PharmaSUG 2013 - Paper AD06

DataSetBuilder - An Application for Creating Edit Check Test Cases within SAS Data Sets by Non-Programmers

Richard Addy, Rho, Chapel Hill, NC

ABSTRACT

Validating edit checks is a necessary but cumbersome process – individual test cases must be created, and complex checks require many records across multiple data sets. The person validating the checks might not be a SAS programmer, and creating these records is a non-trivial task. Documentation of the test cases can be sparse (or non-existent), and it may be difficult for a second party to confirm the successfulness of the validation.

We approached these problems by combining several tools to better handle sections of the validation process. The primary engine of the application is a SAS program (DataSetBuilder or DSB) which modifies the contents of SAS data sets based on the specifications of a validator. A macro-enhanced Excel spreadsheet is the intermediary between the validator and DSB; this spreadsheet has numerous tools to quickly and efficiently specify test cases. Using Dynamic Data Exchange (DDE), DSB provides feedback to the validator on the test cases it creates, including any problems it encounters.

This approach allows the validator to focus on creating test cases, using a familiar tool that does not require any SAS programming proficiency. The SAS program handles the heavy lifting by creating records within the desired data sets and alerting the validator to any potential problems. A second party can quickly review a validator's test cases, or recreate cases on an as-needed basis.

INTRODUCTION

Clinical data is, sadly, not always as clean as it could be. Any number of problems can arise : were all the required fields completed? Did a patient provide inconsistent responses? Were visits conducted according to schedule? Clinical data needs to be checked, and it is efficient to do so programmatically, through edit checks.

But code also needs to be checked prior to its release into production. An edit check that fails to identify a problem is not very useful (at best), and one that incorrectly flags observations wastes time and money. One method to confirm the programming is performing as expected is to create cases in a test environment designed to specifically fail or pass a specific edit check.

A simple approach would be to manually create test cases in a testing environment – most edit checks only involve a few variables, and a data set can be opened and modified quickly. But that requires a certain level of familiarity and comfort with SAS data sets, and it can be haphazard. It would be easy to use the same row to validate multiple checks, and rows can be re-used, which leads to muddy and transient test cases.

This paper describes an application, DataSetBuilder, which creates edit check cases within a structured environment, requiring a minimal amount of SAS knowledge on the part of the validator. The validator uses an Excel spreadsheet to specify their desired test cases; a SAS program reads this spreadsheet, attempts to build the test cases, and supplies feedback by using Dynamic Data Exchange (DDE) to update the spreadsheet. The intent of this application is to allow the validator to focus as much as possible on specifying test cases, while the SAS program automates record creation.

TEST CREATION

PROCESS OVERVIEW

Programming edit checks begins with the study's validation plan. From the plan, edit checks are programmed and validated within a testing environment before being moved into production. In our system, edit checks are not validated through double programming; instead a validator creates test cases to see how well (or not) the edit checks perform.

This can be done manually – the validator can write a SAS program to generate their test cases, or they could open an existing data set and modify values as necessary. We found this approach had several draw backs – it required a level of comfort with SAS that not all the validators possessed, rows tend to get re-used for multiple checks, so often test cases were overwritten, and complex checks (especially those involving records across multiple data sets) were

difficult to keep aligned. At the end of the process, the validator was often left with little more than notes on scrap paper detailing how they created their test cases.

We decided it would be a better use of the validators' time if they could, as much as possible, focus on specifying the test cases they needed to create. Other activities, such as creating and documenting the test cases, should be automated. The work of the validator should be reviewable and reproducible. Ideally, the validator would not be required to learn a new programming language; the system should take advantage of the skills the validator already posses.

We chose SAS® as the primary tool for automating test case creation – our data management systems is SAS-based, SAS interacts well with a number of other applications, and since we need to build SAS datasets, there's no better platform than SAS.

However, we needed a way for a validator with little or no SAS programming experience to describe their test cases and to receive feedback on the creation of those test cases. It needed to be a structured environment – both to allow the validator to focus on one edit check at a time, and to maximize re-use of the application across studies. As much as possible of the validator's work needed to be within this single system – having to deal with multiple systems and files would slow things down. With these factors in mind, we chose to use an Excel spreadsheet as the interface with the validator. Excel is a common application and familiar to our validators, and SAS can interact with it well.

