proceeding with the compare.

The COMPARE Procedure
Comparison of O.VS with P.VS
(Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs
O.VS	27JUL11:19:17:27	27JUL11:19:17:27	19	10691
P.VS	08MAR11:18:03:44	08MAR11:18:03:44	19	1868

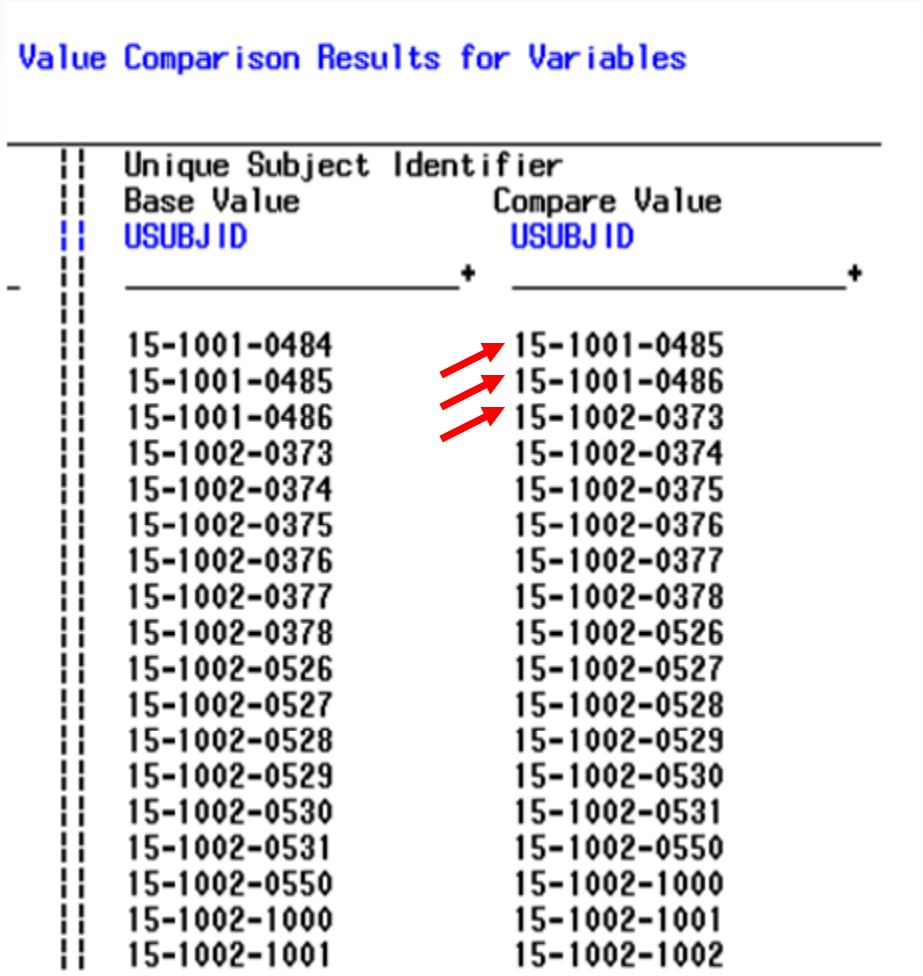
Variables Summary

Number of Variables in Common: 19.
Number of Variables with Differing Attributes: 6.
Number of ID Variables: 1.

This compare indicates a huge discrepancy in sample size (10,691 vs. 1,868) that should be looked into first.

Dataset Compare:Step2

- Step 2: Sorting order
- After the number of records match, it should be ensured that both datasets are sorted in the same order. If even one sort variable is different, the entire comparison could be different.
- As in the example on the left, if the values not lined up exactly right, the sort order may not be identical. Confirm with both programmers that the sorting order is the same and matches the sorting keys in the specifications.



Data step Compare: Step3

- Step 3: Check each Individual variable
- The last check should be to look at each variable individually. It may be useful to keep a unique identifier variable such as USUBJID in an ID statement within the PROC COMPARE. This statement matches certain observations by subject and can be helpful when investigating discrepancies
- A macro should be used to run through all variables. This keeps a record of each variable checked.

