Paper 139-2018

The Doctor Ordered a Prescription…Not a Description: Driving Dynamic Data Governance Through Prescriptive Data Dictionaries That Automate Quality Control and Exception Reporting

Troy Martin Hughes

ABSTRACT

Data quality is a critical component of data governance and describes the accuracy, validity, completeness, and consistency of data. Data accuracy can be difficult to assess, as it requires a comparison of data to the real-world constructs being abstracted. But other characteristics of data quality can be readily assessed when provided a clear expectation of data elements, records, fields, tables, and their respective relationships. Data dictionaries represent a common method to enumerate these expectations and help answer the question *What should my data look like?* Too often, however, data dictionaries are conceptualized as static artifacts that only *describe* data. This text introduces dynamic data dictionaries that instead *prescribe* business rules against which SAS® data sets are automatically assessed, and from which dynamic, data-driven, color-coded exception reports are automatically generated. Dynamic data dictionaries—operationalized within Excel workbooks—allow data stewards to set and modify data standards without having to alter the underlying software that interprets and applies business rules. Moreover, this modularity—the extraction of the data model and business rules from the underlying code—flexibly facilitates reuse of this SAS macro-based solution to support endless data quality objectives.

INTRODUCTION

Data quality cannot be assessed in a vacuum—it requires standards against which data can be judged. At the most basic level, standards should communicate the data content (i.e., right variables or fields), format (i.e., right type of fields), and completeness (i.e., whether data are mandatory, or with what frequency they can be missing). This information is typically maintained within a data dictionary, which describes table- and field-level attributes. Some specifications, such as the requirement that a field contain unique values, must be assessed through inter-observation analysis. Other specifications must be assessed through inter-variable analysis, such as the requirement that variable B can be missing only when variable A is also missing. Thus, while some data dictionaries are merely enumerations of all variables with minimal descriptive information, more comprehensive data dictionaries may contain complex business rules and fuzzy logic that specify the conditions under which certain data requirements must be met.

While descriptive data dictionaries are useful resources for developers, analysts, and other stakeholders, data dictionaries can be difficult to maintain and keep current. Moreover, descriptive data dictionaries—even when complete and accurate—do nothing to enforce the business rules they describe. Prescriptive data dictionaries, rather, can enforce their rules either through data integrity constraints that don't allow invalid data to enter a data set, or through post hoc quality controls that identify, expunge, or delete invalid data after they have been ingested. This text demonstrates the latter method through a series of SAS macros that perform post hoc data quality control by interrogating a user-created data dictionary, extracting business rules, applying rules to user-specified data sets, and creating color-coded HTML exception reports. Prescriptive data dictionaries have two principal advantages over descriptive data dictionaries, in that as data are added or modified, the existing rules are automatically applied, and in that as business rules are modified within a data dictionary, those rules are automatically applied to existing data.

To demonstrate a real-world application of this solution, this text examines a personnel roster that might be maintained by a Human Resources department. A data dictionary is defined within Excel and queried by SAS to determine whether and how the personnel data set violates business rules contained within the data dictionary. The flexibility of this solution is demonstrated in successive examples that modify both the contents of the data dictionary and the contents and format of the personnel roster—all without ever requiring modification to the underlying SAS code. After each modification, the identical series of SAS macros is invoked, after which updated exception reports are automatically generated that both validate data quality and identify invalid data. These color-coded tactical reports identify specific types and instances of invalid data while strategic reports demonstrate validity trends for each variable.

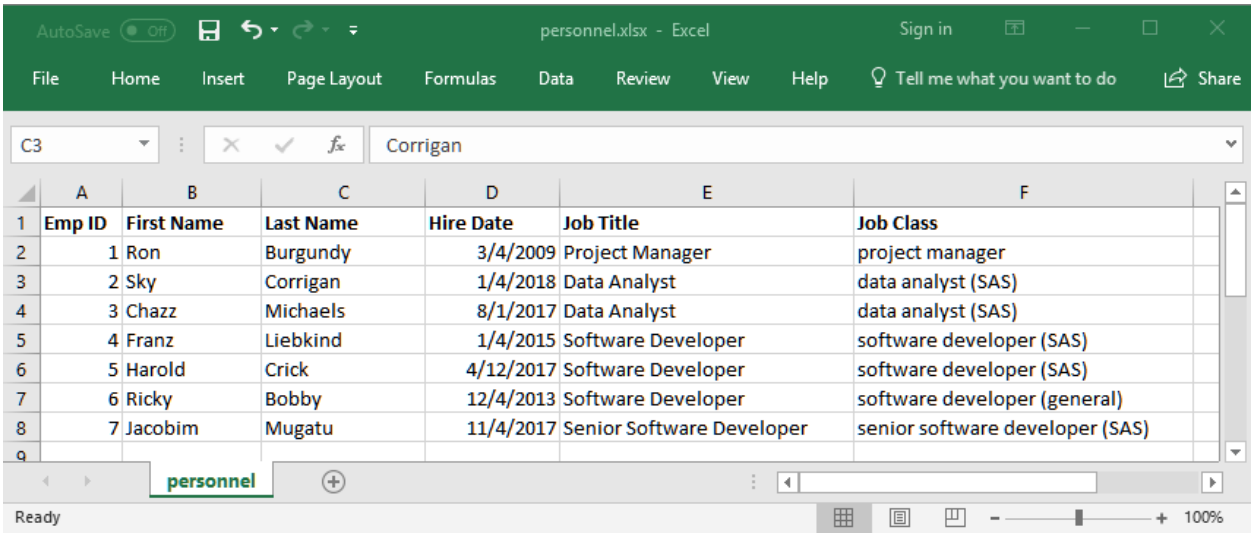
SETUP FOR EXAMPLES

These examples are demonstrated from the end-user perspective, thus focusing on the simplicity and flexibility of implementing the SAS solution rather than on the intricate innerworkings through which it is achieved. In other words, as any Cracker Barrel waitress will extol, “Please enjoy the sausage…but don’t ask how it’s made!” The following steps should be followed to run all examples within this text:

1. Download all code from Appendix A and save to a single SAS program file, Data_dictionary.sas.
2. The %INCLUDE statement should be modified to point to the folder in which the program was downloaded. In the following examples, the folder D:\sas\datadict\ is specified. Also note that the trailing backslash (or slash) is required.
3. The personnel roster, demonstrated in Table 1 and Figure 1, should be created in the same folder and saved (as an Excel workbook) as Personnel.xlsx.
4. The personnel data dictionary, demonstrated in Table 2 and Figures 2, 3, and 4, should be created in the same folder and saved (as an Excel workbook) as Dictionary_personnel.xlsx. Note that all three spreadsheets (i.e., tabs) of the workbook must be created as they appear in the figures.

THE PERSONNEL ROSTER

In this example, Ron Burgundy, fearless leader of a small SAS development and analytic team, maintains a roster of his staff within an Excel workbook (Personnel.xlsx). The spreadsheet is named “personnel” (case-sensitive) and is demonstrated in Figure 1.