The validation process begins with an Excel sheet (DSB.xls), pre-populated with information from the validation plan. The validator specifies the test cases they need built, and then runs a SAS program (DSB.sas). DSB.sas reads the spreadsheet, performs a number of checks, and creates the specified test cases. The validator runs the edit checks and notes which checks passed validation and which did not. Edit checks that did not pass are updated, if necessary, or the validator can update their specifications, as appropriate. DSB.sas can be run as often as needed, until all edit checks are validated.

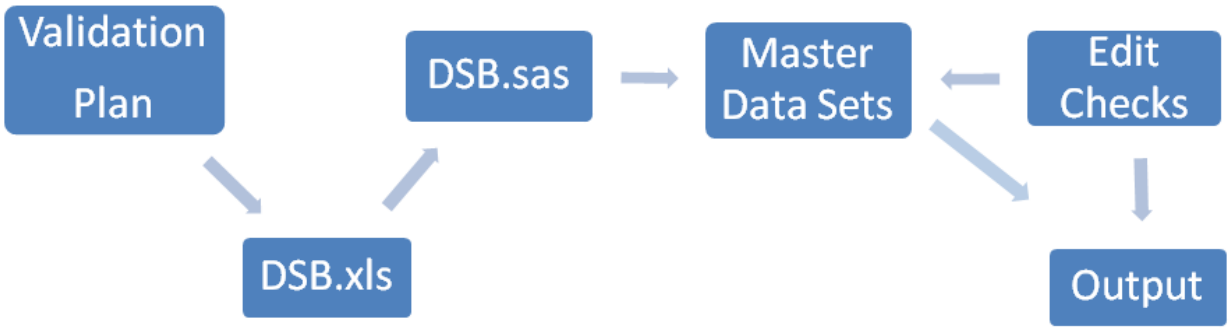


Figure 1. Validation Process using DSB

NAMING TEST CASES

We increased the speed of review by embedding within the test case subject id information about the tab the record is built from, the query code it is intended to validate, and whether the observation should cause an edit check to trip (a fail case) or not (a pass case). In our system, the subject id is split into three parts, as shown in the following figure:

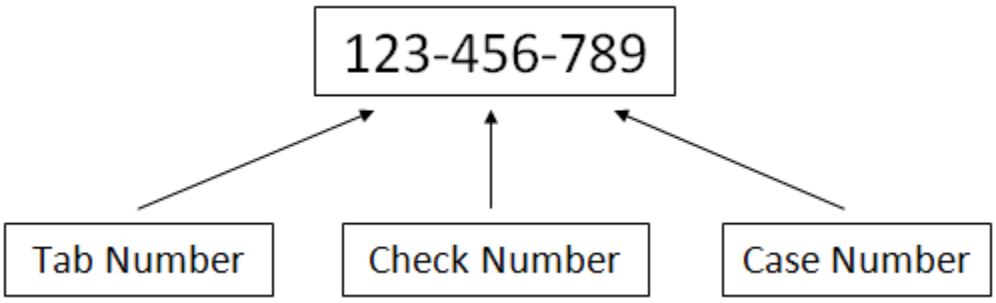


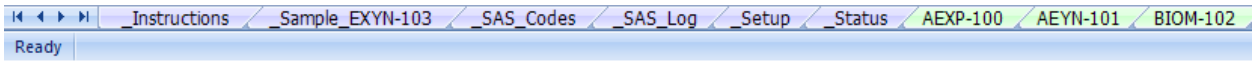
Figure 2. Subject ID Format

By convention, subject ids ending with odd numbers are intended to be fail cases, and subject ids that end with even

numbers are pass cases. Optionally, the validator may indicate that the row should not be built by preceding the subject id with an 'S' (for Skip Subject). This allows the validator to focus on a few test cases at a time, if they prefer.

THE EXCEL SHEET (DSB.XLS)

We created a template DSB.xls file for the validators to use as a starting point. DSB contains a setup tab that is used to describe the study structure, a few tabs containing information in DataSetBuilder as a whole and some pointers, and a series of build tabs where the validator specifies the test cases they need created.



