Validating User-Submitted Data Files with Base SAS®

Michael A. Raithel, Westat

Abstract

SAS programming professionals are often asked to receive data files from external sources and analyze them using SAS. Such data files could be received from a different group within one's own organization, from a client firm, from a Federal agency, or from some other collaborating establishment. Whatever the source, there is likely to be an agreement on the number of variables, variable types, variable lengths, range of values, and variable formats in the incoming data files. Information Technology best practices require that the receiving party perform quality checks (QC's) on the data to verify that they conform to the agreed-upon standards before the data are used in analysis.

This paper presents a rigorous methodology for validating user-submitted data sets using Base SAS. Readers can use this methodology and the SAS code examples to set up their own data QC regimen.

Introduction

This paper introduces the User Data File Template Validation Methodology (UDFTVM). The UDFTVM is a protocol that you can employ in order to validate SAS data sets you receive from external sources. It consists of a series of distinct steps that lead you through a process of establishing a template SAS data set, and then comparing an incoming data set to that template in order to determine its conformity to established standards as well as its completeness.

The UDFTVM is composed of the following five steps:

1. **Obtain the SAS data set's metadata and variable value ranges**. You need to request that your user provide you with specific information about the data set you are to receive, including the number of variables, variable types and names, and other pertinent metadata. You also require information about the range of permissible values for specific variables when they are germane to the analysis you are tasked with performing.

2. **Create a template SAS data set**. Using the information you gather from **Step 1**, you create a "template" of the data set you expect to receive with all of the characteristics that were established. Incoming data sets will be compared against the template to determine how complete they are and how well they conform to its implicit requirements.

3. **Verify the data file's structural metadata with PROC COMPARE**. This step does the following:
   - Verifies that all required variables are in the data set
   - Validates the variables' types, lengths, formats, and labels
   - Determines if there are extra variables in the data set
   - Determines if you have data in the data set

4. **Check for valid values using integrity constraints and audit files.** This step does the following:
   - Creates integrity constraints for the template SAS data set
   - Creates an audit file for the template SAS data set
   - PROC APPENDs the incoming SAS data set against the template SAS data set
   - Notes any observations that dropped out due to violated integrity constraints

5. **Report the results to the entity that submitted the data**. In this step, you communicate the validity of the incoming data set to the entity that submitted it. You report any problems and provide details so that they can correct them. You also let your users know when their data sets successfully passed the QC's.

Completing these five steps provides you with assurance that a new data set either conforms to the established requirements or needs some adjustment in order to do so. When a newly-received data set is fully validated, you can have confidence that the analysis you perform on it will produce solid results based on the data that your client wants to have analyzed.

The UDFTVM is especially ideal for situations where a group of users are charged with sending you individual data files with the same specific layout on a cyclic basis—daily, weekly, monthly, etc. Having a template that all users must conform to helps you to enforce an established standard for the files you receive. You can automate the latter steps of the UDFTVM process to facilitate quick turnaround of your QC process. Early identification and reporting of issues with a data set allows your users to modify, test, and rerun the programs that created that data, and then resend it to you in a timely manner.

The following sections of this paper describe the five steps of the User Data File Template Validation Method.

Step 1: Obtain the SAS data set's metadata and variable value ranges.

The first step of the UDFTVM is to communicate with the organization that is going to send you files in order to obtain specific information about their characteristics. Take the initiative and work with your users to establish the following information about the variables in each data set you expect to receive:

- Variable name
- Variable type
- Variable length
- Variable format
- Variable label
- Unique or non-unique values
- Is a missing value allowed
- Valid values or valid value ranges

There are often some already-decided rules about the values of variables within user-supplied data sets. For example, the values of some variables may have to be unique throughout the entire data set. Some variables are categorical and are only supposed to contain a predefined list of distinct values. Some variables hold measurements and should only have values in a specific range. Other variables may be able to hold an unlimited number of values, but are not supposed to ever contain missing values. Consequently, you need to establish the set of rules for the values of variables in the user-supplied data set. That way, you can test the incoming data set to verify that the values adhere to the criteria that has been established in order to complete the UDFTVM process and move on to the work of analyzing the data.

You should ask your users to provide you with the set of rules for the values of variables in the data set. Here are some of the variable characteristics that you should know about:

- **Key variables –** Key variables are one or more variables that uniquely define a record in a data set. Examples are Subject ID, Patient Number, Student ID, and Social Security Number. You should know which variable or variables are the keys and are therefore supposed to have unique values throughout the data set.

- **Categorical variables –** These are qualitative variables used to categorize data. For example, the allowable values for the variable *Employed* may be: 1 for "Yes", 2 for "No", and 0 for "Unknown". Your user should specify the exact values that are permissible for each categorical variable.

- **Discrete variables –** The values of discrete variables are specific and represent a count, such as number of doctor visits, parts, or units sold. You are going to need the range of discrete variables' values, so that you can determine if values you get in the new data set are reasonable or not.

- **Continuous variables –** Continuous variable values represent a measure, such as weight or height. The user should provide you with the valid ranges for continuous variable so that you can determine when values you get in the incoming data set are suspect.

In addition to getting a set of permissible values or ranges of permissible values for the variables, you also need to know which variables should not have missing values in them.

The best way to get all of the aforementioned information is to ask your users to supply it in writing. If the users have a codebook or some other document that maps variables to their exact values or range of acceptable values, then they should send it to you. Once you have documentation, you can move on to adding this information to the SAS program that checks for the specified data constraints.

This type of discussion often requires multiple iterations before you get a clear picture of the characteristics of all of the variables. Ask for the information in written form and then make sure that it is complete for each variable. Once you have an agreement, send your user a document with the complete list of variables and their approved characteristics. Working with your users to ratify the understanding of how the variables should be mapped is key to the quality check steps that you will run once you receive the data sets.