	A	B	C	D	E	F
1	Emp ID	First Name	Last Name	Hire Date	Job Title	Job Class
2	1	Ron	Burgundy	3/4/2009	Project Manager	project manager
3	2	Sky	Corrigan	1/4/2018	Data Analyst	data analyst (SAS)
4	3	Chazz	Michaels	8/1/2017	Data Analyst	data analyst (SAS)
5	4	Franz	Liebkind	1/4/2015	Software Developer	software developer (SAS)
6	5	Harold	Crick	4/12/2017	Software Developer	software developer (SAS)
7	6	Ricky	Bobby	12/4/2013	Software Developer	software developer (general)
8	7	Jacobim	Mugatu	11/4/2017	Senior Software Developer	senior software developer (SAS)

Figure 1. Personnel Roster (Personnel.xlsx)

When downloaded to D:\sas\datadict, the following code imports this workbook into SAS:

```
%let location=D:\sas\datadict\;
%include "&location.data_dictionary.sas";

proc import datafile="&location.personnel.xlsx"
    out=personnel
    dbms=XLSX
    replace;
    sheet='personnel';
run;
```

The identical data set can be created in SAS using the following DATA step:

```
data personnel;
   infile datalines delimiter=',';
   length Emp_ID 8 First_Name $50 Last_Name $50 Hire_Date 8 Job_Title $50
       Job_Class $50;
   input Emp_ID First_Name $ Last_Name $ Hire_Date :mmddyy10. Job_Title $
       Job_Class $;
   format hire_date mmddyy10.;
   datalines;
1, Ron, Burgandy, 3/4/2009, Project Manager, project manager
2, Sky, Corrigan, 1/4/2018, Data Analyst, data analyst (SAS)
3, Chazz, Michaels, 8/1/2017, Data Analyst, data analyst (SAS)
4, Franz, Liebkind, 1/4/2015, Software Developer, software developer (SAS)
5, Harold, Crick, 4/12/2017, Software Developer, software developer (SAS)
6, Ricky, Bobby, 12/4/2013, Software Developer, software developer (general)
7, Jacobim, Mugatu, 11/4/2017, Senior Software Developer, senior software
developer (SAS)
;
```

The resultant SAS data set (Personnel) is demonstrated in Table 1.

Emp ID	First Name	Last Name	Hire Date	Job Title	Job Class
1	Ron	Burgundy	3/4/2009	Project Manager	project manager
2	Sky	Corrigan	1/4/2018	Data Analyst	data analyst (SAS)
3	Chazz	Michaels	8/1/2017	Data Analyst	data analyst (SAS)
4	Franz	Liebkind	1/4/2015	Software Developer	software developer (SAS)
5	Harold	Crick	4/12/2017	Software Developer	software developer (SAS)
6	Ricky	Bobby	12/4/2013	Software Developer	software developer (general)
7	Jacobim	Mugatu	11/4/2017	Senior Software Developer	senior software developer (SAS)

Table 1. Personnel Data Set

THE DATA DICTIONARY

While Ron has meticulously maintained his team's roster, at some point, another employee or Human Resources might take over this responsibility. Because Ron uses his personnel roster in automated SAS processes, it's critical that the established data format be maintained and not mangled during a transfer of responsibility. One of the easiest ways to provide a roadmap for data quality is to create a data dictionary that specifies the requirements for each variable. Figure 2 represents a data dictionary that specifies attributes for the Personnel data set. At this point, the data dictionary is only descriptive because there's no indication that it is being used to enforce the business rules that it describes.

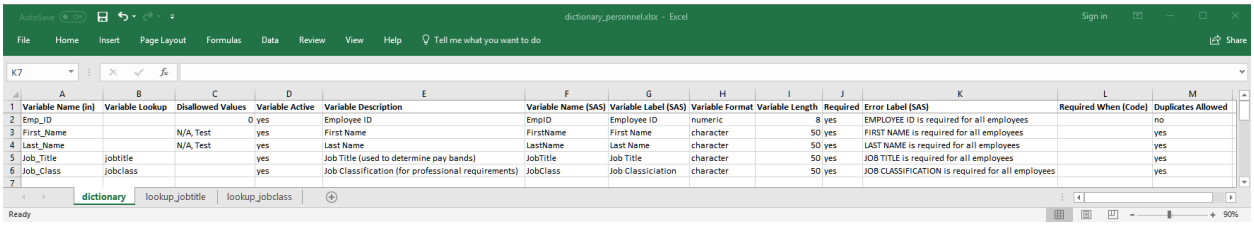


Figure 2. Data Dictionary (Dictionary_personnel.xlsx, "dictionary" spreadsheet) for Personnel Roster

Figure 3 demonstrates the second tab of the workbook (“lookup_jobtitle”), which includes the list of valid values for the Job_title variable. All values are listed in the Value column while the Description column can be optionally used to provide additional context for each value (not demonstrated). The spreadsheet title is case-sensitive and is derived from appending the value in the Variable Lookup column (cell A5) of the spreadsheet (“jobtitle”) to “lookup_”.

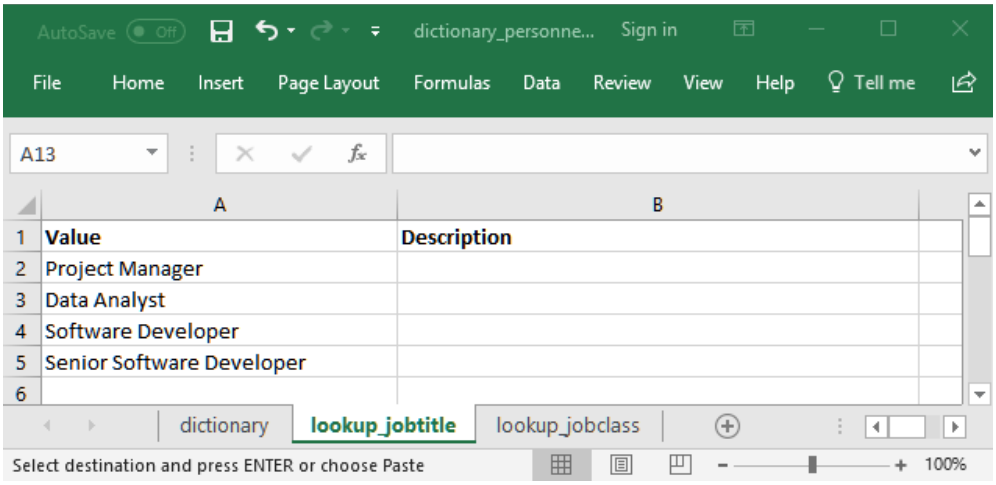


Figure 3. Data Dictionary (Dictionary_personnel.xlsx, “lookup_jobtitle” spreadsheet) for Personnel Roster

Figure 4 demonstrates the third tab of the workbook (“lookup_jobclass”), which includes the list of valid values for the Job_class variable. All values are included in the Value column while the Description column can be optionally used to provide additional context for each value. The spreadsheet title is case-sensitive and is derived from appending the value in the Variable Lookup column (cell A6) of the spreadsheet (“jobclass”) to “lookup_”.