Display 1. , Setup, Informational, and Build Tabs in DSB.xls

Do not edit this column	*Edit purple and orange cells only*		The Flag_Cell macro (SHIFT-CTRL-F) will highlight the active cell with the color of *this* cell.		
Column Name in Validation Plan	**Column Letter in Validation Plan**				
Test Name/Query Code	L		Allow multiple users?	No	
Error Message	D		Date Suffix	DT	Time Suffix TM
Datastream 1	E		Day Suffix	DA	Hour Suffix HR
Variables in DS 1	F		Month Suffix	MO	Minute Suffix MI
Datastream 2	G		Year Suffix	YR	
Variables in DS 2	H				
(optional) Datastream 3		N	← (Optional) Comments 1 (e.g., Reverse message, or similar) column to import		
(optional) Variables in DS 3		O	← (Optional) Comments 2 (or similar) column to import		
Type of Check/Test Type/Expression	K	P	← (Optional) Comments 3 (or similar) column to import		
Import test if Type of Check/Test Type column contains *any* of these words or phrases (case is not important). If cell B14 contains **ALL** then all tests that have a non-empty "Type of Check" column will be imported.	multivariate cross check				
Tell DataSetBuilder.sas to create datasets on ALL tabs or just the tab named here. Use the SHIFT-CTRL-T macro to put the *active tab* name in cell B24. Use the SHIFT-CTRL-A macro to put **ALL** in B24	ALL		**These three variables are imported into all tests:**		
List additional variables (all caps) to add to ALL tests when running the Import macro		ID	← Subject ID variable, e.g. ID, SUBJECTID		
		PHASE	← Phase variable (usually PHASE)		
		SEQNO	← Sequencing variable, e.g. SEQNO, PARSEQ		

Display 2. Setup Tab in DSB.xls

The Setup tab contains several items that could have been handled as parameters passed to a SAS macro – items like the name of the variables that identify the subject, form, and instance, how date values are stored, etc. Because we did not want to presume the validator was a SAS programmer, we elected to keep these items in DSB.xls – we wanted the SAS portion of the process to be a push button process, and to minimize the number of modifications the validator would need to make there.

BUILD TABS

In DSB.xls, test case specifications are spread across a series of tabs. In our data management system, edit check code tends to be organized around the primary data set for the check, so we used that as the basis for how we organized the build tabs. This is reflected in the query code for the check, so, for example, all the test cases for edit checks whose query codes began with 'VIST-' would be on a single tab. Test cases for checks whose query codes began with 'AEXP-' would be on a different tab.

This serves to break the edit checks into manageable chunks, which was necessary as we needed to prevent the use of filters in DSB.xls (it would interfere with DSB.sas' methods for providing feedback). It is common for checks to involve more than one data set, and test cases for any data set can be specified on any build tab. However, an edit check will only appear on a single build tab, no matter how many data sets are involved with the check.

In order for the SAS portion of the application to read DSB.xls, build tabs have a fixed structure. Each build tab has a common header – specifications for query codes, data sets, subject ids, variables, and variable values must occur in specific locations. The name of the build tab includes a numeric portion (to be used in the construction of subject id values) and a character portion that reflects the primary data set for the tab.

The first column is reserved for feedback from DSB.sas (which will, when run, overwrite any values present). The second column contains the query code for the check, and may contain notes from the data manager or programmer.

The third column holds the edit check message (and possibly, additional notes) – this column is for use by the validator and won't be used by DSB.sas. Columns 4 through 7 hold subject id values, form identifiers, sequence identifiers, and data set names. Beginning in column 8, the validator can specify what variables need to be included, and the values for these variables.

Each row with a subject id describes a single record in a single data set, and all variables of interest are specified on the row. With the exception of some internal variables, if a variable is not specified, it is assumed that the default value from the template row is sufficient.

To allow the validator the choice to focus on a small number of checks at a time, the validator has the option of specifying one, a few, or all of the build tabs for DSB.sas to use to create test cases. The validator can also set specific rows to be skipped, if they desire.

MACROS

We wanted the validator to spend the majority of their time describing their test cases, so several visual basic macros were programmed into DSB.xls to facilitate their efforts. A macro to import the edit check messages from the validation plan and create skeletal test cases got the validator off to a good start. Macros to set the current tab as the active build tab (or add it to the list of tabs to build) and to move quickly between tabs sped up navigation. Macros to add and delete subjects, toggle subjects between skip and build or pass and fail let the validator fine-tune their test cases more quickly.