Step 2: Create a Template SAS Data Set

The next step in the UDFTVM is to create a template SAS dat set. The template SAS data set represents the ideal data set that you want to have your user submit to you. As such, it contains all of the variables you expect, including the correct labels, formats, informats, and variable types. The template SAS data set will be used to measure the acceptability of the incoming data set. So, the user-submitted data set is expected to have the exact same number of variables—with their attendant types, lengths, formats, informats, and labels—as the template SAS data set. If not, then the incoming data set does not conform to the standards that have been set for it and will have to be rejected.

Armed with complete information about each variable in the input data set from **Step 1 of the UDFTVM**, you can create the template SAS data set.

Here are two ways that you may choose to create the template SAS data set:

1. Creating a Template SAS Data Set from an Existing SAS Data Set

If you have an existing SAS data set that has all of the variables and variable attributes that you expect from the incoming data set, you can clone it to create the template SAS data set. Here are two examples:

Example 1

```
data templib.MedTemplate1;
set  prodlib.MedData;
stop;
run;
```

In this example, we create a SAS data set named MedTemplate1 from existing SAS data set MedData. MedTemplate1 will have all of the variables in MedData as well as their attributes such as data type, label, and format. Because the STOP statement was placed after the SET statement, the MedTemplate1 SAS data set will not contain any observations.

Example 2

```
proc sql;
create table templib.MedTemplate1
        like prodlib.MedData;
quit;
```

This example uses PROC SQL to copy the attributes of an existing table to a new, empty table. It uses the LIKE clause in the CREATE TABLE statement to create table MedTemplate1 in the image of existing table MedData.

2. Creating a Template SAS Data Set from Scratch

If you do not have a data set that you can clone to create a template data set, then you will have to construct one from scratch. To do that, use common statements in a DATA step to create the variables you need. Here are two examples:

Example 1

```
data templib.classTemplate1;

attrib name   length=$8 label=’Name’
       sex    length=$1 label=’Sex’
       age    length=8  label=’Age’         format=3.
       height length=8  label=’Height CM’   format=5.1
       weight length=8  label=’Weight lbs’  format=5.1
       ;
stop;
run;
```

In this example, we use the ATTRIB statement to specify the variables’ names, lengths, labels, and formats. We could have just as easily used separate LENGTH, LABEL, and FORMAT statements. However, having all of a particular variable’s attributes on the same line can sometimes be more convenient from a documentation standpoint.

Example 2

```
proc sql;

create table templib.classTemplate1
        (name   char(8) label=’Name’,
         sex    char(1) label=’Sex’,
         age    num     label=’Age’         format=3.,
         height num     label=’Height CM’   format=5.1,
         weight num     label=’Weight lbs’ format=5.1
        );
quit;
```

This example uses the SQL procedure to specify the variables’ names, lengths, labels, and formats. It is completely analogous to the previous example and produces the same result: a table with five columns and zero rows.

When you have created the template SAS data set, take some time to verify that it conforms to the criteria you received from your users. Be sure that you have the approved number of variables, that you have the correct variable names, and that you have the agreed-upon metadata (variable type, label, format, and informat) for each of the variables.

When you have successfully created the template SAS data set then you are done with **Step 2 of the UDFTVM**.

Step 3: Verify the Data File’s Structural Metadata with PROC COMPARE

PROC COMPARE is a SAS procedure that compares two SAS data sets and reports on all of the differences between them. It can identify differences in the structural composition of the two files such as dissimilarities in metadata and discrepancies in the number of variables in each file. The COMPARE procedure can also report on the differences of values in the same variables for matching observations between two data sets. PROC COMPARE is a powerhouse of a comparison tool and has many options that can be specified to either zero in on a particular type of comparison or to limit the types of comparisons that are being done.

The UDFTVM uses the COMPARE procedure to check the structural metadata of the user-submitted data set. It compares the incoming data set against the template to:

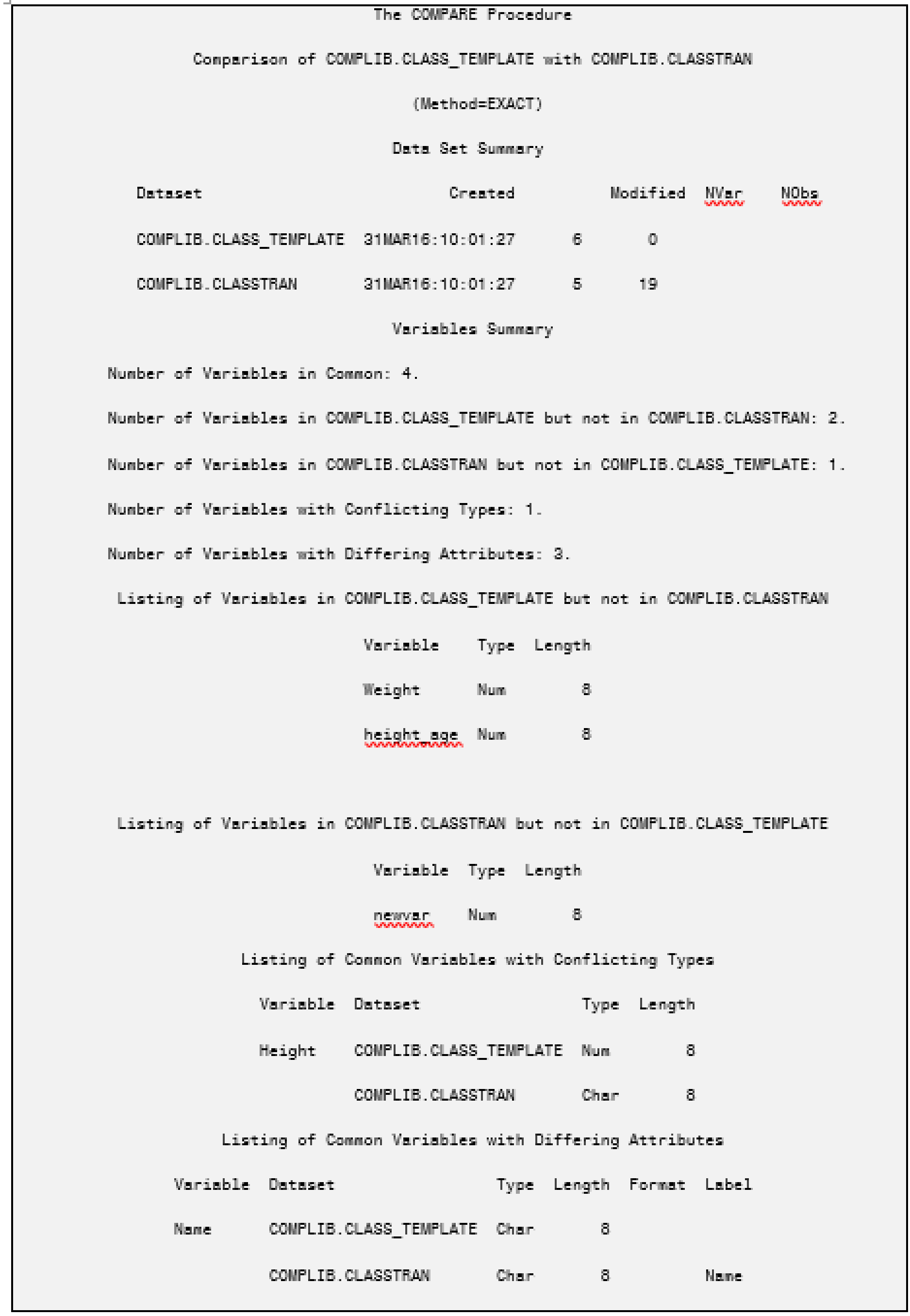
- Verify that all template variables are in the user data set
- Determine if there are extra variables in the user data set
- Validate that these variable attributes match:
    - type
    - length
    - format
    - informat
    - label
- Determine if there is any data in the user data set

