SAS5644-2016

Get Out of DATA Step Code and into Quality Knowledge Bases

Brian Rineer, SAS Institute Inc.

ABSTRACT

SAS® programmers spend hours developing DATA step code to clean up their data. Save time and money and enhance your data quality results by leveraging the power of the SAS® Quality Knowledge Base (QKB). Knowledge Engineers at SAS have developed advanced context-specific data quality operations and packaged them in callable objects in the QKB. In this session a Senior Software Development Manager at SAS explains the QKB and shows how to invoke QKB operations in a DATA step and in various SAS® Data Management offerings.

INTRODUCTION

SAS programmers spend countless hours writing DATA step code to clean up their organizations' text-based data. Such coding efforts typically arise from individual ad hoc quality challenges and are therefore limited in scope. Further, results are typically only partially successful, as programmers are unable to anticipate all data values – and their attendant data quality problems – that might be observed in future executions of their code. This necessitates constant, reactive enhancement of the programs, leading to ever-growing costs.

This paper explains how to reduce programming costs by leveraging data quality capabilities built into the SAS Quality Knowledge Base (QKB), a repository of rules and reference data that can be invoked through numerous SAS software products. We show how the QKB can be used across multiple execution environments, ensuring consistent data quality results across your organization. Finally, we explain how you can customize the QKB to tackle tough data quality problems unique to your business.

WHAT IS A QUALITY KNOWLEDGE BASE?

A QKB is a collection of files that store rules, expressions, and reference data that define data quality operations. SAS software products reference a QKB when performing data quality operations on your data. The contents of a QKB are organized into a set of objects called "definitions". Each definition defines a single context-sensitive data quality operation. For example, a definition in a QKB might provide the capability to extract the name of a city from an address, so that you can search your database for customers who live in a particular metropolitan area. Another definition might provide the capability to determine the gender of an individual by analyzing the individual's name.

QKBs are developed by knowledge engineers at SAS who transform the knowledge of subject matter experts into a proprietary machine-readable format. Multiple SAS software products, including SAS Data Quality Server, DataFlux Data Management Studio, and SAS Data Loader for Hadoop, use QKBs when performing data quality operations on your data.

There are currently two QKBs available from SAS: SAS QKB for Contact Information (QKB CI) and SAS QKB for Product Data (QKB PD). Each contains a set of definitions that are designed to process data from a specific domain. QKB CI contains definitions that are designed to process data such as names, addresses, phone numbers, and company names. QKB PD contains definitions designed for data such as product descriptions, dimensions, materials, and brands. QKB CI and QKB PD are licensed separately by SAS.

EXAMPLE DATA QUALITY PROBLEM – CASING

DeShon (2015) shows how to enhance the results of the DATA step function PROPCASE by writing additional DATA step instructions to correct unintended casing transformations performed on text strings that contain acronyms, for example, this name:

```
JOHN SMITH, DVM
```

is transformed by the PROPCASE function:

```
John Smith, Dvm
```

Here the transformation of `DVM` to `Dvm` is undesired because `DVM` is an acronym that we want to render in uppercase.

DeShon solves this problem by creating a macro that loops through the words in an input string, finds words that contain no vowels, and uppercases those words.

```
%macro upper_consonants(whatvar);
%do i = 1 %to 100;
      if scan(&whatvar,&i) = ' ' then do;
      end;
      else
      if not (findc(scan(&whatvar,&i),'AEIOUY','i'))
      then do;
            substr(&whatvar,
              findw(&whatvar,trim(scan(&whatvar,&i))),
              length(trim(scan(&whatvar,&i))))
            = upcase(trim(scan(&whatvar,&i)));
      end;
%end;
%mend;
```

Each call to PROPCASE is followed by a call to this macro. This sequence produces the desired output for strings containing acronyms with no vowels, such as `DVM`:

```
John Smith, DVM
```

There are obvious limitations to this approach. First, words such as `Dr` and `2nd` that have no vowels will be uppercased by the macro even though we prefer for them to remain propercased. Also, words such as `III` that have vowels are propercased even though they should remain uppercased.