None of the above macros were required – a validator could start with a blank version of DSB.xls and manually specify all the test cases they needed. But it would take them much, much longer to do so.

There is one internal macro that was required, however – the tab names in DSB.xls are study specific, and may vary over time. DSB.xls includes a macro that generates a list of all possible build tabs whenever the file is saved. DSB.sas reads this list as in reads in other elements from the spreadsheet.

SPECIFYING TEST CASES

The import macro in DSB.xls pre-populates a few fields to get the validator started. For each check, the query code and edit check message are placed in the appropriate columns. Three subjects are generated for each data set involved in the check, with appropriate id values (the first part of the id matches the build tab number, the second part matches the query code number, and the final part is numbered sequentially). The import macro makes a guess about the data sets and variables involved in the check – the quality of this guess is strongly associated with how well the validation plan adheres to a standard format. The import macro does not make any guess about what form values or sequence values will be needed (PHASE and SEQNO in the displays below).

		Test	Message	ID	PHASE	SEQNO	Data Stream	Var 1	Var 2
12									
13		TERM-0005	On CRF page 22, Date of Last Visit (!TERMdTERMDT) at Study Termination should be the same as or later than the last Visit Date record. Please review dates.				TERM	TERMDT	
14		Cross Check		117-005-001					
15				117-005-002					
16				117-005-003					
17									
18							VIST	VISITDT	
19				117-005-001					
20				117-005-002					
21				117-005-003					

Display 3. Starting point for test case specification

The validator, using the edit check messages (and possibly notes from the data manager and/or programmer) can then create test cases – specific combinations of values intended to pass or fail the check. The validator can add or delete rows on the build tab, as needed. Variables can be added, if necessary, or removed.

A few items carry forward until they are changed – once set, the query number, form value, sequence value, and data set are assumed to be constant until a new value is found. These values must be assigned, whether from the import macro or by the validator, but they need not be present on every row.

However, when specifying values for variables within the data set, every cell must be completed. An empty cell is used to indicate that the variable should be set to missing – not the value of the cell above.

12		Test	Message	ID	PHASE	SEQNO	Data Stream	Var 1	Var 2
13		TERM-0005	On CRF page 22, Date of Last Visit (!TERMdTERMDT) at Study Termination should be the same as or later than the last Visit Date record. Please review dates.				TERM	TERMDT	
14		Cross Check		117-005-001	5000	000		14JUN2012	
15				117-005-002				15JUN2012	
16				117-005-004				16JUN2012	
17									
18							VIST	VISITDT	
19				117-005-001	0000	000		15JUN2012	
20				117-005-002				15JUN2012	
21				117-005-004				15JUN2012	

Display 4. Test cases ready to be built

THE SAS PROGRAM (DSB.SAS)

The actual creation of test cases is handled by a SAS program, DSB.sas. This program is intended to stand alone – the only adjustment a validator needs to make to the program is to point it to their test environment, and that adjustment only needs to be made once, at the beginning of the validation process. Other than that one value, DSB.sas gets all the information it needs from the validator's spreadsheet.

READING DSB.XLS

When DSB.sas is run, it brings in several items from DSB.xls – it reads the Setup tab to determine the current build tab(s) and study structure, as well as a list of all build tabs contained in the spreadsheet. Each current build tab is examined individually; to preserve as much flexibility as possible, all values on the current build tab(s) are brought in as text strings.

	Test	Message	ID	Phase	seqno	DStream	Var1
11	TERM-0005	On CRF page 22, Date of Last Visit (!TERMdTERMDT) at Study Termination should be the same as or later than the last Visit Date record. Please review dates.				TERM	TERMDT
12	Cross Check		117-005-001	5000	000		14JUN2012
13			117-005-002				15JUN2012
14			117-005-004				16JUN2012
15							
16						VIST	VISITDT
17			117-005-001	0000	000		15JUN2012
18			117-005-002				15JUN2012
19			117-005-004				15JUN2012

Display 5. Initial Read of the Build Tab

Each row is examined to see if it is a test case specification – rows that do not include a specification are discarded. Within a row, the columns are parsed to determine the query code, the data set of interest, and the variables (and their values) that will need to be set. Values for the data set, form, and instance are carried forward until a new value is encountered. The input data set is transposed from one record per observation to one record per observation per variable. The location of each item on the spreadsheet is recorded, in order to provide feedback.