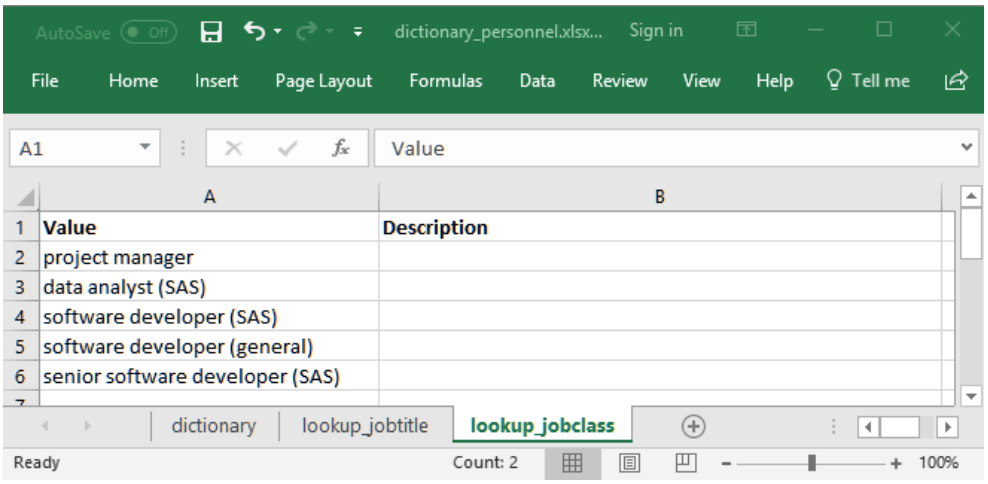


Figure 4. Data Dictionary (Dictionary_personnel.xlsx, “lookup_jobtitle” spreadsheet) for Personnel Roster

Table 2 shows the Dictionary spreadsheet of the Dictionary_personnel workbook and can be used to cut and paste these data into the Excel workbook (Dictionary_personnel.xlsx) that must be created.

Variable Name (in)	Variable Lookup	Disallowed Values	Variable Active	Variable Description	Variable Name (SAS)	Variable Label (SAS)	Variable Format	Variable Length	Required	Error Label (SAS)	Required When (Code)	Duplicates Allowed
Emp_ID		0	yes	Employee ID	EmpID	Employee ID	numeric	8	yes	EMPLOYEE ID is required for all employees		no

First_Name		N/A, Test	yes	First Name	FirstName	First Name	character	50	yes	FIRST NAME is required for all employees		yes
Last_Name		N/A, Test	yes	Last Name	LastName	Last Name	character	50	yes	LAST NAME is required for all employees		yes
Job_Title	jobtitle		yes	Job Title (used to determine pay bands)	JobTitle	Job Title	character	50	yes	JOB TITLE is required for all employees		yes
Job_Class	jobclass		yes	Job Classification (for professional requirements)	JobClass	Job Classification	character	50	yes	JOB CLASSIFICATION is required for all employees		yes

Table 2. Data Dictionary (Dictionary_personnel.xlsx, "dictionary" spreadsheet) for Personnel Roster

Figure 2 and Table 2 both demonstrate the format required by data dictionaries to ensure that they are parsed correctly by the underlying SAS macros. Column names cannot be modified (because they are ingested into SAS and renamed as SAS variables) but all other rows should be modified to reflect the variables within the associated data set. While each of the following columns must appear in the Dictionary spreadsheet, only those columns indicated as "required" must have values for each row (i.e., variable):

1. **Variable Name (in)** – (REQUIRED) This column identifies the variable name of the SAS data set being examined—Personnel, in this example. Note that although the Personnel workbook was ingested, the macros run against the resultant SAS data set, generated from the previous IMPORT procedure. Because this column is converted into a SAS variable name, it must adhere to SAS variable-naming conventions.

2. **Variable Lookup** – (OPTIONAL) This column identifies categorial variables (of type character or numeric) for which the set of valid values is known. For example, cell B5 ("jobtitle") indicates that the Job_Title variable has a known set of values that will be listed on a separate spreadsheet (i.e., tab), "lookup_jobtitle". Thus, any variable having a value in the Variable Lookup column must have a corresponding case-sensitive spreadsheet that lists the possible values. If the Variable Lookup cell is empty, this indicates that the variable is not categorical or that the set of valid values is unknown or undescribed. For example, the variables First_Name and Last_Name have no associated lookup spreadsheets because no comprehensive list of names exists.

3. **Disallowed Values** – (OPTIONAL) This comma-delimited list includes values that are prohibited. For example, the employee ID (Emp_ID) cannot be "0" and the First_Name cannot be either "N/A" or "Test". Quotation marks are not required (for either character or numeric data); however, this rudimentary feature will fail if quotations, commas, or some other special characters are included.

4. **Variable Active** – (REQUIRED) The value should be either "yes" or "no", indicating whether the variable should be included in the quality control review. This feature is beneficial, for example, while incrementally constructing a data dictionary, because only completed rows can be selectively added. This is also beneficial when applying a more comprehensive data dictionary to a data set that lacks some of the variables.

5. **Variable Description** – (OPTIONAL) This value can be used to specify information about the variable. For example, in cells E5 and E6, Job_Title and Job_Class (two similarly sounding entities) are differentiated.

6. **Variable Name (SAS)** – (REQUIRED) This column contains the names of the SAS variables that are created internally, whereas the Variable Name (in) column is used to specify the SAS variable names on the original data set being analyzed. This is useful because the ingested variable name can be changed (e.g., if an external data source is modified) without having to change the internal SAS variable name. In practice, the two columns (Variable Name (in) and Variable Name (SAS)) will typically have identical values for each

row. Because this column is converted into a SAS variable name, it must adhere to SAS variable-naming conventions.

7. **Variable Label (SAS)** – (OPTIONAL) This column represents the SAS label applied to the Variable Name (SAS) column, and as such is restricted to SAS label-naming conventions.

8. **Variable Format** – (REQUIRED) The value should be either “numeric” or “character”, representing the type of variable being analyzed. Only unformatted values are compared against the data dictionary, so in this version, SAS dates and other complex variable formats are unrecognized and cannot be analyzed.

9. **Variable Length** – (REQUIRED) The value should correspond to the SAS LENGTH statement, as applied to the associated variable.

10. **Required** – (REQUIRED) The value should be either “yes” or “no”, representing whether the variable in the data set must have a value. In this example, all variables must be complete. Because of the complexity of (and fuzzy logic business rules related to) assessing data completeness, a separate error label (Error Label (SAS) column) can be used to describe the rules associated with assessing completeness. Moreover, the Required When (Code) column is used to specify under what conditions a variable must be complete.

11. **Error Label (SAS)** – (OPTIONAL) The value describes the error code and is displayed in the HTML exception reporting. For example, because Employee ID is a required variable, if it is missing, hovering over a blank cell (for the Emp_ID variable) in the HTML report will display the text “EMPLOYEE ID is required for all employees”. This column is only required when the Required column indicates “yes”.

