

Data Quality Management, Data Cleansing, and Discrepancy Reporting

Jenine Milum, The Ginn Group (CDC), Atlanta, Georgia

ABSTRACT

Whether your data comes to you in the form of SAS datasets or in another form, Data Quality Management and Data Cleansing are a must. With Data Cleansing comes Discrepancy reporting. This session covers Data Quality Management, Data Cleansing, and Discrepancy reporting techniques applicable to most SAS shops. Data Cleansing is important if the sources are external and especially critical if the data source is from manual input. Whether you are just getting started on a new project or realize more Data Management is necessary for a current project, these areas require proper documentation and planning using best standards. This paper will offer practical approaches to these topics using Base SAS/Macros.

Introduction

Having worked in many industries, it has become glaringly apparent that efficient Data Quality Management and Cleansing are absolutely necessary for the success of any programming effort. And data documentation is a requirement in many organizations. One needs to understand the data and its nuances to provide accurate reporting, and that's a start, but for a SAS programmer it is necessary to get into the data to really understand it. It is also a best practice to check the data for discrepancies up front rather than work around awkward problems after report coding is started. The tool of choice for these efforts, of course, is SAS.

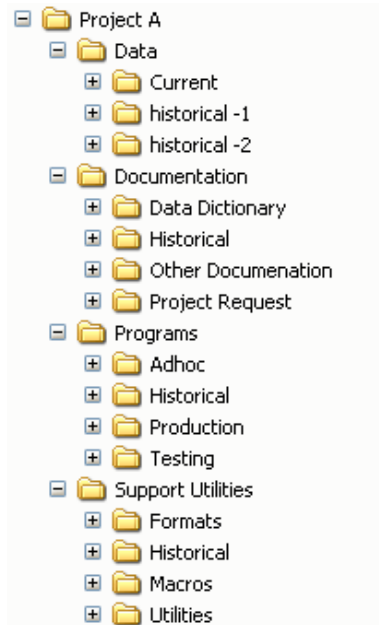
Getting Started

It may seem a bit obvious and basic, but knowing where your data is located and where current, updated documentation is available is a primary task. There is more than one way to accomplish this necessary step. Some companies have specific data storage and documentation standards that need to be followed. Other SAS shops provide a looser adherence to best standards and practices in this area. This paper provides some simple guidelines and suggestions that usually work in most situations.

STORAGE

Storage should be considered for four major groups of information: Data, Programs, Miscellaneous Supporting Utilities, and Project Documentation. The ability to locate this information without confusion is critical. Multiple copies, locations, and versions of this information are potentially confusing and how they're managed can make or break a project. Inaccurate reports are too frequently the result of not using the most current data or are based on documentation without the latest updates. All four storage groups don't necessarily need to be dealt with in the same manner. Depending on your business expectations, there are some basic guidelines that can be applied.

Here is a suggestion of a simplified project storage directory management structure:



Data storage should contain a folder/library where the most current updated datasets can be found. If this is not possible, then a very stable method for programs to identify the most current data needs to be in place, perhaps using a standardized file naming scheme. It can be incredibly helpful to include a text file document in each of data storage location that contains information about the datasets, such as where the data was captured, the frequency of the updates, where data dictionaries and other supporting documentation can be located, and even key personnel that are responsible for the data capture and its management. This text file doesn't need to contain detailed information but more is better and it will help those working with the data by providing a starting point if questions arise.

Documentation storage should contain any and all available information about datasets, programs, requests business rules, diagrams, and anything else that might prove useful. Again, more is better. One specific location to search for such materials can make life for anyone easier, and you yourself will greatly appreciate after, six months. Retaining older documentation is also quite helpful but be sure to clearly label superseded material to lessen the chance of referencing retired documents. Actually, one best practice is on the cover page of a superseded document added "Superseded by Version XXX dated MMDDYYYY, posted by Conscientious Employee". It takes a bit more work, but it's worth the effort.

Program storage is usually broken out into four major categories; Production programs, Testing programs, Adhoc programs, and Historical programs. This doesn't mean there aren't other important groups that could be broken out further and identified. A clearly identified location for such programs is paramount

Production program directories should contain those programs that are well tested and utilized on a regular basis. These programs are usually the basis for all final data manipulation and reporting. Frequently such locations are well protected and there are rigorous rules to update and alter the programs there. These may be broken out further into program groupings such as Data collection, Manipulation, and Reporting. As mentioned previously, it is frequently helpful to include a text document in each storage location specifying methods and rules to update the programs, location of supporting documentation, key staff that are knowledgeable about the process, as well as anything else that will quickly lead someone to the proper documentation and critical answers..