	ID	Phase	seqno	DStream	rownum	varnam	varval	colnum
1	117-005-001	5000	000	TERM	12	TERMDT	14JUN2012	8
2	117-005-002	5000	000	TERM	13	TERMDT	15JUN2012	8
3	117-005-004	5000	000	TERM	14	TERMDT	16JUN2012	8
4	117-005-001	0000	000	VIST	17	VISITDT	15JUN2012	8
5	117-005-002	0000	000	VIST	18	VISITDT	15JUN2012	8
6	117-005-004	0000	000	VIST	19	VISITDT	15JUN2012	8

Display 6. Data Set Values to Build

ERROR CHECKS

Once the rows that contain test case specifications are identified, some error checks are performed. The first few are structural checks: the study to be modified must be in a development area, the data set specified must exist within the study, and the variables specified must be present in the data set. There must be at least one row in the template data set that will be used to be the test cases. Character values are not allowed for numeric variables, with the exception of single character to indicate a special missing value. Date and time variables must have valid values.

Following that, some checks to confirm the test specification matches the constraints of how test cases should be constructed. Only a single record per subject, per form, per instance, per data set can be specified – duplicates are not allowed. The first section of the subject id should match the build tab number, and the second section of the subject id should match the numeric portion of the query code. A variable should only be specified once per data set.

Test cases are not created for rows that fail these checks – feedback is provided to the validator as to the reason(s) the row was failed.

CREATING TEST CASES

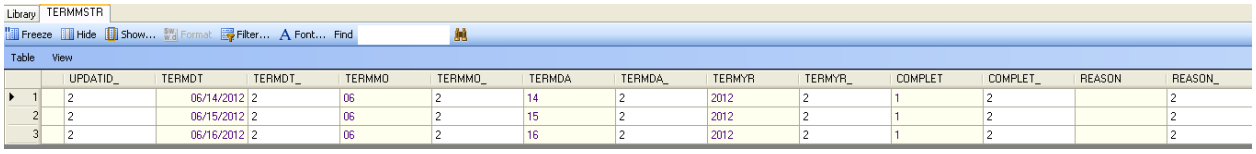
Once valid test case specifications are identified, the actual test cases are created. DSB uses a set of template data sets to serve as the basis for the new records – data sets generated during the development of the project's data entry system. We chose this option over building the rows based on project metadata, as the template data sets tended to be more complete.

DSB uses the first row of each template data set as the basis for all observation created within that data set. Unless otherwise specified in DSB.xls, the values in that template row will be used – for example, if the validator is building a record in an adverse events dataset, and needs to set the verbatim term of an event, the final record will have a verbatim term derived from DSB.xls, but other variables (onset date, outcome, etc.) will take their values from the template row.

Each time DSB.sas is run, it empties the master data sets of all observations that could have been built based on the current build tab(s). For example, if the build tab is TERM-117, DSB.sas deletes all records where the subject id begins with '117' from all master datasets. This prevents records generated from previous runs of DSB from interfering with the test cases the validator currently wants to generate. The validator also has the option of starting fresh with each run – in that case, DSB.sas deletes all records in the master data sets when it runs.

Each variable specified in DSB.xls is examined to determine if it is a number or a character, and if it is numeric, whether it is a date or a time. The format of the variable is used when setting its value – generally, this is only an issue with numeric variables, but it is possible that a validator has specified a character value that exceeds the length of the variable.

In addition to setting a variable to a specified value, DSB.sas also handles as many background tasks as possible. If the variable being set has flags associated with it, they are reset. If a date is stored as a day, a month, and a year value, in addition to the SAS date, DSB.sas also updates the records accordingly. If the data set has a key field that needs to be updated with the validator's subject ids, DSB.sas handles that as well. Variables that are not included in the test case specification are left unchanged from the template row.


Library TERMMSTR

Freeze Hide Show... Format Filter... Font... Find

Table View

	UPDATID_	TERMDT	TERMDT_	TERMMO	TERMMO_	TERMDA	TERMDA_	TERMYR	TERMYR_	COMPLET	COMPLET_	REASON	REASON_
1	2	06/14/2012	2	06	2	14	2	2012	2	1	2		2
2	2	06/15/2012	2	06	2	15	2	2012	2	1	2		2
3	2	06/16/2012	2	06	2	16	2	2012	2	1	2		2

Display 7. Example of a Portion of an Output data set

Finally, the value created in the data set is compared to the value specified in DSB.xls. If there is a difference between the two, this is reported back to the validator. The most common cause of a difference is truncation – the value created is shorter than the specified value.