12. **Required When (Code)** – (OPTIONAL) The value includes a conditional logic statement that optionally specifies the conditions under which a variable is required. For example, if JobClass were only required for Employee IDs greater than 5, then “EmpID > 5” [without quotations] could be entered in cell L6 to limit the scope of the requirement. Thus, the JobClass variable would be allowed to be missing only for IDs one through five.

13. **Duplicates Allowed** – (REQUIRED) The value should be either “yes” or “no”, representing whether the variable can have duplicates within the data set. This can be useful when a primary key is defined; however, as each row of the data dictionary workbook is treated separately, no functionality exists to define composite keys in which the combination of two or more variables must be unique across observations.

No internal validation for the data dictionary exists, so it is essential that entries be made per the preceding guidance. A more robust solution would ensure that aberrant values were not entered or might validate each row against several criteria, excluding rows that did not meet all criteria.

RUNNING THE EXAMPLE

After following the steps in the “Setup for Examples” section, the following code imports the personnel roster (from Excel), imports the data dictionary (from Excel), creates an HTML version of the data dictionary, applies the data dictionary business rules to the personnel roster, and finally creates an exception report:

```
%let location=/folders/myfolders/datadriven/;
%include "&location.data_dictionary.sas";

* import the personnel roster into SAS;
proc import datafile="&location.personnel.xlsx"
    out=personnel
    dbms=XLSX
    replace;
    sheet='personnel';
run;

* import and process the data dictionary;
```

```
%import_data_dic(dictspread=&location.dictionary_personnel.xlsx);

* create HTML version of data dictionary;
%make_html_dic(htmlfile=&location.personnel_data_dictionary.html,
   title=Personnel Data Dictionary);

* apply business rules from data dictionary to data set;
%apply_business_rules(dsn=personnel);

%data_dic_report(htmlrptpath=&location,
   htmlrpt=personnel_data_quality.html,
   title=Personnel Exception Report,
   display=all);
```

When the previous %INCLUDE statement is run, the following FORMAT procedures execute from within the Data_dictionary.sas program file:

```
proc format;
   value expected
   1='Required'
   2='Optional'
   3='Disallowed';
run;

proc format;
   value observed
   1='Valid'
   2='Missing'
   3='Invalid'
   4='Duplicate';
run;
```

The formats are used to convey what is both expected of and observed from the data, and every cell being analyzed within the data set receives both an Expected and Observed Value. For example, “exp_EmpID=1” indicates that the Employee ID variable is expected for a specific observation while “exp_EmpID=2” indicates that the variable can be missing. However, whether the variable is missing (or valid, invalid, or duplicate) is contained within the exp_EmpID variable. The Expected and Observed variables are created automatically within the IMPORT_DATA_DIC macro.

The IMPORT_DATA_DIC macro ingests the data dictionary and dynamically writes business rules (encapsulated in the global macro variable &RULESLIST) that are injected into SAS code in the following step. In this example, &RULESLIST has the following value after this macro is invoked:

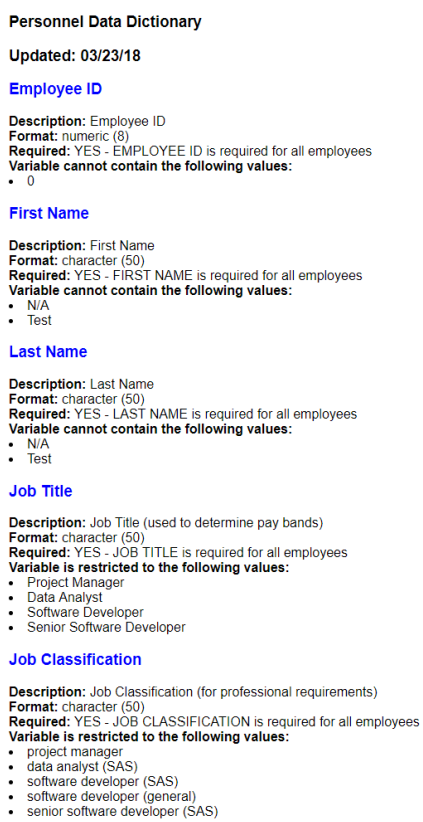
```
exp_EmpID=1;if missing(EmpID) then obs_EmpID=2;else if EmpID in(0) then
obs_EmpID=3; else obs_EmpID=1;if
 exp_EmpID=1 then obsreq_EmpID= obs_EmpID; else
obsreq_EmpID=.;exp_FirstName=1;if missing(FirstName) then obs_FirstName=2;else
if
 strip(lowcase(FirstName)) in("n/a","test") then obs_FirstName=3; else
obs_FirstName=1;if exp_FirstName=1 then obsreq_FirstName=
 obs_FirstName; else obsreq_FirstName=.;exp_LastName=1;if missing(LastName)
then obs_LastName=2;else if strip(lowcase(LastName))
 in("n/a","test") then obs_LastName=3; else obs_LastName=1;if exp_LastName=1
then obsreq_LastName= obs_LastName; else
 obsreq_LastName=.;exp_JobTitle=1;if missing(JobTitle) then obs_JobTitle=2;else
if strip(JobTitle) in ("Project Manager","Data
```

```
 Analyst","Software Developer","Senior Software Developer") then
obs_JobTitle=1; else obs_JobTitle=3;if exp_JobTitle=1 then
 obsreq_JobTitle= obs_JobTitle; else obsreq_JobTitle=.;exp_JobClass=1;if
missing(JobClass) then obs_JobClass=2;else if
 strip(JobClass) in ("project manager","data analyst (SAS)","software developer
(SAS)","software developer (general)","senior
 software developer (SAS)") then obs_JobClass=1; else obs_JobClass=3;if
exp_JobClass=1 then obsreq_JobClass= obs_JobClass; else
 obsreq_JobClass=.;
```

Inserting appropriate line breaks in the first few lines yields a more reader-friendly version of the dynamic code, indicating the logic that is followed to assign the Expected and Observed values for the first variabls:

```
exp_EmpID=1;
if missing(EmpID) then obs_EmpID=2;
else if EmpID in(0) then obs_EmpID=3;
else obs_EmpID=1;
if exp_EmpID=1 then obsreq_EmpID= obs_EmpID;
else obsreq_EmpID=.;
```

The MAKE_HTML_DIC macro creates a simple HTML version of the data dictionary. Distributing or posting the HTML version ensures that the Excel workbook—the control table producing the dynamism—is kept safe and not accessible to those who only need to view the data dictionary. This more straightforward version of the data dictionary is also useful to stakeholders responsible for remediating errors identified in an exception report. For example, the report makes it easy to identify what the acceptable values of the JobTitle variable are, should an invalid (or missing) value be detected. The previous MAKE_HTML_DIC macro produces the report (personnel_data_dictionary.html) demonstrated in Figure 5.