Here is an example of using PROC COMPARE to compare two data sets:

```
proc compare base=complib.class_template
             compare=complib.classtran
             LISTVAR;
title1 "Testing the Data Set's Structural Metadata";
run;
```

In this example, the template SAS data set is CLASS_TEMPLATE and resides in the COMPLIB SAS data library. We are comparing the user-submitted CLASSTRAN SAS data set to the template SAS data set. We include the LISTVAR option to specify that we want to list all variables that are found in only one of the two data sets, signifying a mismatch in the basic structure of the files.

This example produces the report in Figure 1.



```
                              The COMPARE Procedure

               Comparison of COMPLIB.CLASS_TEMPLATE with COMPLIB.CLASSTRAN

                                  (Method=EXACT)

                                Data Set Summary

     Dataset                          Created          Modified  NVar    NObs

     COMPLIB.CLASS_TEMPLATE  31MAR16:10:01:27      6       0

     COMPLIB.CLASSTRAN       31MAR16:10:01:27      5      19

                                Variables Summary

  Number of Variables in Common: 4.

  Number of Variables in COMPLIB.CLASS_TEMPLATE but not in COMPLIB.CLASSTRAN: 2.

  Number of Variables in COMPLIB.CLASSTRAN but not in COMPLIB.CLASS_TEMPLATE: 1.

  Number of Variables with Conflicting Types: 1.

  Number of Variables with Differing Attributes: 3.

   Listing of Variables in COMPLIB.CLASS_TEMPLATE but not in COMPLIB.CLASSTRAN

                            Variable    Type  Length

                            Weight      Num        8

                            height_age  Num        8


   Listing of Variables in COMPLIB.CLASSTRAN but not in COMPLIB.CLASS_TEMPLATE

                             Variable  Type  Length

                             newvar    Num        8

                  Listing of Common Variables with Conflicting Types

                     Variable  Dataset                 Type  Length

                     Height    COMPLIB.CLASS_TEMPLATE  Num        8

                               COMPLIB.CLASSTRAN       Char       8

                 Listing of Common Variables with Differing Attributes

         Variable  Dataset                 Type  Length  Format  Label

         Name      COMPLIB.CLASS_TEMPLATE  Char       8

                   COMPLIB.CLASSTRAN       Char       8          Name
```

Figure 1 – PROC COMPARE of user-submitted data set to template data set.

The resulting PROC COMPARE report reveals several issues with the user's data set as illustrated in Figure 1. First, the incoming data set is missing the *Weight* and *height_age* variables that are in the template SAS data set. Secondly, the user's data set has an extra variable, *newvar*, which is not in the template SAS data set. Thirdly, the variable *Height* on the incoming data set has a *Type* of Char (character) while the template SAS data set has the *Height* variable specified as Num (numeric). Finally, the *Name* variable in the user's data set has a label while the one in the template does not.

In this example, we have established that there are significant differences between the two data sets. So, no further comparisons or checks need to be done. We can proceed to **Step 5 of the UDFTVM** and report the results to the entity that submitted the data set. Our feedback will allow the users to examine and modify their data set creation procedures in order to re-create a data set that conforms to the standards that were established. When they submit a modified data set, it will go through the same vetting process, starting back here at **Step 3 of the UDFTVM**.

At some point, the goal of having a clean structural metadata compare between the data sets will be achieved. When it is, then the following sections *will not* appear in the PROC COMPARE report:

- Listing of Variables in COMPLIB.CLASS_TEMPLATE but not in COMPLIB.CLASSTRAN
- Listing of Variables in COMPLIB.CLASSTRAN but not in COMPLIB.CLASS_TEMPLATE
- Listing of Variables with Conflicting Types
- Listing of Common Variables with Differing Attributes

When none of these sections are produced, check your SAS log to verify that the PROC COMPARE ran without issues. When that is the case, you are done with checking the new data set's structural metadata and can move on to **Step 4 of the UDFTVM**.