PROVIDING FEEDBACK

Since DSB.sas interprets cells within the spreadsheet by context, it was important to provide feedback in cases where a problem developed and also, in cases where the test case record was built successfully. This allowed the validator to confirm that all rows they had intended to be built had, in fact, been built, or a reason was provided for lack of success.

When DSB.sas reads the build tab, it keeps track of the location of items it reads on the spreadsheet. As a result, DSB.sas can send formatting instructions to the spreadsheet using Dynamic Data Exchange (DDE). So, the primary method of feedback is color – variables are colored by type (green for text, light blue for numbers, purple for dates, darker blue for times). Problems are flagged in red. Additionally, the first column of the build tab contains a single period for every test case that was successfully built or an error code to describe what went wrong.

12		Test	Message	ID	PHASE	SEQNO	Data Stream	Var 1	Var 2
13		TERM-0005	On CRF page 22, Date of Last Visit (!TERMdTERMDT) at Study Termination should be the same as or later than the last Visit Date record. Please review dates.				TERM	TERMDT	
14	.	Cross Check		117-005-001	5000	000		14JUN2012	
15	.			117-005-002				15JUN2012	
16	.			117-005-004				16JUN2012	
17									
18							VIST	VISITDT	
19	.			117-005-001	0000	000		15JUN2012	
20	.			117-005-002				15JUN2012	
21	.			117-005-004				15JUN2012	

Display 8. Feedback on a successful build

12		Test	Message	ID	PHASE	SEQNO	Data Stream	Var 1	Var 2
13		TERM-0005	On CRF page 22, Date of Last Visit (!TERMdTERMDT) at Study Termination should be the same as or later than the last Visit Date record. Please review dates.				TERM	TERMDT	
14	.	Cross Check		117-005-001	5000	000		14JUN2012	
15	.			117-005-002				29FEB2012	
16	X-7			117-005-004				29FEB2013	
17									
18							VIST	VISITDT	VISYN
19	X-11			117-005-001				15JUN2012	1
20	X-6			117-005-002	0000	000		15JUN2012	1
21	X-6			117-005-004				15JUN2012	1

Display 9. Feedback on an unsuccessful build

For the sake of brevity, only a small error code is provided on the build tab; a more complete description of the error is presented on a separate informational tab within DSB.xls. In addition to highlighting the error code, the cell containing problematic value is also set to an error color.

Additionally, a brief summary of DSB.sas' activities (including number of rows built and number of rows failed) is written to a log tab in the DSB spreadsheet. This log tab and the first column in the build tab(s) are the only places where DSB.sas modifies the contents of cells within the spreadsheet. In all other instances, DSB.sas only modifies the formats of cells. This was important because the validator is responsible for the content of the cells, and we did not intend for DSB.sas to step on their toes.

S	Subject skipped (not built) by request. Subjects starting with an "S" are skipped; Use the SHIFT-CTRL-S macro to toggle Subject ID(s) between **nnn-nnn-nnn** and **S nnn-nnn-nnn**
D-1	The value built by DSB in SAS did not match the user's input.
X-1	Duplicate values for ID-var+Phase-var+Seq-variable within the same datastream
X-2	The name of the datastream is more than 4 characters
X-3	The first 3 characters of the subject ID do not match the tab number
X-4	The datastream specified does not exist in the Backup folder
X-5	There are no observations in the Backup dataset for this datastream
X-6	The variable indicated does not exist in the specified datastream
X-7	Not a valid date value
X-8	Not a valid time value
X-9	Not a valid numeric value
X-10	No variable name was provided, but a value was given
X-11	No Phase and/or Sequence value was provided for first subject in datastream (or value provided is illegal).
X-12	Middle segment of the subject ID must match the 7th, 8th, and 9th characters of the query codes type (e.g.) **DEMO-0015 (subj. ID must be nnn-015-nnn)** Middle segment of the subject ID must match the 6th, 8th, and 9th characters of the query codes type (e.g.) **DEMO-M015 (subj. ID must be nnn-M15-nnn)**

Display 10. Error Codes

FINDINGS

DSB IN PRACTICE

In general, we found that validators tended to focus on a few edit checks at a time – while DSB.sas can handle multiple build tabs, it was uncommon than it needed to do so.