Personnel Data Dictionary

Updated: 03/23/18

Employee ID

Description: Employee ID
Format: numeric (8)
Required: YES - EMPLOYEE ID is required for all employees
Variable cannot contain the following values:
- 0

First Name

Description: First Name
Format: character (50)
Required: YES - FIRST NAME is required for all employees
Variable cannot contain the following values:
- N/A
- Test

Last Name

Description: Last Name
Format: character (50)
Required: YES - LAST NAME is required for all employees
Variable cannot contain the following values:
- N/A
- Test

Job Title

Description: Job Title (used to determine pay bands)
Format: character (50)
Required: YES - JOB TITLE is required for all employees
Variable is restricted to the following values:
- Project Manager
- Data Analyst
- Software Developer
- Senior Software Developer

Job Classification

Description: Job Classification (for professional requirements)
Format: character (50)
Required: YES - JOB CLASSIFICATION is required for all employees
Variable is restricted to the following values:
- project manager
- data analyst (SAS)
- software developer (SAS)
- software developer (general)
- senior software developer (SAS)

Figure 5. Data Dictionary HTML Report

The APPLY_BUSINESS_RULES macro applies the business rules macro variable (&RULESLIST) created in the IMPORT_DATA_DIC macro to the data set being inspected. If the values of a variable must be unique, the data set is sorted by that variable, so this step can be time-consuming depending on the number of unique variables analyzed and the size of the data set.

The DATA_DIC_REPORT macro creates the color-coded exception report that identifies any data quality issues. In addition to color-coding, the report also includes HTML popup windows that appear when a user hovers over a color-coded cell. The previous code produces the HTML exception report (Personnel_data_quality.html) demonstrated in Figure 6.

Personnel Exception Report

Updated: 03/24/18

No errors detected

Figure 6. Exception Report Showing No Errors

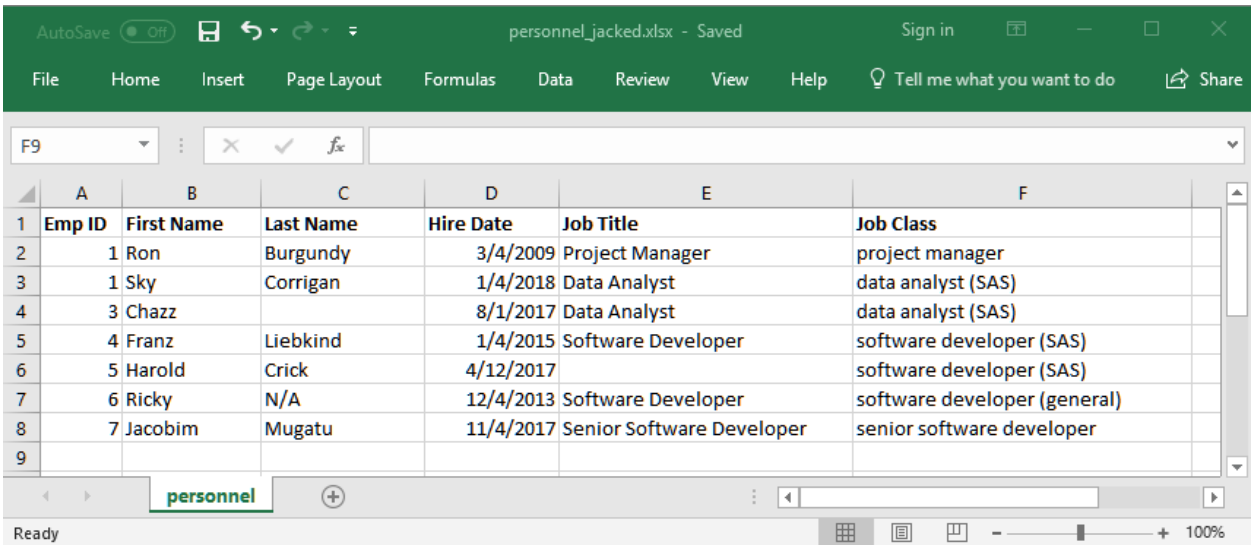
The report indicates that no errors were detected, as defined within the data dictionary and as observed within the personnel roster. In the next section, the personnel roster is modified (erroneously) and errors are discovered.

EXCEPTION REPORTING

To demonstrate how exceptions are detected and reported, five modifications are made in the Personnel spreadsheet, which is renamed Personnel_jacked. These changes include:

- Cell A3 – The Employee ID “1” is repeated twice.
- Cell C4 – The Last Name “Michaels” is deleted.
- Cell E6 – The Job Title “Software Developer is deleted.
- Cell C7 – The Last Name “Bobby” was replaced with the invalid value “N/A”.
- Cell F8 – The Job Class “senior software developer (SAS)” is changed to “senior software developer”.

These changes are reflected in the updated spreadsheet, demonstrated in Figure 6.



	A	B	C	D	E	F
1	Emp ID	First Name	Last Name	Hire Date	Job Title	Job Class
2	1	Ron	Burgundy	3/4/2009	Project Manager	project manager
3	1	Sky	Corrigan	1/4/2018	Data Analyst	data analyst (SAS)
4	3	Chazz		8/1/2017	Data Analyst	data analyst (SAS)
5	4	Franz	Liebkind	1/4/2015	Software Developer	software developer (SAS)
6	5	Harold	Crick	4/12/2017		software developer (SAS)
7	6	Ricky	N/A	12/4/2013	Software Developer	software developer (general)
8	7	Jacobim	Mugatu	11/4/2017	Senior Software Developer	senior software developer
9						

Figure 6. Personnel Spreadsheet (Personnel_jacked.xlsx) with Injected Errors

The identical code can be run to demonstrate the revised exception report, albeit replacing the spreadsheet Personnel.xlsx with Personnel_jacked.xlsx in the IMPORT procedure:

```
proc import datafile="&location.personnel_jacked.xlsx"
    out=personnel
    dbms=XLSX
    replace;
    sheet='personnel';
run;

%import_data_dic(dictspread=&location.dictionary_personnel.xlsx);

%make_html_dic(htmlfile=&location.personnel_data_dictionary.html,
    title=Personnel Data Dictionary);

%apply_business_rules(dsn=personnel);

%data_dic_report(htmlrptpath=&location,
    htmlrpt=personnel_data_quality.html,
    title=Personnel Exception Report,
    display=all);
```

Refreshing the HTML exception report, demonstrated in Figure 7, now identifies each of the errors that was just introduced, without the necessity to modify any of the underlying code.

#	Employee ID	First Name	Last Name	Job Title	Job Classification
1	1	Ron	Burgundy	Project Manager	project manager
2	1	Sky	Corrigan	Data Analyst	data analyst (SAS)
3	3	Chazz		Data Analyst	data analyst (SAS)
4	4	Franz	Liebkind	Software Developer	software developer (SAS)
5	5	Harold	Crick		software developer (SAS)
6	6	Ricky	N/A	Software Developer	software developer (general)
7	7	Jacobim	Mugatu	Senior Software Developer	senior software developer

Figure 7. Personnel Exception Report (Personnel_jacked.xlsx) with Injected Errors

The report correctly identifies each of the injected errors with the following color-coding:

- YELLOW cells contain duplicate values that should be unique.
- ORANGE cells contain invalid values (either those explicitly prohibited or those not included in an associated lookup spreadsheet).
- RED cells are required but missing.