Deshon tackles these limitations by defining lists of exceptional words that must be corrected with special-case logic:

```
%let make_proper = 'DR' 'DRS' 'JR' 'MR' 'MRS' 'RD' 'SVC' 'ST' 'WM'

%let make_upper = 'ABC' 'AFB' 'APO' 'DBA' 'II' 'III' 'IV' 'NE' 'SE' 'USDA'
'XYZ';
```

The macro is adjusted such that it scans these lists and accordingly applies uppercasing or propercasing to relevant words in a post-processing step. Special handling for ordinals is also added. The result is a fifty-one line macro that corrects undesired casing results for ordinal numbers and some acronyms.

As Deshon explains, programmers must periodically update the exception lists `make_proper` and `make_upper` as new exceptions are found. Since exceptions are typically found by inspecting the output of the program, the evolution of the program occurs through a tedious reactive process and requires programmers to update macro code each time a new exception is identified.

The use of the macro with word lists is a clever improvement to the PROPCASE function. However, there are limitations to the effectiveness of this approach, even beyond the requirement for programmers to regularly update the exception word lists.

First, note that beyond searching for words that contain no vowels, the macro does not use pattern-matching to identify exceptions. This means, for example, that family names beginning with prefixes such as `Mc` or `Mac` would be incorrectly cased:

```
Ronald Mcdonald
```

To ensure correct casing results for these names, a programmer would need to write additional post-processing code. It would not suffice to add these names to the `make_proper` list, since the names do not follow normal propercasing rules.

Further, the special-case logic in the macro is not context-specific. Consider the following example:

```
Boston, MA
```

To correctly case the word `MA` in this string, we could add `MA` to the `make_upper` list in the macro. But now consider this example:

```
Yo-Yo Ma
```

Here the word `Ma` is not a state abbreviation, but a family name. In this case we do not want to uppercase the word `Ma`. In order to uppercase the word `MA` when it appears as a state abbreviation, and propercase it when it appears as a family name, a programmer would need to create two separate macros – one for use with address data and one for use with names.

Even more challenging is the situation in which a word requires different casing rules based on its positioning within a string. Consider this example:

```
HANK WILLIAMS JR
```

Here we can state with confidence that the word `JR` should be propercased to `Jr`. But what about in the following example?

```
JR EWING
```

Here we prefer for the initials `JR` to be uppercased. In fact, it would be even better if we could add a whitespace between the initials and render the name as follows:

```
J R Ewing
```

How can we achieve these results through DATA step programming? Certainly this type of context-specific handling would require a tremendous amount of programming effort.

USING THE QKB FOR CASING

As an alternative to creating long and complex DATA step programs, we can consider using the DQCASE function to invoke a Case definition in the QKB. Here is an example call to the DQCASE function:

```
DATA _null_;
  propname=dqcase("RONALD MCDONALD", 'Proper (Name)');
run;
```

In this example, the DQCASE function transforms the string `RONALD MCDONALD` by invoking a Case definition called **Proper (Name)**. After executing this DATA step, the value of the variable `PROPNAME` is:

```
Ronald McDonald
```

The **Proper (Name)** Case definition is designed to propercase names of individuals. It works by applying a series of table- and expression-based transformations that have been developed specially for name data. For example, regular expressions are used to capture functionality analogous to that in Deshon's simple acronym finding macro:

Regular Expression	Substitution	Notes
(?i)^((B|C|Ç|Ć|Ĉ|Ċ|Č|D|Ď|Đ|F|G|Ğ|Ġ|Ģ|H|...	\U\1	three or more consonants
(?i)^(\p{Latin})((\1){2,})(?=$)	\U\1\2	three or more instances of the same letter

Display 1. Regular Expressions in a Case Definition in the QKB

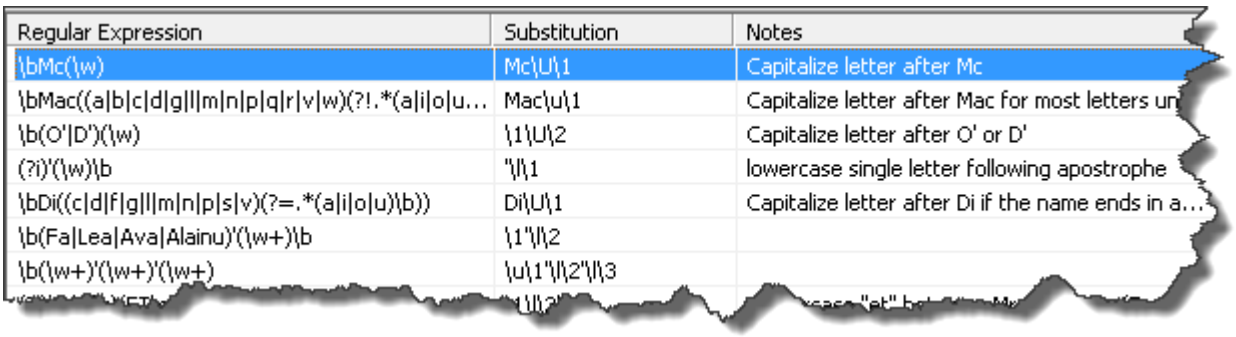
A two-column table is used to specify words and map them to corresponding desired cased values. An excerpt of such a table is shown here.