Step 4: Checking for Valid Values Using Integrity Constraints and Audit Files

The most rigorous and programmatic method of reviewing the validity of a set of variables' values is through the use of SAS integrity constraints and audit files. This section of the paper is divided into four sub-sections that show you how to do just that. The sub-sections are:

1. **What Are SAS Integrity Constraints?** – Provides a basic overview of SAS integrity constraints
2. **What Are SAS Audit Files?** – Presents a simple summary of SAS audit files
3. **Overview of the Methodology** – Furnishes an overview of how integrity constraints and audit files are used to vet the data
4. **SAS Program Examples** – Presents two SAS program examples of checking for valid values using integrity constraints and audit files.

If you are already knowledgeable about SAS Integrity Constraints and Audit Files, consider skipping down to the section titled: **Overview of the Methodology**.

What Are SAS Integrity Constraints?

Integrity constraints are built-in data set validation rules that have their roots in the world of SQL programming. They are a set of rules used to restrict the values stored in variables in SAS data sets. You can use the DATASETS procedure to create integrity constraints (rules) that limit the values that can be stored in variables in a SAS data set. SAS then enforces those rules whenever observations are added, modified, or deleted from the data set.

There are two major categories of integrity constraints: Referential and General. Referential integrity constraints exist between two or more SAS data sets. Since we do not use Referential integrity constraints in the UDFTVM process, they will not be discussed further. General integrity constraints exist for the variables within a file. There are four possible General integrity constraints:

- **check** – Limits the values in a variable to a range, set, or list of values. You can also limit the values of a variable depending upon the value of another variable in the same observation. For instance, if Gender equals "Male", then Pregnant must equal "No".
- **not null** – Specifies that a variable cannot contain missing values
- **primary key** – States that all occurrences of this variable or set of variables in the SAS data set must be unique. There can only be one *primary key* for a SAS data set. Customer ID, Part Number, and Social Security Number are all examples of typical *primary keys*.
- **unique** – Specifies that all occurrences of this variable must have unique values within the data set. This is similar to the *primary key* constraint. However, there may be many variables with the *unique* constraint, but only one with the *primary key* constraint.

Here is an example of creating integrity constraints for the SHOES SAS data set:

```
proc datasets library=udftvm nolist;

modify shoes;
        ic create pkey = primary key (storenumber);
        ic create regprodsub = distinct (region product subsidiary)
                  message = "Region, Product, Subsidiary combination must be unique";
        ic create storelimit = check(where=(stores < 50))
                  message = "Limit of 50 stores";
        ic create returnsales = check(where=(returns+sales <= inventory))
```

```
                         message = "Returns + Sales cannot exceed Inventory";
        run;

        quit;
```

In the example, above, we create four integrity constraints for the SHOES SAS data set. Note that we provided a name for each of them. The five integrity constraints are:

- **pkey** – States that variable STORENUMBER is the primary key for the SHOES SAS data set. So, all values of STORENUMBER must be unique within the data set.
- **regprodub** – Specifies that the combined values of REGION, PRODUCT, and DISTRICT must be unique (DISTINCT) for all observations within the SHOES SAS data set. Note, that we could have used the UNIQUE keyword instead of the DISTINCT keyword.
- **storelimit** – Limits the value of STORES to less than fifty for all observations in the data set.
- **returnsales** – Limits the value of RETURNS plus SALES to be less than or equal to the value of INVENTORY for each observation.

<u>What Are SAS Audit Files?</u>
You can use the AUDIT statement of the DATASETS procedure to initiate using an audit trail for a particular SAS data set. An audit trail is a special SAS file that keeps track of which observations are added, deleted, or modified in a SAS data set. By creating an audit file, you can determine who modified the data set, when it was altered, and what was changed. In the context of the UDFTVM, we will use the SAS audit file to hold observations from the user data file that were "rejected" because they did not meet our established validation criteria.

When you create an audit trail for a SAS data set, SAS automatically creates a new file with the same name as the original SAS data set, but with the file extension of ***.sas7baud***. For example, if you create an audit trail for the SNACKS SAS data set, the audit trail file will be named SNACKS.sas7baud. The audit trail file is created in the same directory as the original SAS data set. It remains there until you use the TERMINATE option in the DATASETS procedure, at which time it is deleted and auditing ceases for the specified SAS data set.

The audit trail file is automatically set up to contain each variable in the original SAS data set as well as six special audit trail variables. Those variables begin with "_AT" and describe aspects of the SAS variables that ended up in the audit file. The _AT variables are:

• _ATDATETIME_	- The date/time of the modification
• _ATUSERID_	- The UserID associated with the modification
• _ATOBSNO_	- The observation number affected by the modification
• _ATRETURNCODE_	- The event return code
• _ATMESSAGE_	- The SAS log message of the modification
• _ATOPCODE_	- The code describing the type of modification

Step 4 of the UDFTVM only uses _ATMESSAGE_. The other _AT variables are described here so that you can have an understanding of them in case you print all of the variables in an audit file during your work within this step of the UDFTVM.

Here is an example of creating an audit file for the SNACKS SAS data set:

```
proc datasets library=udftvm nolist;

        audit snacks;
                initiate;
                log admin_image=yes
                    before_image=yes
                    data_image=no
                    error_image=yes;
        run;

        quit;
```

In this example, the LOG option was used to specify the four possible audit settings:

- **admin_image** – States whether or not the SUSPEND and RESUME administrative actions are logged to the audit file.
- **before_image** – States whether the before-image of updated observations are recorded to the audit file.
- **data_image** – States whether the after-image of added, updated, and deleted observations are recorded to the audit file.
- **error_image** – States whether the error images are recorded in the audit file.

For purposes of the UDFTVM, we will specify YES to all audit setting except *data_image*, which we code to NO. We do this because we only want to have a single copy of the observations that are rejected by integrity constraints in our audit file. We want the "before image" of rejected observations. Note that this concept should become clearer in the example in the section *SAS Program Examples*, below.