Additional directories or locations for storing retired programs should be diligently maintained. Too frequently events occur that require rerunning old production programs. Well-maintained storage of those programs can make a hero out of the programmer who is able to easily recreate a previous scenario.

Testing Programs may or may not be utilized in all shops. If this is a requirement, then a clear location to safely develop and work with test programs and updates is a necessity. The location should provide a self-contained environment to include updates and changes while testing without jeopardizing the integrity of production programs and data. Testing libraries and locations are traditionally managed with less protocol and safety precautions.

Adhoc program locations are usually handled with the same, less restricted rules as those for Testing programs. It also needs to be a safe environment for code development, testing, and then producing results without jeopardizing the Production process and its programs' integrity.

Support Utilities come in all shapes and sizes. Everything from formats, call routines/macros, and other standard and reusable utilities should be stored in one central location. If a utility will be used by more than one program, it needs to be stored in just one project-specific directory/library. If updates are necessary, it's far easier to locate and manipulate changes if it only needs to occur in one location. For example, if there are a set of formats that many programs utilize, then a standard location enhances accessibility and allows a single change rather than multiple changes to multiple programs, thereby allowing faster turn-around and reducing the chance of coding errors and mistakes.

There are several common threads to each of the storage 'units'. Historical data clearly identified as such provides a valuable path to the past. Including text documents within directories guides a search and will answers at least basic questions without a great deal of time spent finding and reviewing lengthy documents. Avoid complicated directory structure schemes, and take advantage of long directory and file names. In Windows you have 256 characters to work with, and in a UNIX environment you can use underscores to separate words. Whatever you do, the structure needs to be easily identified and navigated. The more effectively storage is designed; the easier it will be for everyone who uses it.

DOCUMENTATION

To effectively perform Data Quality Management, Data Cleansing, and Discrepancy Reporting as much documentation as possible is the ideal. There's no such thing as too much documentation, providing the document and file names make sense. That said, there are several items from a programmer's perspective which are more useful than others, will speed understanding, and enable the ability to write code with more ease. My particular favorites are data dictionaries, flow diagrams, and metadata.

Data Dictionaries

Much has been written and discussed about Data Dictionaries. They need to be clear, concise, and comprehensive. This is one time when too much information, especially items that do not provide information needed for coding, can slow you down. Below are the items that are necessary and effective:

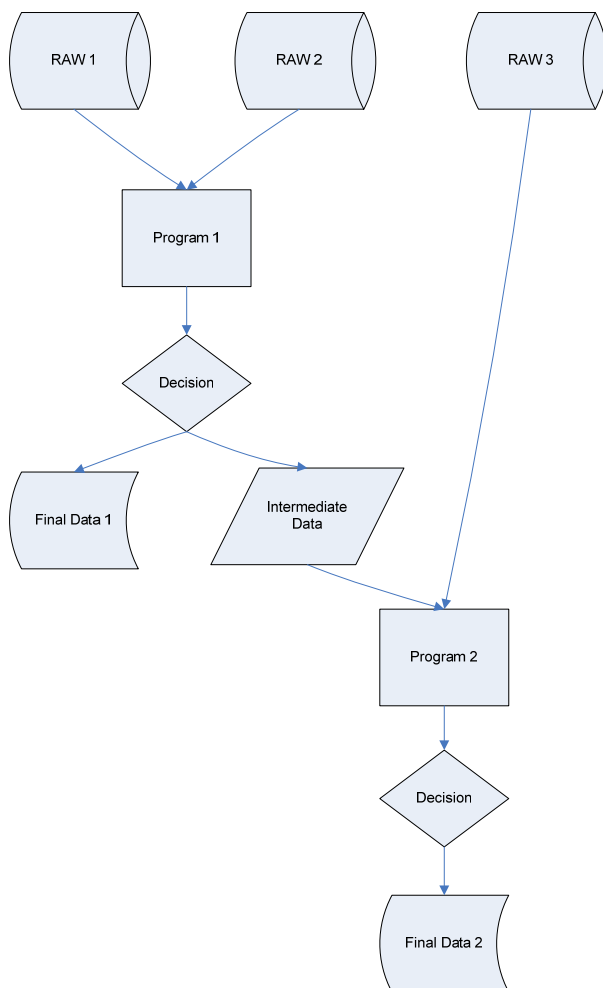
- Dataset name, location, and last update
- Dataset description (label)
- Variable name
- Variable description (label)
- Variable type (character/numeric/date)
- Valid values and ranges (this might be different from the values that are actually in the data)
- Formats applied

The data information should be listed in either alpha-numeric order or in a logical sequence appropriate to the order the data is interpreted or used. For example, all variables related to geographic location may be grouped together even though the names don't follow one another in alpha order.