Value Comparison Results for Variables

USUBJID	Planned Arm Code Base Value ARMCD	Compare Value ARMCD
15-1002-1188	PLB	OSP30MG
15-1005-0428	OSP60MG	OSP30MG
15-1005-0460	PLB	OSP30MG
15-1016-3016	PLB	OSP30MG
15-3144-0270	PLB	OSP30MG
15-3144-0937	OSP60MG	OSP30MG
15-3147-0097	OSP60MG	OSP30MG
15-3147-0694	OSP60MG	OSP30MG
15-3244-0023	OSP60MG	OSP30MG
15-3292-0371	PLB	OSP30MG
15-3292-0913	OSP60MG	OSP30MG

ID variable helps identify which subjects have the problem

Overview of Dataset Comparisons

- In an ideal situation, a dataset would not be considered validated until ALL variables are a 100% match

- In practice, if a discrepancy exists, and it is clear that the issue is on the validation side, it may be possible to ignore it, as long as the discrepancy is documented. For example:
  - Sometimes the lengths of the variables between source and validation will not be the same, so there may be extra space to the right side of a value.
  - Source is upcase but validation is propcase

- However, if a discrepancy is left as is and documented, please be sure that ALL values are checked. By default, PROC COMPARE only displays the first 50 records. The MAXPRINT option can be used to display more records, which will be necessary if the dataset has more than 50 records

- Finally look for this message at the bottom of the output "*Note: No unequal values were found.All values compared are exactly equal".* When you see this message in addition to matching step1, 2 & 3 values then your job of validation is done!!

Table Double Programming

- As with dataset programming, two programmers should look at the specifications and/or SAP and independently attempt to create the same table
- The validation programmer should not be as concerned with matching the exact format here, as this adds no value
  - However, the closer in overall format and structure the validation output is to the table, the easier it is for a third party to make comparisons, which saves time
- The validation programmer should generate output using ODS, such as a RTF
- Unless specified, validation programmer does not have to create percentages
  - However, tables in the form of xx/xx (xx.x%), the numerator AND either the denominator OR percentage should be shown
- Same “independence” rules apply as with dataset programming
- When both programmers are complete, a third party may be notified to visually compare the two outputs

Comparing Table Output

- A visual comparison should be made between the source and validation tables
  - Typically, just one output needs to be printed, while the other can remain on the computer screen
  - Printing double-sided and saving as much paper as possible should be a priority
- Depending on the size and complexity of the table, either a full check (all values compared) or a spot check (approximate percentage of values compared) should be completed, which should be determined by project lead
- The person doing the comparison notifies source and validation of discrepancies, via validation logs and/or e-mails, and the two programmers should discuss their logic and rationale until issues resolved
- Once initial validation is complete, it may be useful to save a copy of the source and validation output, as there may be updates to datasets, SAP, etc
  - This allows for electronic comparisons (e.g. Word's Merge and Compare, PDF compare) between updated and previously validated versions and spares a visual comparison
  - Only the "changed" values need to be compared and any changes seen should be found in both source and validation output

Listing and Figure Double Programming

- It will not be necessary to recreate *output* for listings and figures during validation
- For listings and figures, source programmer should output the “final” dataset that goes into the final procedure, such as PROC REPORT, PROC GPLOT, etc
- The validation programmer will then create a program to produce the same dataset, but does not have to run the final procedure
- Then, a third programmer can use a PROC COMPARE and follow steps as in dataset validation, until datasets match
- Following these steps, it will still be required that the *output* of the listing or figure be checked, to ensure that all datasets and options in the final procedure were implemented correctly

General Techniques to Facilitate

- Using PROC FREQ for validation
- MSGLEVEL=I in merge statement
- Merge statement with IN=option
- Using Macros effectively and judiciously
- Maintain a clean log
- Flagging problem data
- Don’t Delete
- Drop the duplicates