You can determine which variables are in a SAS audit trail data set by using the CONTENTS procedure. Use the TYPE= option on the DATA statement to specify the AUDIT data set. Here is an example:

```
proc contents data=udftvm.snacks(type=audit);
run;
```

This is the *Alphabetic List of Variables and Attributes* from the CONTENTS output:

```
     Alphabetic List of Variables and Attributes

#    Variable           Type    Len    Format
3    Advertised         Num       8
5    Date               Num       8
4    Holiday            Num       8
2    Price              Num       8
6    Product            Char     40
1    QtySold            Num       8
8    _ATDATETIME_       Num       8    DATETIME19.
13   _ATMESSAGE_        Char      8
9    _ATOBSNO_          Num       8
12   _ATOPCODE_         Char      2
10   _ATRETURNCODE_     Num       8
11   _ATUSERID_         Char     32
```

Notice that the SAS audit trail data set contains all of the variables in the SNACKS data set as well as the six "_AT" variables.

Equipped with a fundamental knowledge of SAS audit data sets, we will move on to how they are used in conjunction with SAS integrity constraints to validate the values of variables in the user-submitted data file.

Overview of the Methodology

The methodology used to check for valid values using integrity constraints and audit files has five steps:

1. **Create a *target* SAS data set by copying the template data set that was created in *Step 2 Create a Template SAS Data Set*.** The target SAS data set will have all of the characteristics of the template data set. It will have the agreed-upon variables and have all of those variables' features—their lengths, formats, labels, etc. The target SAS data set will be created with no observations in it.

2. **Create an audit file for the target SAS data set**. The audit file will be used to house observations that are rejected by the integrity constraints in step 4. It is important for our reporting purposes to set *data_image = no* and the other three options to yes when creating the audit file.

3. **Create an integrity constraint for the target SAS data set**. The integrity constraint characterizes one of the data validation rules your users have provided for this data set.

4. **Append the user SAS data set to the target SAS data set**. Using PROC APPEND, specify the target SAS data set in the BASE= parameter and the user SAS data set in the DATA=parameter. This causes SAS to attempt to append the observation in the user data set to the empty target data set. Any observations that violate the integrity constraint created in step #3 are not written to the target data set. More importantly, those rejected observations are written to the target's audit SAS data set.

5. **Create a report of observations that do not match the validation criteria**. Use the PRINT or the REPORT procedure to create a report listing the observations from the user SAS data set that violated the data validation rules set forth in the integrity constraint. You do this by printing selected variables from the audit SAS data set. You should include identifying information such as the key variable values for the observations in question as well as the variable with the erroneous values in it. It is also helpful to specify the integrity constraint rule that was violated.

These five steps must be repeated for each individual data validation rule that you are going to test for using integrity constraints. The reason for this is that SAS immediately rejects an observation when it detects that it has violated the first of several integrity constraints. Consequently, an observation that has errant data which would trigger several integrity constraint contraventions is only flagged for the first such violation. This means that you must specify one integrity constraint at a time or risk not knowing that a particular observation would have been invalidated by multiple integrity constraints. For this reason, SAS data sets with many variables may require many passes of these five steps—one pass for each validation rule specified for each variable.

The following two example SAS programs execute all five steps of the methodology just described. The first example has a single validation rule and hence a single integrity constraint. The second example illustrates how a SAS macro can be used to iterate through the five steps to specify and check several integrity constraints for a single user-supplied data set.

SAS Program Examples

In this first example, our user sent us a SAS data set named ORSALES containing 918 observations and 8 variables. We have already built a template data set named orsalesTemplate and successfully vetted the new ORSALES data set via the PROC COMPARE in **Step 3 of the UDFTVM**. There were no discrepancies. So, the next step is to check the file for invalid values. We have an agreement with the user that they will never send us an analysis data set that has observations where the value of Product_Group is equal to "LSF." So, we create an integrity constraint that directs SAS to reject any observations where Product_Group equals "LSF" and test the ORSALES data set for that particular condition.

This is what the program looks like:

1

```
/* Use the Template data set to create a target SAS data set*/
data udftvm.orsalesTarget;
set  udftvm.orsalesTemplate; /* <--This is the template SAS data set*/
stop;
run;
```

2

```
/* Specify an audit trail for the target SAS data set*/
proc datasets library=udftvm nolist;
        audit orsalesTarget;
        initiate;
        log admin_image=yes
            before_image=yes
            data_image=no
            error_image=yes;
     run;
quit;
```

3

```
/* Specify an Integrity Constraint with the validation criteria*/
proc datasets library=udftvm nolist;
modify orsalesTarget;
        ic create GROUP01 = check(where=(Product_Group ne "LSF"))
         message = "GROUP01 - Product Group cannot be LSF" msgtype=user;
quit;
```

4

```
/* Append the user data set to the empty data set to test the validation rules. */
proc append base=udftvm.orsalesTarget
            data=udftvm.orsales; /* <--This is the user SAS data set*/
run;
quit;
```

5

```
/* Create a report of observations not matching validation criteria. */
ods pdf file="G:\My report directory\Observations That Violated the Validation Criteria for
Product_Group.pdf";

proc print noobs data=udftvm.orsalesTarget(type=audit) label blankline=10;
var StoreID Product_Group;
```

```
title1 "Observations That Violated the Validation Criteria:";
title2 "GROUP01 - Product Group cannot be LSF";
run;

ods pdf close;
```

Here is what is happening with each step numbered above:

1. We create the ORSALESTARGET SAS data set using the ORSALESTEMPLATE data set. ORSALESTARGET will have all of the characteristics of ORSALESTEMPLATE and have zero observations in it.