The most helpful items one can use when starting a new project or task are flow diagrams, also known as a flowchart. This is a visual representation of the relationships between files, programs, and data and can greatly speed up understanding and programming efforts. Continued updates to and maintenance of these diagrams should be a standard, normal routine. One can usually find flow diagrams prominently displayed at a programmers desk for quick reference. They also facilitate discussions with others.

When creating a diagram of data relationships there are a number of key factors that should be included. The flow for a data integration and transformation process begins with the original data sources and continues through in step-by-step order to the final and complete version datasets. The logical flow of one source into another and the high-level programming steps combined with sub-setting flows of subsequent datasets is the key to an effective diagram.

Next is an example of a Data Flow diagram. They can often become quite elaborate and contain multiple pages, but nothing can aid in the initial understanding of a project or task better than a view of the big picture.



Additional diagrams showing relationships between data and processes could include the logic surrounding the connections between each. In the clinical trial world, you may have a screening dataset and the next logical dataset an enrollment dataset. While one flows into the next, not every participant in the screening file is necessarily in the

enrollment dataset. But every participant in the enrollment dataset must be in the screening dataset. This type of logic and understanding will be heavily necessary in the further steps of Data Cleansing.

Metadata is another “go to” source to help in a programming task. A simple Proc Contents of a dataset in combination with an abbreviated print out of the actual data provides a great deal of useful information. In fact, it is highly recommended when working with new data to first start with this step before moving forward with any programming.

```
proc contents data=sashelp.shoes;

proc print data=sashelp.shoes (obs=5);
```

Most of the information created by the two simple PROCs above is quite helpful. The file name, number of observations, whether the data is sorted, and the creation date are some items to look at first. If this information doesn't contain the expected values, the programmer has saved much time in realizing they're using incorrect data.

The CONTENTS Procedure

Data Set Name	SASHELP.SHOES	Observations	395
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	Wednesday, May 12, 2004 10:54:45 PM	Observation Length	88
Last Modified	Wednesday, May 12, 2004 10:54:45 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Fictitious Shoe Company Data		
Data Representation	WINDOWS_32		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	5
First Data Page	1
Max Obs per Page	92
Obs in First Data Page	71
Number of Data Set Repairs	0
File Name	C:\Program Files\SAS\SAS 9.1\core\sashelp\shoes.sas7bdat
Release Created	9.0101M3
Host Created	XP_PRO

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
6	Inventory	Num	8	DOLLAR12.	DOLLAR12.	Total Inventory
2	Product	Char	14			
1	Region	Char	25			
7	Returns	Num	8	DOLLAR12.	DOLLAR12.	Total Returns
5	Sales	Num	8	DOLLAR12.	DOLLAR12.	Total Sales
4	Stores	Num	8			Number of Stores
3	Subsidiary	Char	12			

Nothing can take the place towards understanding the information you are going to be working with than physically looking at it. What you think may be contained in a field based on its description may actually be something rather different.

Printout of Data

Obs	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
2	Africa	Men's Casual	Addis Ababa	4	\$67,242	\$118,036	\$2,284
3	Africa	Men's Dress	Addis Ababa	7	\$76,793	\$136,273	\$2,433
4	Africa	Sandal	Addis Ababa	10	\$62,819	\$204,284	\$1,861
5	Africa	Slipper	Addis Ababa	14	\$68,641	\$279,795	\$1,771

DATA CLEANSING

What is Data Cleansing? It is the task of verifying as clearly as possible file to file dependencies as well as validating individual variable value content. To do this effectively, a clear understanding of the business rules is crucial. This task is by no means easy, but it sure is necessary. It may take on many forms and directions depending on the nature of your business. Utilizing clear data dictionaries as well as business documentation are the tools most used to accomplish this effort.

File to file dependencies will utilize the program flows designed earlier. If an account/participant is in one file, can it also be found in the other files the business rules determine they should be located? In reverse, are accounts/participants that should be dropped continued on into further datasets when they shouldn't exist? Too frequently raw or original data may contain duplicates or inaccuracies that cause such transfer of information to be inappropriate. Programs which identify such discrepancies should be included in the process.

The more tedious task of validating individual data fields for accurate information requires a superb Data Dictionary. Numeric fields should contain numeric data. Date fields should contain valid dates. Text fields should contain the exact type of information within them as expected.