Using PROC FREQ for Validation

- Most commonly used in Clinical Data Validation
- Helpful when performing cross variable checks

An example:

```
data demo;
    set orglib.demo;
        if sexcd eq 1 then sex = "Male";
        else if sexcd eq 2 then sex = "Female";
run;

proc freq data=demo;
    tables sexcd*sex / list missing;
    title 'CHECK RECODES';
run;
```

Cont…

- In this case it is creating a character version of a variable that was originally collected as a numeric variable
- The code needs to prove that the meaning of the variable being transformed has not changed
- In this output, it is easy to spot the error in reformatting the SEXCD variable

SEXCD	sex	Frequency
1	Male	78
2	Fema	166

MSGLEVEL=I in Merge statement

- MERGE is a very effective and powerful tool
- Can give surprising and undesirable results

out1

v2	constant	v1
1	1	1
2	1	1
3	1	1

main

v2	constant	v1
1	1	2
2	1	2
3	1	2

```
data out3;
        merge out1 main;
        by constant v2;
run;
```

v2	constant	v1
1	1	2
2	1	2
3	1	2

Cont...

- The MSGLEVEL system option gives additional information in the log when merging the datasets

```
options msglevel=I;
data out3;
        merge out1 main;
        by constant v2;
run;
```

```
105  options msglevel=I;
106  data out3;
107      merge out1 main;
108      by constant v2;
109  run;

INFO: The variable v1 on data set WORK.OUT1 will be overwritten by data set WORK.MAIN
NOTE: There were 3 observations read from the data set WORK.OUT1.
NOTE: There were 3 observations read from the data set WORK.MAIN.
NOTE: The data set WORK.OUT3 has 3 observations and 3 variables.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

MERGE statement with IN=option

- The IN=dataset option is especially useful when merging and concatenating datasets.
- The IN=dataset option tells SAS to create a temporary variable.

```
data vitals checkme;
            merge vitals(in=invl) visit (in=invt);
            by inv_no patid visit ;
            if invl and invt then output vitals ;
            else output checkme;
run ;
```

```
NOTE: There were 1723 observations read from the data set WORK.VITALS.
NOTE: There were 1726 observations read from the data set WORK.VISIT.
NOTE: The data set WORK.VITALS has 1723 observations and 19 variables.
NOTE: The data set WORK.CHECKME has 3 observations and 19 variables.
NOTE: DATA statement used:
      real time               0.02 seconds
      cpu time                0.02 seconds
```

Using MACROS effectively

- General rule for truly efficient programming is to add macros only when they add significantly to the process
- Macros can also create validation nightmares if used in excess
- Important to consider the cost benefit ratio
- Use **mprint**, **mlogic** and **symbolgen** for macros validation

Maintain a clean log

- Log not only be free of error but also free of warnings and some of the notes
  - NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column)
  - NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column)
- It is much easier to notice real issue if they arise
- Any issues caused by new data are easy to see as you skim through the file

Flagging problem data

- Useful when tracking how data is moving through complicated logic statements

```
data flags ;
    set orglib.vitals ;
    if pr gt 95 then do ;
     gothere = 1 ;
     if resp le 16 then do ;
                gothere = 2 ;
                if temp ge 99 then newvar = 1 ;
     end ;
    end ;
run ;
```

Flagging problem data