2. The audit trail for the ORSALESTARGET target SAS data set is created via the DATASETS procedure. Any observations rejected by the upcoming PROC APPEND will be written to the audit SAS data set.
   The log from this step looks in part like this:

   ```
   37   /* Specify an audit trail for the target SAS data set*/
   38   proc datasets library=udftvm nolist;
   39       audit orsalesTarget;
   40       initiate;
   WARNING: The audited data file UDFTVM.ORSALESTARGET.DATA is not password protected.  Apply an
            ALTER password to prevent accidental deletion or replacement of it and any associated
            audit files.
   41       log admin_image=yes
   42       before_image=yes
   43       data_image=no
   44       error_image=yes;
   45   run;

   NOTE: The data set UDFTVM.ORSALESTARGET.AUDIT has 0 observations and 15 variables.
          46   quit;
   ```

   The warning message is routine and nothing to be concerned about for our purposes. Note that the audit data set has zero observations in it at this time.

3. We specify the integrity constraint for the ORSALESTARGET SAS data set. Remember that our only restriction was that Product_Group is never supposed to be equal to "LSF." Knowing that, we create an integrity constraint that specifies this limit. The message "Group01 – Product Group cannot be LSF" will be written to the SAS log if any violating observations are encountered in the upcoming PROC APPEND.

   The log from this step looks in part like this:

   ```
   48   /* Specify an Integrity Constraint with the validation criteria*/
   49   proc datasets library=udftvm nolist;
   50   modify orsalesTarget;
   51       ic create GROUP01 = check(where=(Product_Group ne "LSF")) message = "GROUP01 - Product
   51 ! Group cannot be LSF" msgtype=user;
   NOTE: Integrity constraint GROUP01 defined.
   52   quit;
   ```

   This signals that we have successfully created the integrity constraint .

4. The APPEND procedure is executed to attempt to append the user data set, ORSALES, to our target data set, ORSALESTARGET. All observations are appended except those in which Product_Group is equal to "LSF." Copies of any rejected observations are written to the audit SAS data set.

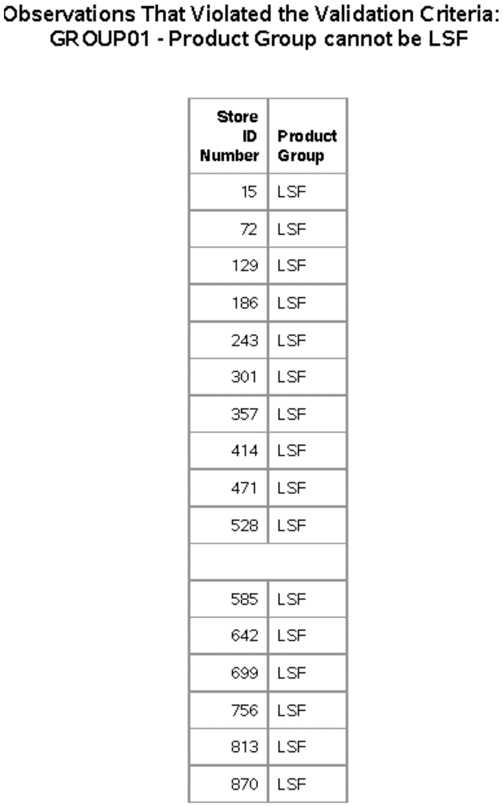
   The log from this step contains the following message:

   ```
   WARNING: GROUP01 - Product Group cannot be LSF  (Occurred 16 times.)
   NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
   NOTE: 896 observations added.
   ```

```
NOTE: The data set UDFTVM.ORSALESTARGET has 896 observations and 9 variables.
```

The warning denotes that 16 observations were rejected because they violated the integrity constraint we specified for Product_Group. That is backed up by the fact that only 896 of the 912 observations in ORSALES were allowed to be written to ORSALESTARGET.

5. We create a report of rejected observations from the audit SAS data set. Notice that we specify *TYPE=audit* on the DATA statement so SAS knows we want to input the audit data set. We print StoreID to uniquely identify the observations containing the error. We also print the value of Product_Group and _ATMESSAGE_ to underscore why each observation was flagged as errant. **Figure 2** has the report that was produced.



Observations That Violated the Validation Criteria:
GROUP01 - Product Group cannot be LSF

Store ID Number	Product Group
15	LSF
72	LSF
129	LSF
186	LSF
243	LSF
301	LSF
357	LSF
414	LSF
471	LSF
528	LSF
585	LSF
642	LSF
699	LSF
756	LSF
813	LSF
870	LSF

Figure 2 – Report of observations that violated the validation criteria for Product_Group.

The program above created an error report PDF file named **Observations That Violated the Validation Criteria for Product_Group.pdf**. We exercise **Step 5 of the UDFTVM** by attaching the report to an email that explains the issue we found with the file and send it to our client. That concludes our first complete run of the User Data File Template Validation Methodology on the ORSALES file they sent to us.

Our feedback on the first file generates additional discussions with our users about the characteristics of the data they are going to send to us for analysis. They determine that there are other rules they intend to put into place for filtering the data before submitting it. Those rules are for the following variables:

- **StoreID** – This is a primary key variable and must be unique in the data set.
- **Product_Line** – This is going to be used as an analysis variable so it cannot have missing values.
- **Product_Group** – They do not intend to send us any observations with the value of “LSF” because they do not want them in the analysis.
- **Quantity** – The analysis is for stores that have 2000 items or fewer.
- **Profit** – The analysis is for stores whose profits do not exceed $150,000.

We need to turn those five rules into five integrity constraints to validate the next ORSALES data set that our user submits to us. Each of those integrity constraints needs to be checked separately so that we can invalidate all observations that violate it. Instead of running the program above five times—each time with a different integrity constraint—we macrotize it and run the macro five times.