Hovering over any color-coded cell (in the HTML report only) produces a popup that displays an associated error message. For example, in Figure 7, the following popup messages are displayed:

- Line 1 / Employee ID – EmpID cannot be duplicate
- Line 2 / Employee ID – EmpID cannot be duplicate

- Line 3 / Last Name – LASTNAME always required
- Line 5 / Job Title – JOBTITLE always required
- Line 6 / Last Name – LASTNAME value disallowed
- Line 7 / Job Classification – JOBCLASS not in lookup table

As the Personnel spreadsheet continues to be modified or updated with new data, the underlying code will continue to produce dynamic macro code that assesses the data quality through updated exception reports.

CONCLUSION

Data dictionaries are often construed as static documents—artifacts that must be arduously updated independent of data models or data quality controls. While these *descriptive* data dictionaries do exist in some environments and provide some value to stakeholders, *prescriptive* data dictionaries automate data quality functions by extracting business rules and applying these dynamically to data sets. As prescriptive data dictionaries are modified incrementally over time, and as data sets morph and grow, these data dictionaries can comfortably flex with to ensure that data sets are always being assessed against current quality controls. The dynamic, data-driven solution within this text demonstrates a modular design in which endless data dictionaries can be constructed, and through which color-coded exception reports are generated to alert stakeholders to data quality issues.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. DATA DICTIONARY MACROS (DATA_DICTIONARY.SAS)

```
* formats required to calculated expected and observed metrics for data
governance;
proc format;
   value expected
   1='Required'
   2='Optional'
   3='Disallowed';
run;

proc format;
   value observed
   1='Valid'
   2='Missing'
   3='Invalid'
   4='Duplicate';
run;

*-;
*---;
*----- IMPORT DATA DICTIONARY BUSINESS RULES;
*---;
*-;

%macro import_data_dic(dictspread = /* data dictionary spreadsheet
   path and file name */);
%global varlist varcnt oldvarlist lookuplist lookupformatlist headerlength
   headerformat headerlabel ruleslist renamelist noduplist;
%let varlist=;
%let oldvarlist=;
%let lookuplist=;
%let lookupformatlist=;
%let headerlength=;
%let headerformat=;
%let headerlabel=;
%let ruleslist=;
%let renamelist=;
%let noduplist=;
%local i var lookupformat;
proc import datafile="&dictspread"
   out=dict1
   dbms=xlsx
   replace;
   getnames=yes;
   sheet="dictionary";
run;
* create variable lists for LENGTH and FORMAT statements;
data dict2 (drop=varlist oldvarlist lookuplist lookupformatlist varcnt);
   length varIN $30 lookup $50 blacklist $1000 status $15 desc $200 varSAS $30
      labelSAS $50 form $10 len 8 errSAS $200 req $3 reqCode $2000 dups $3
      varlist $10000 oldvarlist $10000 lookuplist $10000
      lookupformatlist $10000 varcnt 8 noduplist $10000;
   format varIN $30. lookup $50. blacklist $1000. status $15. desc $200.
      varSAS $30. labelSAS $50. form $10. len 8. errSAS $200. req $3.
```

```
      reqCode $500. dups $3. noduplist $10000.;
   set dict1 (rename=(variable_name__in_=varIN variable_lookup=lookup
      disallowed_values=blacklist variable_active=status
         variable_description=desc variable_name__sas_=varSAS
         variable_label__sas_=labelSAS variable_format=form
         variable_length=len error_label__sas_=errSAS
         required=req required_when__code_=reqCode
         duplicates_allowed=dups)) end=eof;
   if _n_=1 then do;
      varlist='';
      oldvarlist='';
      lookuplist='';
      lookupformatlist='';
      noduplist='';
      varcnt=0;
      end;
   if lengthn(strip(lookup))=0 then lookup='N/A';
   if strip(lowcase(status))='yes' then do; * only process active variables;
      varcnt=varcnt+1;
      varlist=catx(' ',varlist,strip(varSAS));
         * space-delimited (used in KEEP statements);
      oldvarlist=catx(' ',oldvarlist,strip(varIN));
      lookuplist=catx('*',lookuplist,strip(lookup));
      lookupformatlist=catx('*',lookupformatlist,strip(lowcase(form)));
      if strip(lowcase(dups))='no' then noduplist=
         catx(' ',noduplist,strip(varSAS));
      end;
   if eof then do;
      call symput('varlist',strip(varlist));
      call symput('oldvarlist',strip(oldvarlist));
      call symput('lookuplist',strip(lookuplist));
      call symput('lookupformatlist',strip(lookupformatlist));
      call symput('varcnt',strip(put(varcnt,8.)));
      call symput('noduplist',strip(noduplist));
      end;
   retain varlist oldvarlist lookuplist lookupformatlist varcnt noduplist;
run;

* read all lookup tables into macro variable lists;
%do i=1 %to &varcnt;
   %if "%scan(&lookuplist,&i,*)"^="N/A" %then %do;
      proc import datafile="&dictspread"
         out=t1
         dbms=xlsx
         replace;
         sheet="lookup_%upcase(%scan(&lookuplist,&i,*))";
      run;
      %let lookupformat=%scan(&lookupformatlist,&i,*);
      %global lookup_%scan(&varlist,&i,,S) lookup2_%scan(&varlist,&i,,S);
      %let lookup_%scan(&varlist,&i,,S)=;
      %let lookup2_%scan(&varlist,&i,,S)=;
      data _null_;
         set t1 end=eof;
         length list $10000 list2 $10000;
         if _n_=1 then do;
```

```
            list='';
            list2='';
            end;
         %if "&lookupformat"="character" %then %do;
            list=catx(',',list,'"'||strip(value))||'"';
            %end;
         %else %do;
            list=catx(',',list,strip(value));
            %end;
         list2=catx('^',list2,strip(value));
            * for data dictionary HTML report;
         if eof then do;
            call symputx("lookup_%scan(&varlist,&i,,S)",strip(list));
            call symputx("lookup2_%scan(&varlist,&i,,S)",strip(list2));
            end;
         retain list list2;
      run;
      %end;
   %end;

* dynamically write code to detect invalid data;
data _null_ (drop=headerlength headerformat headerlabel ruleslist i
      blacklist_quoted);
   set dict2 end=eof;
   length headerlength $10000 headerformat $10000 headerlabel $10000
      ruleslist $30000 renamelist $10000 i 3 blacklist_quoted $10000;
   if _n_=1 then do;
      headerlength='length ';
      headerformat='format ';
      headerlabel='label';