```
proc print data=flags (where=(gothere ne .)) ;
     var inv_no patid visit pr resp temp gothere newvar ;
     title "CHECK LOGIC FOR NEWVAR" ;
run ;
```

INV_NO	PATID	VISIT	PR	RESP	TEMP	gothere	newvar
1	9	Week 5	104	14	99.4	2	1
1	10	Day -1	98	12	98.4	2	.
1	17	Week 6	97	14	98.9	2	1
1	40	Week 4	96	14	98.4	2	.
1	41	Week 6	100	18	104.8	1	.
2	62	Week 3	100	16	98.6	2	1
2	62	Week 6	96	16	99.3	2	1
2	65	Week 6	96	14	97.9	2	.
2	73	Week 5	98	14	97.3	2	.
2	95	Week 2	96	16	97.0	2	.
2	111	Day -1	96	16	98.1	2	.
2	112	Day -8	96	12	98.6	2	1
2	207	Week 1	100	14	97.6	2	.
2	207	Week 2	100	14	98.4	2	.
2	210	Day -8	100	16	97.1	2	.
2	210	Day -1	96	16	97.2	2	.
2	210	Week 2	98	16	98.0	2	.
3	125	Week 4	96	18	97.9	1	.
3	131	Week 1	100	16	97.0	2	.
3	216	Day -8	96	20	97.9	1	.
4	139	Day -1	96	19	98.6	1	.
4	140	Day -8	96	20	97.1	1	.
4	142	Day -8	96	20	98.1	1	.
4	144	Day -8	96	21	98.6	1	.
4	146	Day -8	96	18	98.9	1	.
4	149	Day -8	96	16	97.8	2	.
4	149	Day -1	96	14	97.5	2	.
4	160	Week 5	96	18	98.3	1	.

Don’t Delete

- Often you might want to remove unnecessary records from a dataset
- Generally tempted to code a simple statement like:

if temp it 0 then delete;

- This does not allow to check the deleted records

Cont…

```
data temp dropped ;
  set vitals (keep=inv_no patid visit temp) ;
  if temp lt 0 then output dropped ;
          else output temp ;
run ;

proc print data=dropped ;
  title 'TEMP LESS THAN 0 SO DROPPED FROM DATA SET' ;
run ;
```

```
43    data temp dropped ;
44       set vitals (keep=inv_no patid visit temp) ;
45       if temp lt 0 then output dropped ;
46                    else output temp ;
47    run ;

NOTE: There were 1727 observations read from the data set WORK.VITALS.
NOTE: The data set WORK.TEMP has 1720 observations and 4 variables.
NOTE: The data set WORK.DROPPED has 7 observations and 4 variables.
NOTE: DATA statement used:
      real time              0.09 seconds
      cpu time               0.03 seconds
```

Drop the duplicates

- Often datasets contain duplicate records that needs to be removed
- This can be done using Proc Sort and options NODUPKEY or NODUPREC
- To check the dropped duplicated records use the DUPOUT option in SAS 9 PROC SORT

Cont…

```
proc sort data=vitals (where=(pr gt 90))
        out=htemp
        nodupkey
        dupout=dropped ;
  by inv_no patid ;
run ;

proc print data=htemp;
    title 'PATIENTS WITH PULSE RATE OVER 90' ;
run ;