Data	Standard
GM	GM
HR	HR
IHM	IHM
II	II
III	III
IS	IS
IT	IT
IV	IV
JD	JD
JP	JP

Display 2. Standardization Table in a Case Definition in the QKB

When a term in the left column is encountered in an input string, it is transformed to the corresponding value in the right column. The lookup is itself case-insensitive, so, for example, `Gm` will be transformed to `GM`.

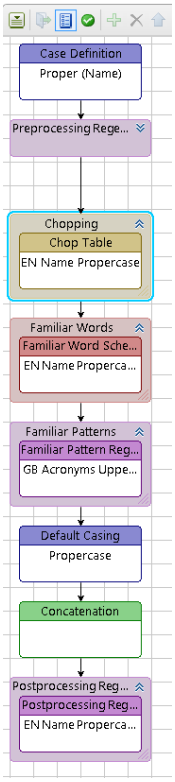
Additional regular expressions perform transformations that are consistent with subject matter experts' understanding of casing rules for names.



Regular Expression	Substitution	Notes
\bMc(\w)	Mc\U\1	Capitalize letter after Mc
\bMac((a|b|c|d|g|l|m|n|p|q|r|v|w)(?!.*(a|i|o|u...	Mac\u\1	Capitalize letter after Mac for most letters un
\b(O'|D')(\w)	\1\U\2	Capitalize letter after O' or D'
(?i)'(\w)\b	'\l\1	lowercase single letter following apostrophe
\bDi((c|d|f|g|l|m|n|p|s|v)(?=.*(a|i|o|u)\b))	Di\U\1	Capitalize letter after Di if the name ends in a...
\b(Fa|Lea|Ava|Alainu)'(\w+)\b	\1'\l\2	
\b(\w+)'(\w+)'(\w+)	\u\1'\l\2'\l\3	

Display 3. Regular Expressions Implementing Knowledge-Based Casing Rules in a Case Definition

These transformations are combined in a process flow to produce a Case definition with complex casing behavior. This screenshot from a QKB editing tool shows the process flow for the **Proper (Name)** Case definition:



Display 4. Flow Diagram for a Case Definition

When DQCASE applies the **Proper (Name)** Case definition to the string `YO-YO MA` , it propercases the word `MA` according to normal propercasing rules. It does not interpret `MA` as an abbreviation for Massachusetts, because state abbreviations do not belong to the domain of names of individual people. In order to correctly propercase `MA` when it appears as a state abbreviation, we need for the software to know that we are operating in a different context. We can do this by invoking a different Case definition – the **Proper (City - State/Province - Postal Code)** Case definition.

The **Proper (City - State/Province - Postal Code)** Case definition understands that MA is a state abbreviation in this string:

```
BOSTON, MA
```

Accordingly, it will produce the following output:

```
Boston, MA
```

By using domain-specific Case definitions in conjunction with the DQCASE function, we can take advantage of the power of the QKB and drastically reduce the amount of effort required to write our DATA step program.

Now let's consider a more challenging case – a case in which a word requires different casing rules in different situations, even in the same contextual domain. Recall the two examples from above:

```
HANK WILLIAMS JR
JR EWING
```

The desired renderings of these two strings are as follows:

```
Hank Williams, Jr
J R Ewing
```

Unfortunately, the **Proper (Name)** Case definition does not understand that `JR` is a pair of initials in the name `JR EWING`. It will transform `JR` to `Jr` regardless of where the word appears relative to other words in a name. Further, the **Proper (Name)** Case definition will not split `JR` into two separate initials; nor will it add a comma after `Williams` in the first example. To obtain these types of transformations, we need a more complex type of QKB definition.