This is what the macrotized program looks like:

```
        /*********************************************************/
        /* BEGIN macro to check the validation criteria.        */
        /*********************************************************/
        /* This macro accepts three parameters:                 */
        /*                                                       */
        /* Keys - Variable(s) that uniquely identify an obs     */
        /*                                                       */
        /* Vars - Variable(s) whose values are being validated  */
        /*                                                       */
        /* Integ - Integrity Constraint to validate Vars        */
        /*********************************************************/
A       %MACRO VALIDATE(Keys=, Vars=, Integ=);


        /* Use the Template data set to create an empty SAS data set*/
        data udftvm.orsalesTarget;
1       set  udftvm.orsalesTemplate; /* <--This is the template SAS data set*/
        stop;
        run;

        /* Specify an audit trail for the empty SAS data set*/
        proc datasets library=udftvm nolist;
2               audit orsalesTarget;
                initiate;
                log admin_image=yes
                    before_image=yes
                    data_image=no
                    error_image=yes;
              run;
        quit;

        /* Specify an Integrity Constraint for the validation criteria*/
        proc datasets library=udftvm nolist;
3       modify orsalesTarget;
                &Integ
        quit;

        /* Append the user data set to the empty data set to test the validation criteria. */
        proc append base=udftvm.orsalesTarget
4                   data=udftvm.orsales; /* <--This is the user SAS data set*/
        run;
        quit;


        /* Put _ATMESSAGE_ in a macro variable for the report. */
B       data _null_;
        set udftvm.orsalesTarget(type=audit);
        call symput("ATMSG",strip(tranwrd(_ATMESSAGE_,"ERROR:","")));
        stop;
        run;


        /* Create a report of observations not matching validation criteria. */
        ods pdf file=" G:\My report directory\Observations That Violated the Validation Criteria for
        &ATMSG..pdf";

        Proc print noobs data=udftvm.orsalesTarget(type=audit) label blankline=10;
5       var &KEYS &VARS;
        title1 "Observations That Violated the Validation Criteria:";
        title2 "&ATMSG";
        title3 " For variable(s): &Vars";
        run;

        ods pdf close;

        /************************************************/
        /* END macro to check the validation criteria.  */
        /************************************************/
```

```
%MEND VALIDATE;
```

C

```
/* Test each one of the validation criterias. */
%Validate(Keys=StoreID,
          Vars=StoreID,
          Integ=ic create PRIMEKEY = primary key (StoreID)message = "PRIMEKEY - Duplicate StorIDs Are
                    Not Allowed" msgtype=user; );

%Validate(Keys=StoreID,
          Vars=Product_Line,
         Integ=ic create PRODLINE01 = not null(Product_Line) message = "PRODLINE01 - Product Line cannot
                   be missing" msgtype=user;);

%Validate(Keys=StoreID,
          Vars=Product_Group,
          Integ=ic create GROUP01 = check(where=(Product_Group ne "LSF")) message = "GROUP01 - Product
                    Group cannot be LSF" msgtype=user;);

%Validate(Keys=StoreID,
          Vars=Quantity,
          Integ=ic create QUANTITY01 = check(where=(quantity <= 2000)) message = "QUANTITY01 - Limit of
                    2000 items" msgtype=user;);

%Validate(Keys=StoreID,
          Vars=Profit,
         Integ=ic create PROFIT01 = check(where=(profit <= 150000)) message = "PROFIT01 - Profit cannot
                   exceed 150,000" msgtype=user;);
```

The SAS code in the numbered steps in the macrotized example is the same as that in the previous example with the exception that some of the steps have macro variables in them. The macro variables have been highlighted for ease of identification in the code. The three letters (A, B, and C) are specified to draw your attention to:

A. This is the beginning of the VALIDATE macro. The macro accepts three parameters:

- **Keys** – One or more variables that uniquely identify an observation. This is included in the report so that your users can hone in on the errant observations that you find using whatever uniquely identifies them in the data set they sent to you.
- **Vars** – Specifies the variable whose values are going to be checked via the integrity constraint. *Note that there may be instances where several variables are used in an integrity constraint, such as in the example of the REGPRODSUB constraint specified for the SHOES data set in the section of this paper titled: What Are Integrity Constraints. In such cases, you would specify all of the variables here.*
- **Integ** – This is the fully specified integrity constraint that will be used to validate the values of the Vars variable.

B. The DATA _NULL_ step is used to capture the value of _ATMESSAGE_ in a macro variable named ATMSG. This is used as both the name of the report PDF file we create in the next step as well as in Title2 of the actual report. Because SAS prepends the word “ERROR:” to our user message when creating _ATMESSAGE_, we remove this unnecessary wording when creating the ATMSG macro variable.

C. This is where the VALIDATE macro is invoked. You can see that we invoked the macro five times; once for each variable whose values we need to validate.

Unfortunately, the revises ORSALES file that our user sent to us was not clean. We got a hit on each of the integrity constraint tests that we tested:

```
WARNING: Duplicate StorIDs Are Not Allowed , 9 observations rejected.
NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
NOTE: 903 observations added.
NOTE: The data set UDFTVM.ORSALESTARGET has 903 observations and 9 variables.

WARNING: PRODLINE01 - Product Line cannot be missing  (Occurred 4 times.)
NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
NOTE: 908 observations added.
NOTE: The data set UDFTVM.ORSALESTARGET has 908 observations and 9 variables.
```