DataSetBuilder did not decrease the overall speed of the validation process – it took just as long (or longer) for the validator to use DSB to specify their test cases, create them, and run edit checks using them. However, we did see a decrease in the time it took to correct an edit check that failed validation – the validator was able to provide the programmer with specific examples of values that did not perform correctly (values that the programmer could use in their own development environment), and this lead to the validator and programmer to reach a consensus more quickly.

Also, being able to view all the test cases for a study was helpful in training new validators. Being able to look at another study got them up to speed quickly, and errors in specifying test cases could be corrected early.

As DSB.sas only modifies specific variables and brings forward the values from a template row for others, it is common for a test case to unintentionally trip other edit checks. For example, test cases intended to validate a check of Visit 1 and Visit 2 dates could trip an edit check between the Visit 2 and Visit 3 dates. This lead to noisy output – the validator often had to go through a large amount to review an edit check's performance.

CHALLENGES ENCOUNTERED

DataSetBuilder does not lend itself to multiple validators working within a single project. Even when setting DSB,xls to be a shared spreadsheet, validators tended to get in each other's way, and it was easy for one validator to unintentionally overwrite another's test cases. In general, when more than one validator was needed, each validator it was necessary to set up each of them in a separate test area.

Some values in DSB.xls (PHASE, SEQNO, DATASTREAM) were assumed to carry forward until changed, while values for the other variables were assumed not to carry forward. While this was intended as a convenience, it often confused the validators.

DSB is of limited use in Electronic Data Capture (EDC) systems that don't store their data in SAS data sets. In our systems, EDC study data was available as SAS exports, so DSB can be used on the back end checks (generally, the ones requiring manual review). Using DSB to create test cases to validate front end checks would require an additional import step.

We had initial hopes of being able to set up a test case library, which validators could use to populate their

spreadsheets. However, differences in project structures between studies made this impractical. However, we did find that having example test cases was helpful to the validators.

NEXT STEPS

There is an opportunity to further automate the validation system – currently, the validator must manually confirm that fail cases causes edit checks to trip and pass cases do not. Since the subject id contains information about the edit check code, and whether it should cause the check to trip or not, it is possible to programmatically examine the edit check output and determine if a check performed as expected.

CONCLUSION

DataSetBuilder is an application that combines a SAS program and an Excel file in way to maximize the automation of test cases for edit check validation. This allows a validator to spend more of their efforts in specifying test cases and validating edit checks and less on the mechanics of creating records in data sets. However, it is not an application that increases the speed of the overall validation effort. However, it did increase the quality of the validation. Better and more comprehensive test cases were specified, and communication between the validator and programmer was improved.

Currently, DataSetBuilder is designed to construct test cases for the purposes of validating edit checks, and several sections of it are predicated on that premise. However, at its core, DSB is intended to create specific records with specific values in specific data sets, and its use could be broadened to other instances where a non-SAS programmer needs to create records within SAS datasets.

ACKNOWLEDGMENTS

Special thanks to Jared Weinberger, for the initial concept of DataSetBuilder, his work on the DSB.xls macros and spreadsheet, and his dogged persistence in making DataSetBuilder as flexible a tool as possible.

RECOMMENDED READING

Vyverman, Koen, 2001, Using Dynamic Data Exchange to Export Your SAS® Data to MS Excel — Against All ODS, Part I —, Proceedings of the Twenty-Sixth Annual SAS® Users Group International Conference, Cary, NC, SAS Institute Inc., Available at http://www2.sas.com/proceedings/sugi26/p011-26.pdf

CONTACT INFORMATION

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

Name:	Richard Addy
Enterprise:	Rho
Address:	6330 Quadrangle Drive
City, State ZIP:	Chapel Hill, NC 27517
Work Phone:	(919) 408-8000
Fax:	(919) 408-0999
E-mail:	richard_addy@rhoworld.com
Web:	http://www.rhoworld.com/
Twitter:	https://twitter.com/rhoworld