      ruleslist='';
      renamelist='';
      blacklist_quoted='';
      end;
   if ^missing(blacklist) then do;
      i=1;
      do while(length(scan(blacklist,i,','))>1);
         blacklist_quoted=catx(',',blacklist_quoted,
            quote(strip(lowcase(scan(blacklist,i,',')))));
         i=i+1;
         end;
      end;
   if strip(lowcase(status))='yes' then do;
      * generate dynamic LENGTH, FORMAT, and LABEL statements;
      headerlength=strip(headerlength) || ' ' || strip(varSAS) || ' ' ||
         ifc(lowcase(form)='character',' $',' ')
         || strip(put(len,8.));
      headerlength=strip(headerlength) || ' exp_' || strip(varSAS) || ' 3';
      headerlength=strip(headerlength) || ' obs_' || strip(varSAS) || ' 3';
      headerlength=strip(headerlength) || ' obsreq_' || strip(varSAS) || ' 3';
      * if not character, then date, if not date, then defaults to numeric;
      headerformat=strip(headerformat) || ' ' || strip(varSAS) ||
         ifc(lowcase(form)='character',' $' || strip(put(len,8.)),
         ifc(lowcase(form)='date',' mmddyy10',' ' || strip(put(len,8.))))
         || '.';
```

```
headerformat=strip(headerformat) || ' exp_' || strip(varSAS)
   || ' expected.';
headerformat=strip(headerformat) || ' obs_' || strip(varSAS)
   || ' observed.';
headerformat=strip(headerformat) || ' obsreq_' || strip(varSAS)
   || ' observed.';
headerlabel=strip(headerlabel) || ' ' || strip(varSAS) || '="'
   || strip(labelSAS) || '"';
headerlabel=strip(headerlabel) || ' exp_' || strip(varSAS) || '="'
   || strip(upcase(varSAS)) ||
   ifc(strip(lowcase(req))='no',' not required"',
   ifc(missing(reqCode),' always required"',' ' || strip(errSAS)
   || '"'));
headerlabel=strip(headerlabel) || ' obs_' || strip(varSAS) || '="'
   || upcase(strip(varSAS)) ||
   ifc(^missing(lookup) and strip(lowcase(lookup))^='n/a',
      ' not in lookup table"',
   ifc(strip(lowcase(lookup))='n/a' and ^missing(blacklist),
      ' value disallowed"',' does not restrict values"'));
headerlabel=strip(headerlabel) || ' obsreq_' || strip(varSAS) || '="'
   || upcase(strip(varSAS)) || ' Status When Required"';
renamelist=strip(renamelist) || ' ' || strip(varIN) || '='
   || strip(varSAS);
* EXPECTED busines rules;
if strip(lowcase(req))='no' then ruleslist=strip(ruleslist) || 'exp_'
   || strip(varSAS) || '=2;';
else if strip(lowcase(req))='yes' then do;
   if missing(reqCode) then ruleslist=strip(ruleslist) || 'exp_'
      || strip(varSAS) || '=1;';
   else ruleslist=strip(ruleslist) || 'if ' || strip(reqCode)
      || ' then exp_' || strip(varSAS) || '=1; else exp_'
      || strip(varSAS) || '=2;';
   end;
* OBSERVED business rules;
ruleslist=strip(ruleslist) || 'if missing(' || strip(varSAS) || ')
   then obs_' || strip(varSAS) || '=2;';
if strip(upcase(lookup))='N/A' or missing(lookup) then do;
   if strip(lowcase(form))='character' then do;
      if ^missing(blacklist) then ruleslist=strip(ruleslist) ||
            'else if strip(lowcase(' || strip(varSAS) || ')) in(' ||
            strip(lowcase(blacklist_quoted)) || ') then obs_' ||
            strip(varSAS) || '=3; else obs_' || strip(varSAS)
            || '=1;';
      else ruleslist=strip(ruleslist) || 'else obs_' || strip(varSAS)
            || '=1;';
      end;
   else do;
      if ^missing(blacklist) then ruleslist=strip(ruleslist)
            || 'else if ' || strip(varSAS) || ' in(' ||
            strip(blacklist) || ') then obs_' || strip(varSAS)
            || '=3; else obs_' || strip(varSAS) || '=1;';
      else ruleslist=strip(ruleslist) || 'else obs_' || strip(varSAS)
            || '=1;';
      end;
   end;
```

```
      else do;
         if strip(lowcase(form))='character' then do;
            ruleslist=strip(ruleslist) || 'else if strip(' || strip(varSAS)
                  || ') in (' || resolve('&lookup_' || strip(varSAS)) || ')
                  then obs_' || strip(varSAS) || '=1; else obs_'
                  || strip(varSAS) || '=3;';
            end;
         else do;
            ruleslist=strip(ruleslist) || 'else if ' || strip(varSAS)
                  || ' in (' || resolve('&lookup_' || strip(varSAS)) || ')
                  then obs_' || strip(varSAS) || '=1; else obs_'
                  || strip(varSAS) || '=3;';
            end;
         end;
      ruleslist=strip(ruleslist) || 'if exp_' || strip(varSAS) || '=1
         then obsreq_' || strip(varSAS) || '= obs_' || strip(varSAS)
         || '; else obsreq_' || strip(varSAS) || '=.;';
         * set to missing so it will not be counted in summary stats later;
      end;
   if eof then do;
      call symput('headerlength',strip(headerlength));
      call symput('headerformat',strip(headerformat));
      call symput('headerlabel',strip(headerlabel));
      call symput('ruleslist',strip(ruleslist));
      call symput('renamelist',strip(renamelist));
      end;
   retain headerlength headerformat headerlabel ruleslist renamelist;
run;
%let headerlength=&headerlength%str(;);
%let headerformat=&headerformat%str(;);
%let headerlabel=&headerlabel%str(;);
%mend;

*-;
*---;
*----- GENERATE HTML VERSION OF DATA DICTIONARY;
*---;
*-;

%macro make_html_dic(htmlfile = /* path and file name of HTML
      data dictionary created */,
   title = /* data dictionary title */);
%local today;
%let today=%sysfunc(date(),mmddyy8.);
data _null_;
   file "&htmlfile";
   set dict2 end=eof;
   where strip(lowcase(status))='yes';
   length tempvar $1000 tempvarsmall $100 i 3 lenchar $8;
   if _n_=1 then do;
      put '<html><header>';
      put '</header>';
      put '<body>';
      put '<font face=Arial>';
      put "<h3>&title<br>";
```

```
      put "<h3>Updated: &today<p>";
      end;
   put '<h3><font color=blue>' labelSAS '</font><br></h3>';
   put '<b>Description:</b> ' desc '<br>';
   lenchar='('||strip(put(len,8.))||')';
   if lowcase(form)='character' then put '<b>Format:</b> character '
      lenchar '<br>';
   else if lowcase(form)='numeric' then put '<b>Format:</b> numeric '
      lenchar '<br>';
   else if lowcase(form)='date' then put '<b>Format:</b> date<br>';
   if lowcase(req)='yes' then put '<b>Required:</b> YES - ' errSAS '<br>';
   else if lowcase(req)='no' then put '<b>Required:</b> NO<br>';
   if strip(lowcase(lookup))^='n/a' and ^missing(lookup)then do;
      tempvar=resolve("&lookup2_"||varSAS);