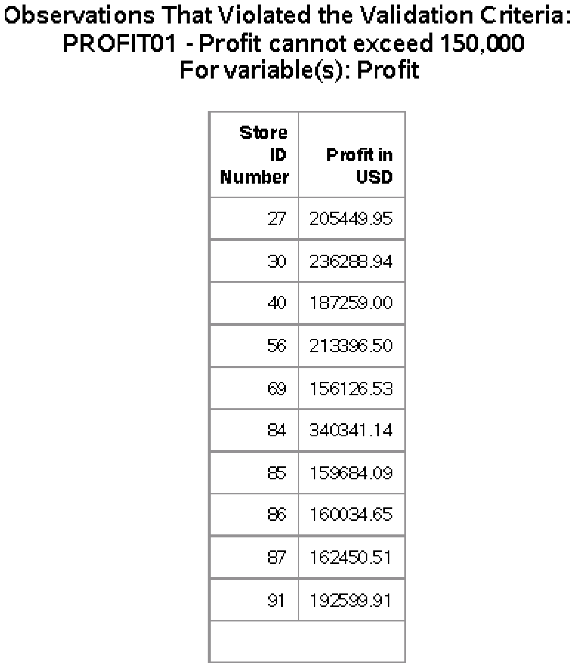
```
WARNING: GROUP01 - Product Group cannot be LSF  (Occurred 16 times.)
NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
NOTE: 896 observations added.
NOTE: The data set UDFTVM.ORSALESTARGET has 896 observations and 9 variables.

WARNING: QUANTITY01 - Limit of 2000 items  (Occurred 222 times.)
NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
NOTE: 690 observations added.
NOTE: The data set UDFTVM.ORSALESTARGET has 690 observations and 9 variables.

WARNING: PROFIT01 - Profit cannot exceed 150,000  (Occurred 129 times.)
NOTE: There were 912 observations read from the data set UDFTVM.ORSALES.
NOTE: 783 observations added.
NOTE: The data set UDFTVM.ORSALESTARGET has 783 observations and 9 variables.
```

So, once again we need to proceed to **Step 5 of the UDFTVM** and notify our users of the issues we found with their data. We do this by attaching the five report PDF's that were generated:

- Observations That Violated the Validation Criteria for PRIMEKEY - Duplicate StorIDs Are Not Allowed.pdf
- Observations That Violated the Validation Criteria for PRODLINE01 - Product Line cannot be missing.pdf
- Observations That Violated the Validation Criteria for GROUP01 - Product Group cannot be LSF.pdf
- Observations That Violated the Validation Criteria for QUANTITY01 - Limit of 2000 items.pdf
- Observations That Violated the Validation Criteria for PROFIT01 - Profit cannot exceed 150,000.pdf

…to another explanatory email and send them to our user.

Figure 3, below contains the first ten rows of the report that was produced for the violation of the integrity constraint for the PROFIT variable.



Observations That Violated the Validation Criteria:
PROFIT01 - Profit cannot exceed 150,000
For variable(s): Profit

Store ID Number	Profit in USD
27	205449.95
30	236288.94
40	187259.00
56	213396.50
69	156126.53
84	340341.14
85	159684.09
86	160034.65
87	162450.51
91	192599.91

Figure 3 – Report of observations that violated the validation criteria for Profit.

That concludes our second complete run of the User Data File Template Validation Methodology on the "revised" ORSALES file that was sent to us. Upon receiving the reports, our user will be able to revamp their system for creating the ORSALES file they are going to send to us for analysis. We would expect that the next ORSALES SAS data set they send would have valid values for StoreID, Product_Line, Product_Group, Quantity, and Profit. However, we will *trust but verify* by rerunning the next file we get through the UDFTVM, again starting at **Step 3**.

Step 5: Report the results to the entity that submitted the data.

Once you have run the incoming data set through the UDFTVM, you need to send feedback to the organization that sent it to you. If the data set was invalidated via **Step 3 of the UDFTVM** which checked the data set's structural metadata, send your user the .lst file for the PROC COMPARE. If the data set was discredited by **Step 4 of the UDFTVM** which used integrity constraints to check for valid values of variables, send the relevant reports. Armed with this information, your user should be able to fix the issues you identified and resubmit the updated SAS data set to you.

Of course, you should also let your users know when their data sets successfully passed the QC's. They will be pleased to know that their file has been validated through your rigorous system (the UDFTVM) and that you are ready to process the data in it.

Conclusion

This paper introduced the User Data File Template Validation Methodology (UDFTVM); a protocol that you can utilize to validate SAS data sets you receive from external sources. The UDFTVM consists of five distinct steps that lead you through a process of establishing a template SAS data set, and then comparing the incoming file to that template in order to determine its conformity to approved standards.

The five steps of the UDFTVM are:

1. Obtain the SAS data set's metadata and variable value ranges.

2. Create a template SAS data set.

3. Verify the data file's structural metadata with PROC COMPARE.

4. Check for valid values using integrity constraints and audit files.

5. Report the results to the entity that submitted the data.

Completing the five steps of the UDFTVM assures you that a user-supplied data set either conforms to established data attribute requirements or falls short of them. When files successfully pass UDFTVM testing, you can be confident that the analysis you perform on them will produce reliable results that are firmly based on the data that your client has sent to you.

Disclaimers

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

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.

References

Raithel, Michael A. 2017. Did You Know That? Essential Hacks for Clever SAS Programmers: Over 100 Essential Hacks to Make Your Programs Leaner, Cleaner, and More Competitive. Bethesda, Maryland: Michael A. Raithel
Available: http://www.amazon.com/Michael-A.-Raithel/e/B001K8GG90/ref=ntt_dp_epwbk_0

Raithel, M.A. (2017). PROC DATASETS; The Swiss Army Knife of SAS Procedures. *Proceedings of the SAS Global Forum 2017 Conference*.
Available: http://support.sas.com/resources/papers/proceedings17/0963-2017.pdf

SAS Institute Inc. 2015. *Base SAS® 9.4 Procedures Guide, Fifth Edition*. Cary, NC: SAS Institute Inc.
Available: http://support.sas.com/documentation/cdl/en/proc/68954/PDF/default/proc.pdf

SAS Institute Inc. 2015. *SAS® 9.4 Language Reference: Concepts, Fifth Edition*. Cary, NC: SAS Institute Inc.
Available: http://support.sas.com/documentation/cdl/en/lrcon/68089/PDF/default/lrcon.pdf

Acknowledgements

The author would like to thank former Westat Vice President and colleague Mike Rhoads for reviewing the paper and providing many sage suggestions for improving its focus and readability.

Contact Information

I would love to get your feedback on this paper; especially if you found it helpful. You can contact me at the following email address:
michaelraithel@westat.com