USING THE QKB FOR STANDARDIZATION

The type of QKB definition we need for these transformations is called a Standardization definition. Standardization definitions are similar to Case definitions – both use table- and expression-based transformations to implement complex processing. But Standardization definitions also use natural language processing techniques to recognize semantically distinct substrings, or "tokens", within a string. After identifying tokens, a Standardization definition applies specific transformation rules to each token.

A programmer can invoke a Standardization definition in a similar fashion to that used to invoke a Case definition. But instead of using the function DQCASE, we use a function called DQSTANDARDIZE. Here is an example in which we use DQSTANDARDIZE to invoke the **Name** Standardization definition:

```
DATA _null_;
  stdname=dqstandardize("JR EWING", 'Name');
run;
```

Since Standardization definitions recognize semantically distinct tokens within a string, they can rearrange tokens to render a string with a preferred word order, for example, for this input:

```
MCDONALD, MISTER RONALD, JUNIOR
```

The **Name** Standardization definition will produce the following output:

```
Mr Ronald McDonald, Jr
```

Naturally, it would be very difficult and time-consuming to replicate this type of processing through DATA step programming. Using Case definitions and Standardization definitions saves time and improves data quality results.

EXAMPLE DATA QUALITY PROBLEM – DATA MATCHING

The previous sections illustrate the advantages of using the QKB for casing and standardization. Now we'll consider another common data quality challenge – fuzzy data matching. Cantu-Perez and Klekar (2015) describe an approach for joining physician records from two disparate health care data sets. In their use case, the only field common to the two sets is a field containing the names of the individual physicians. The join must therefore be done using only physician name data. The authors note several challenges involved with joining name data:

- The optional presence of suffixes such as "Jr" or "III"
- The use of nicknames and diminutive forms of given names
- Inconsistent use of punctuation such as '-'
- Misspellings

To circumvent these difficulties, Cantu-Perez and Klekar use Structured Query Language (SQL) instructions and calls to multiple DATA step functions to perform fuzzy matches on the physician names. This approach allows them to match names such as the following:

```
SAM BOONE
SAMSON BOONE

STEVEN WILLIAMS
STEVEN WILLIAMS II
```

Cantu-Perez and Klekar use two programs to perform their fuzzy joins. The programs consist of seventeen and twenty-four lines of code. They contain complex SQL. They include a hardcoded list of

characters that are ignored during string comparisons, and calls to the spelling-distance calculation function SPEDIS.

The SPEDIS function enables matching of names that differ only slightly in spelling, such as `GRIFFIN GRAEHL` and `GRIFIN GRAEHL`. Unfortunately, SPEDIS is not effective at matching names in which one name contains a diminutive form of the other, where the diminutive form and the original form have very different spellings. For example, a solution using SPEDIS will have difficulty matching names like `BOB JONES` and `ROBERT JONES`, or `DICK GEPHARDT` and `RICHARD GEPHARDT`.

USING THE QUALITY KNOWLEDGE BASE FOR FUZZY DATA MATCHING

The QKB provides a type of definition called a Match definition as a facility for performing context-specific fuzzy matching. A Match definition reads an input string and creates a corresponding "matchcode", which is a fuzzy representation of the input string. Here's an example of how to invoke the **Name** Match definition to generate matchcodes for names in two data sets using the DATA step function DQMATCH:

```
DATA physician_match;
  set physicians;
  matchcode=dqmatch(physician_name, 'Name');
run;

DATA clinical_data_match;
  set clinical_data;
  matchcode=dqmatch(physician_name, 'Name');
run;
```

In each output data set, the `matchcode` variable will be populated with a matchcode that is a fuzzy representation of the name stored in the `physician_name` variable.

To extend the example from the previous section, let's look at the matchcodes for the two names `SAM BOONE` and `SAMSON BOONE`. Notice that these two strings get the same matchcode in each of the two example data sets, as do `STEVEN WILLIAMS` and `STEVEN WILLIAMS II`:

Data Set: Physicians

physician_name	matchcode
SAM BOONE	M&B$$$$$$$$$$$$4&B$$$$$$$$$
WILLIAM STEVENS	4~&M&B$$$$$$$$$M7WW$$$$$$$$

Data Set: Clinical Data

physician_name	matchcode
SAMSON BOONE	M&B$$$$$$$$$$$$4&B$$$$$$$$$
WILLIAM STEVENS II	4~&M&B$$$$$$$$$M7WW$$$$$$$$

After executing the above two DATA steps to create a matchcode for every name in each data set, we can perform a simple join on the matchcode fields in the two data sets to achieve the desired join results. This is a simpler approach than writing complex SQL and using SPEDIS. What's more, by using the **Name** Match definition, we can achieve matches that would not be feasible with SPEDIS. For example, we would be able to match physician names where the given name is `BOB` in one data set and `ROBERT` in the other.