proc print data=dropped ;
    title 'DUPLICATES DROPPED FROM PROC SORT' ;
run ;
```

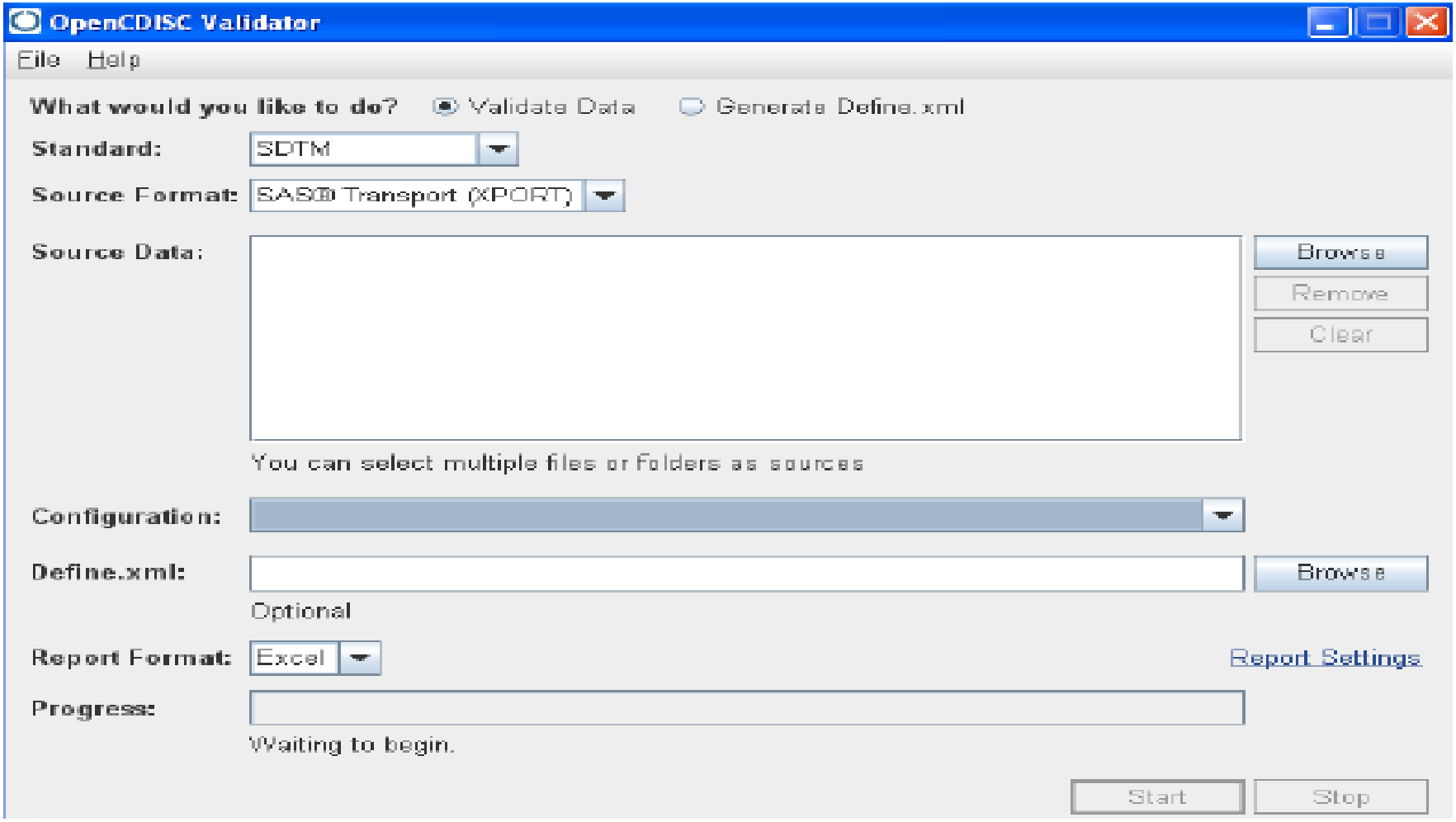
OpenCDISC validator tool:

OpenCDISC validator:

OpenCDISC validator is an open source java based project that provides validation of datasets against CDISC models (e.g. SDTM).

OpenCDISC validator is a collaborative project to develop an open source, freely available, and commercial-quality tool for ensuring clinical data compliance with CDISC standards, including SDTM, ADaM, SEND, Define.xml and others.

OpenCDISC validator tool:



MMS Holdings Inc.

ISO 9001:2008 Certified

Pros:

- Does not require any additional tools and takes up <10MB space on the drive
- Future expansion to CDISC, ADAM and SEND
- Generates validation reports in Excel, CSV and HTML
- The tool separates validation rules from application logic, so no need to update and revalidate
- Very intuitive – use through GUI

Cons:

- No user's guide or validation documentation provided with the installation
- Requires users to create their own configuration files if additional sponsor defined checks are to be included

OCV Key Features

- Metadata driven and configurable
- Validation rules and messages stored in xml
- Facilitates compliance with CDISC models
- Supports SDTM V3.1.2 and V3.1.1
- Checks against SDTM specification, controlled terminology, and Define.xml

Conclusion

- Validate and document - it's required
- Follow your organization's validation plans
- Create your own personal validation process
- Remember even the most simple algorithms must perform consistently and to specifications
  - Remember the bolts and bytes
  - Do the right thing!