Taking this a step further, a numeric field usually has specific valid inputs. If a credit score may only be within 0-1000, then a test on that field should include identifying those records that don't fall into the acceptable limits. Dates as well will have acceptable and unacceptable values. If a Date of Birth is included, a date can't be in the future, is unlikely to be January 1st, 1960 or over 75 years ago. Text fields tend to be more difficult. Obviously a first name field may contain any combination of letters imaginable. A State field should only contain a valid USA state, otherwise, it's incorrect. Moreover, any of the 3 variable types may have content that should always be included while others may reasonably be blank. This information should be clear in the Data Dictionary or other supporting documentation.

Cleaning data thoroughly tends to be one of the less appreciated tasks. The results usually lead to additional research and the identification of additional problems. The "Data Cleanser" becomes the bearer of bad news. Just remember, the cleaner the data, the more accurate the reporting.

DATA MANAGEMENT

Many inherit the data they work with day in and day out. For those lucky enough to have input into data creation and data output, there are some basic rules of thumb to follow.

1. If a field that will be used for matching or is contained in more than one file but represents the exact information (such as an ID), the names and lengths of the variables should be exactly the same.
2. If a variable contains numeric data, specify the field as numeric.
3. For character fields, insure they are either upper case or lower case. Avoid mixed case fields.
4. For date variables, make it a SAS date field rather than a text field.
5. Use variable names that make sense like ID, gender rather than Var1, Var2.

Adhering to these five rules will minimize much wasted programming time. There have been far too many instances of programmers struggling to match datasets when the data structures and above rules aren't applied. The effort to reformat, rename, change lengths, and make fields match requires time and effort that could be better spent elsewhere.

Here are some simple tools that could be used in case you do have problems due to the previous rules having not been applied. They are easy programming solutions to deal with the frustrating disassociation of data.

```
data one;
    var1 = 'four';
    id = 4567;
    otherid = '4567';
    startdate = '01/23/1963';
run;

data two;
    length var1 $6.;
    * Change length from $4 to $6;

    set one(rename=(otherid=othid id=custid startdate=stdate));    * Rename
variables so they may be altered;

    otherid = othid*1;
    * Convert character to numeric after rename;

    var1 = upcase(var1);
    * Alter the case of a variables contents;

    startdate = input(stdate,mmddyy10.);
    *Convert text date to SAS date;

    drop othid stdate;
run;
```

DISCREPANCY REPORTING

With the ground worked laid for excellent clean data, unanticipated problems still arise, producing unexpected results. Managing and reporting such instances are the most proactive task a programmer may perform in good faith with those whom rely on the information delivered. There is no one way to perform this task. Having said that, here are several suggestions:

- Read your logs! More programmers skip this step more than any other. There are a number of already created SAS programs/routines that may be utilized to check logs and only report errors when they occur. Such programs can be found in several books, on SAS-L, and other conference papers. Still, there are clean logs that show valuable information such as match/merge counts, dataset counts, etc. that still require a vigilant eye.
- Create a dynamic log of issues as they become apparent. It should contain the problem, the date of its identification, the person/process that identified the problem, the eventual solution and its date, and those

that were involved in resolving the issue. Report programs' listing of the status of such an error reporting tool becomes a useful endeavor.

- Programs written to validate file record counts in comparison with previous versions are another positive action to insure when hiccups occur along the way they are identified early and resolutions quickly activated.
- Any other tool that may be utilized to catch problems before they catch you.

Conclusion

The underlying purpose of this paper is to provide suggestions and tools to make our lives as programmers easier. So much time can be wasted trying to locate correct data, information about processes, identifying problems with data and correcting the mistakes of poor Data Management decisions made in the past. Each site and industry has their own specific set of issues and procedures to deal with inaccurate data. But inaccurate data is not special to any one industry or SAS programming shop. Being able to have the ability to influence the creation of data before it becomes our responsibility, if possible, is a great way to alleviate difficulties later on. If there is one suggestion that could be gleaned from the study of Data Management, it would be to "understand your data completely".

Disclaimer

The thoughts and recommendations presented in this paper are strictly those of the author's and is in no way a reflection of the opinions or ideas of another person or organization. They are based on the author's many years programming in SAS, providing training internationally, and having worked in many industries

ACKNOWLEDGMENTS

A big "Thank you" to several friends that helped me prepare this paper and presentation. A special thanks to my friends in Cameroon and Kenya for allowing me to practice this presentation in their attendance

CONTACT INFORMATION

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

Jenine Milum
The Ginn Group (CDC)
Atlanta, Georgia
E-mail: JenineMilum@bellsouth.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.