(Note: We could perhaps match names like BOB and ROBERT using SPEDIS if we allow matches at a very low SPEDIS score threshold. However, that approach would probably lead to many undesired false positives.)

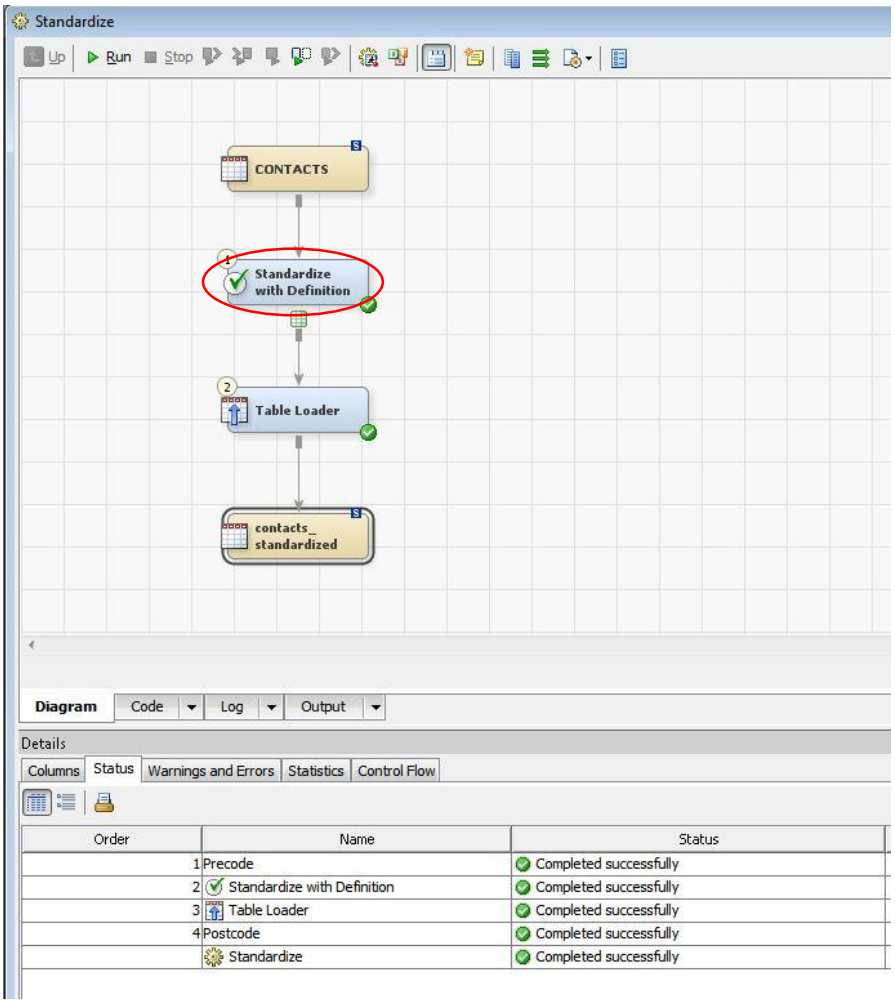
Much like Case definitions and Standardization definitions, Match definitions use a combination of natural language processing rules, regular expressions, and transformation tables to implement complex, intelligent behavior. Match definitions also use phonetic reduction rules, which allow for matching words with similar sounds. Finally, Match definitions also accept an optional "sensitivity" parameter, which controls the level of fuzziness used when creating a matchcode. This allows users to specify looser or tighter matching depending on the nature of the job at hand.

ADDITIONAL DATA QUALITY OPERATIONS

The QKB supports additional data quality operations besides Casing, Standardization, and Matching. For example, you can use QKB definitions to determine the semantic type of a string or to determine the gender of a name. You can extract attributes and substrings from an input string, or generate a character pattern representing a string for use in a data profiling application. See Rineer (2015) for examples of additional operations supported by the QKB.