      put '<b>Variable is restricted to the following values:</b><br>';
      i=1;
      do while(lengthn(scan(tempvar,i,'^'))>0);
         tempvarsmall=scan(tempvar,i,'^');
         put '<li>' tempvarsmall;
         i=i+1;
         end;
      end;
   if ^missing(blacklist) then do;
      put '<b>Variable cannot contain the following values:</b><br>';
      i=1;
      do while(lengthn(scan(blacklist,i,','))>0);
         tempvarsmall=scan(blacklist,i,',');
         put '<li>' tempvarsmall;
         i=i+1;
         end;
      end;
   put '<p>';
   if eof then do;
      put '</font>';
      put '</body>';
      put '</html>';
      end;
run;
%mend;

*-;
*---;
*----- APPLY BUSINESS RULES TO SAS DATA SET TO ENFORCE QUALITY CONTROL;
*---;
*-;

%macro apply_business_rules(dsn= /* data set name in LIB.DSN or DSN format */);
%local j var;
data rules_applied;
   length totReq 3 totReqMiss 3 totReqInv 3;
   format totReq 8. totReqMiss 8. totReqInv 8.;
   label totReq='Attributes Required' totReqMiss='Required Attributes Missing'
      totReqInv='Required Attributes Invalid';
   &headerlength;
   &headerformat;
```

```
   &headerlabel;
   set &dsn (rename=(&renamelist) keep=&oldvarlist);

   * apply all business rules created previously;
   &ruleslist;

   * calculate percent missing or invalid;
   totReq=0;
   totReqMiss=0;
   totReqInv=0;
   %let j=1;
      %do %while(%length(%scan(&varlist,&j,,S))>1);
         %let var=%scan(&varlist,&j,,S);
         if exp_&var=1 then totReq=totReq+1;
         if exp_&var=1 and obs_&var=2 then totReqMiss=totReqMiss+1;
         else if exp_&var=1 and obs_&var=3 then totReqInv=totReqInv+1;
         %let j=%eval(&j+1);
         %end;
   run;

   * identify duplicate values (that should be unique);
   %let j=1;
   %do %while(%length(%scan(&noduplist,&j,,S))>1);
      %let var=%scan(&noduplist,&j,,S);
      proc sort data=rules_applied;
         by &var;
      run;
      data rules_applied;
         set rules_applied;
         by &var;
         if ^missing(&var) and (^first.&var or ^last.&var) then obs_&var=4;
         if ^missing(&var) and exp_&var=1 and (^first.&var or ^last.&var)
            then obsreq_&var=4;
      run;
      %let j=%eval(&j+1);
      %end;
%mend;

*-;
*---;
*----- CREATE DATA QUALITY EXCEPTION REPORT;
*---;
*-;

%macro data_dic_report(htmlrptpath= /* path of HTML exception rpt created */,
   htmlrpt= /* file name and extension of HTML exception report created */,
   title= /* title of exception report */,
   display=ERRORS /* display=ALL to show correct and invalid records */);
%local i j errcnt;
%local today;
%let today=%sysfunc(date(),mmddyy8.);
* create text for HTML flyover popups;
data dsn_for_report;
   set rules_applied;
   length var_err1 - var_err&varcnt $200; *
```

```
      boolean values for each variable where 1 represents at least one err;
   length var_label1 - var_label&varcnt $200;
      * descriptive label of at least one finding;
   %do i=1 %to &varcnt;
      * correct if it 1) has a valid value or 2) is not missing
         and is allowed to be missing;
      if obs_%scan(&varlist,&i,,S)=1 or (exp_%scan(&varlist,&i,,S)=2
         and obs_%scan(&varlist,&i,,S)=2) then var_err&i=0;
      else if exp_%scan(&varlist,&i,,S)=1 and obs_%scan(&varlist,&i,,S)=2
            then do; * denoted MISSING if it is expected to be there;
         var_err&i=2;
         call label(exp_%scan(&varlist,&i,,S), var_label&i);
         end;
      else if obs_%scan(&varlist,&i,,S)=3 then do;
            * denoted INVALID whether it is expected to be there or not;
         var_err&i=3;
         call label(obs_%scan(&varlist,&i,,S), var_label&i);
         end;
      else if obs_%scan(&varlist,&i,,S)=4 then do;
         * denoted DUPLICATE whether it is expected to be there or not;
         var_err&i=4;
         var_label&i="%scan(&varlist,&i,,S) cannot be duplicate";
         end;
      %end;
run;
* create the reports;
proc sql noprint;
   select count(*) into: errcnt
   from rules_applied
   where totReqMiss>0 or totReqInv>0;
   quit;
%if &errcnt=0 %then %do;
   data _null_;
      file "&htmlrptpath.&htmlrpt";
      put '<html><header>';
      put '</header>';
      put '<body>';
      put '<font face=Arial>';
      put "<h3>&title<br>";
      put "<h3>Updated: &today<p>";
      put "<h3>No errors detected";
      put '</font>';
      put '</body>';
      put '</html>';
   run;
   %end;
%else %do;
   ods html path="&htmlrptpath" file="&htmlrpt";
   title "&title";
   proc report data=dsn_for_report nocenter nowindows nocompletecols
         style(report)=[foreground=black backgroundcolor=white
         background=black] style(header)=[font_size=2 background=black
         backgroundcolor=black foreground=white]
         style(column)=[backgroundcolor=very light grey];
      %if %upcase(&display)^=ALL %then %do;
```

```
          where totReqMiss>0 or totReqInv>0;
          %end;
       column obs &varlist var_err1 - var_err&varcnt var_label1 -
           var_label&varcnt dummy;
       define obs / computed '#';
       %do j=1 %to &varcnt;
          define %scan(&varlist,&j) / display;
          %end;
       %do j=1 %to &varcnt;
          define var_err&j / display noprint;
          %end;
       %do j=1 %to &varcnt;
          define var_label&j / display noprint;
          %end;
       define dummy / computed noprint;
       compute obs;
          obs_pvt+1;
          obs=obs_pvt;
          call define("_c1_",'style','style=[backgroundcolor=black
foreground=white]');
          endcomp;
       compute dummy;
          %do j=1 %to &varcnt; * create the color-coding;
             if var_err&j=2 then call define ("_c%eval(&j+1)_",
                    'style','style=[backgroundcolor=very light red]');
             else if var_err&j=3 then call define ("_c%eval(&j+1)_",
                    'style','style=[backgroundcolor=very light orange]');
             else if var_err&j=4 then call define ("_c%eval(&j+1)_",
                    'style','style=[backgroundcolor=very light yellow]');
             %end;
          %do j=1 %to &varcnt; * create the error popup labels;
             if var_err&j^=0 then call define
                    ("_c%eval(&j+1)_",'style/merge','style=[flyover="'
                    || _c%eval(&varcnt+&varcnt+&j+1)_ || '"]');
             %end;
          endcomp;
   run;
   ods html close;
   %end;
%mend;
```