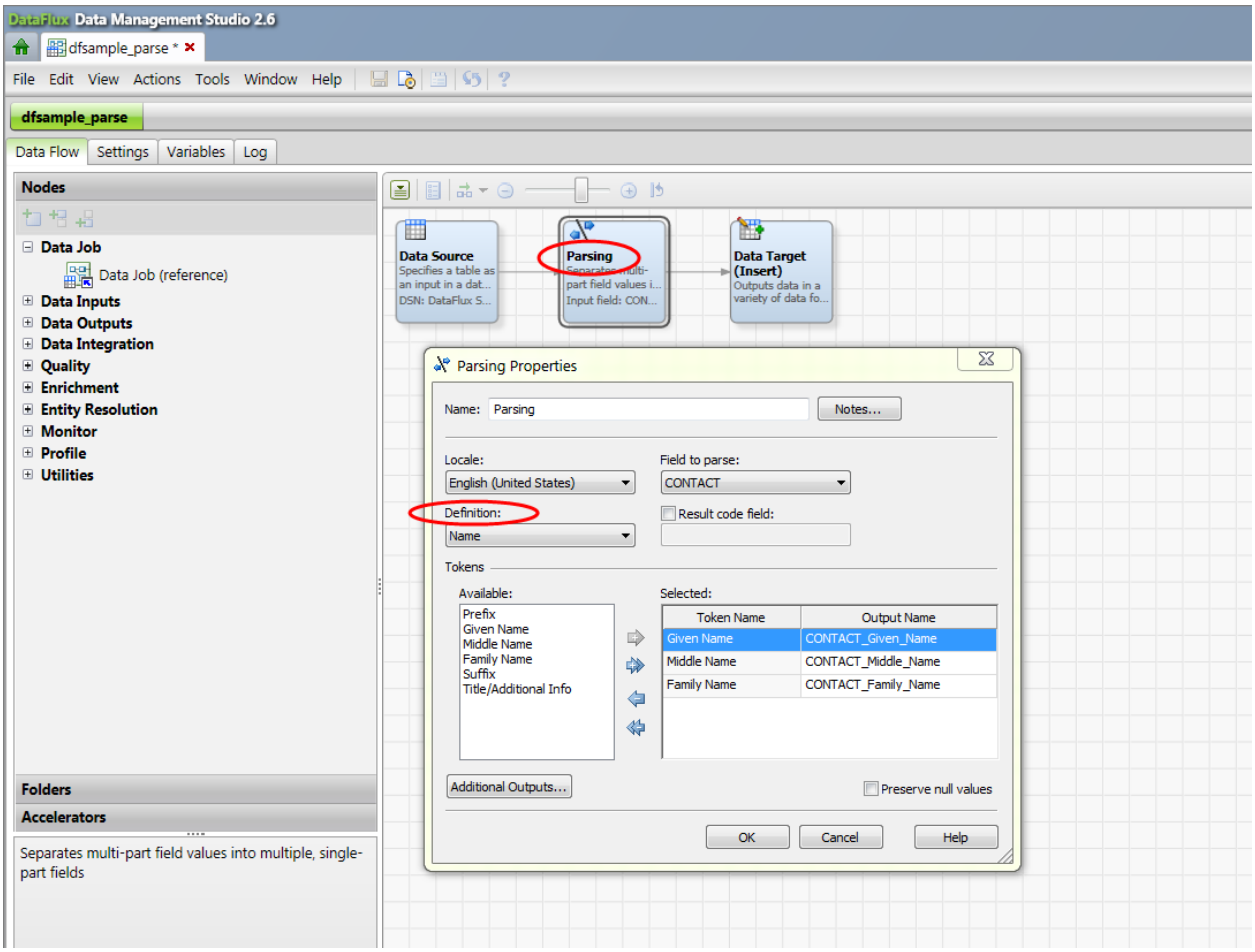
USING A QKB OUTSIDE THE DATA STEP

SAS provides access to the QKB via multiple products that are available on different execution environments. Those who prefer a graphical user interface over writing SAS code might choose to invoke QKB definitions via SAS Data Integration Studio transformations:



Display 5. Using a QKB in SAS Data Integration Studio

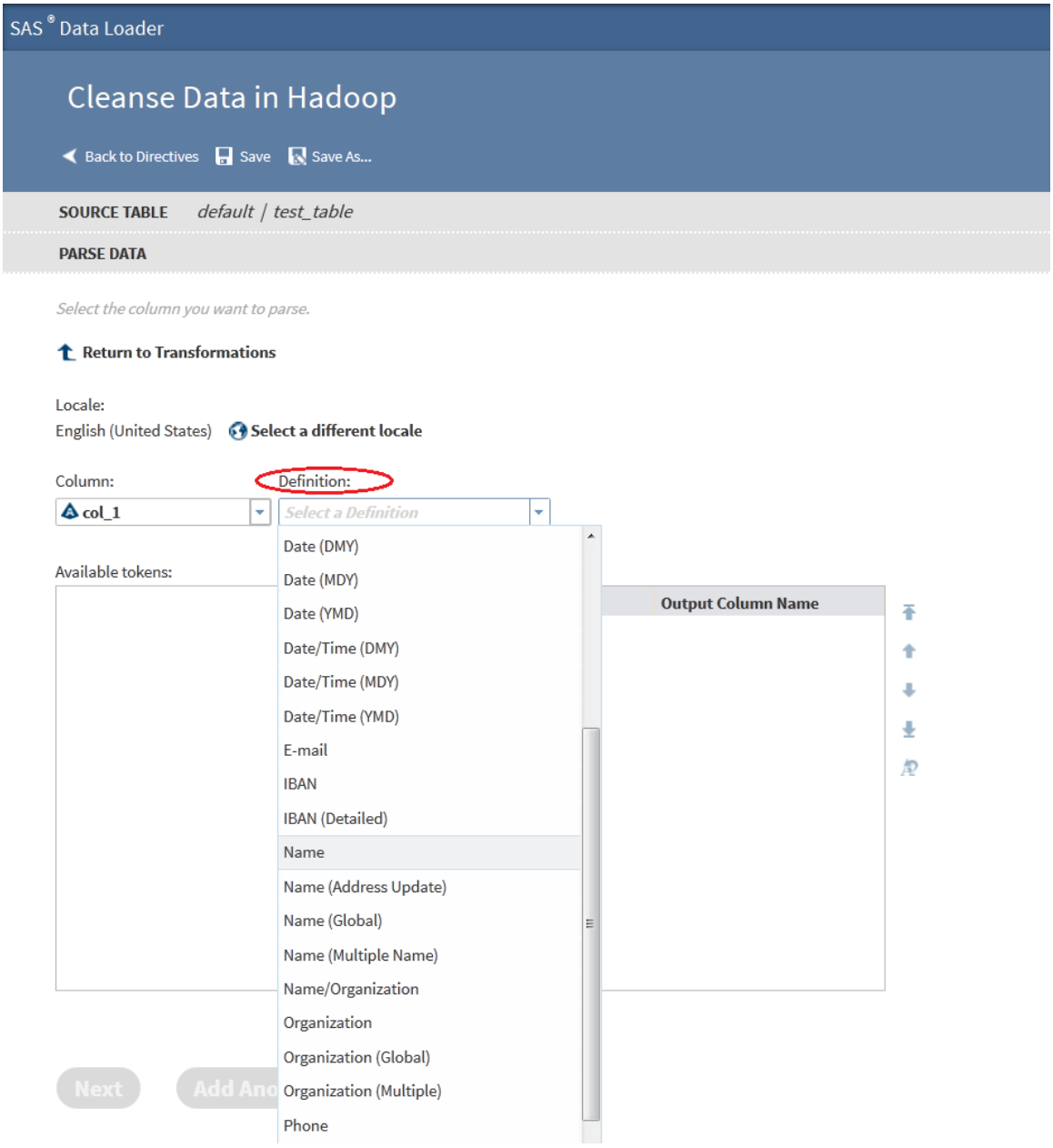
DataFlux Data Management Studio users can access QKB definitions through nodes in a Data Job:



Display 6. Using a QKB in DataFlux Data Management Studio

Data Jobs created with the Windows only DataFlux Data Management Studio might be executed in the DataFlux Data Management Server, which is available for several UNIX platforms.

Hadoop users can apply QKB definitions to their data in Hadoop by using the SAS Data Loader for Hadoop:



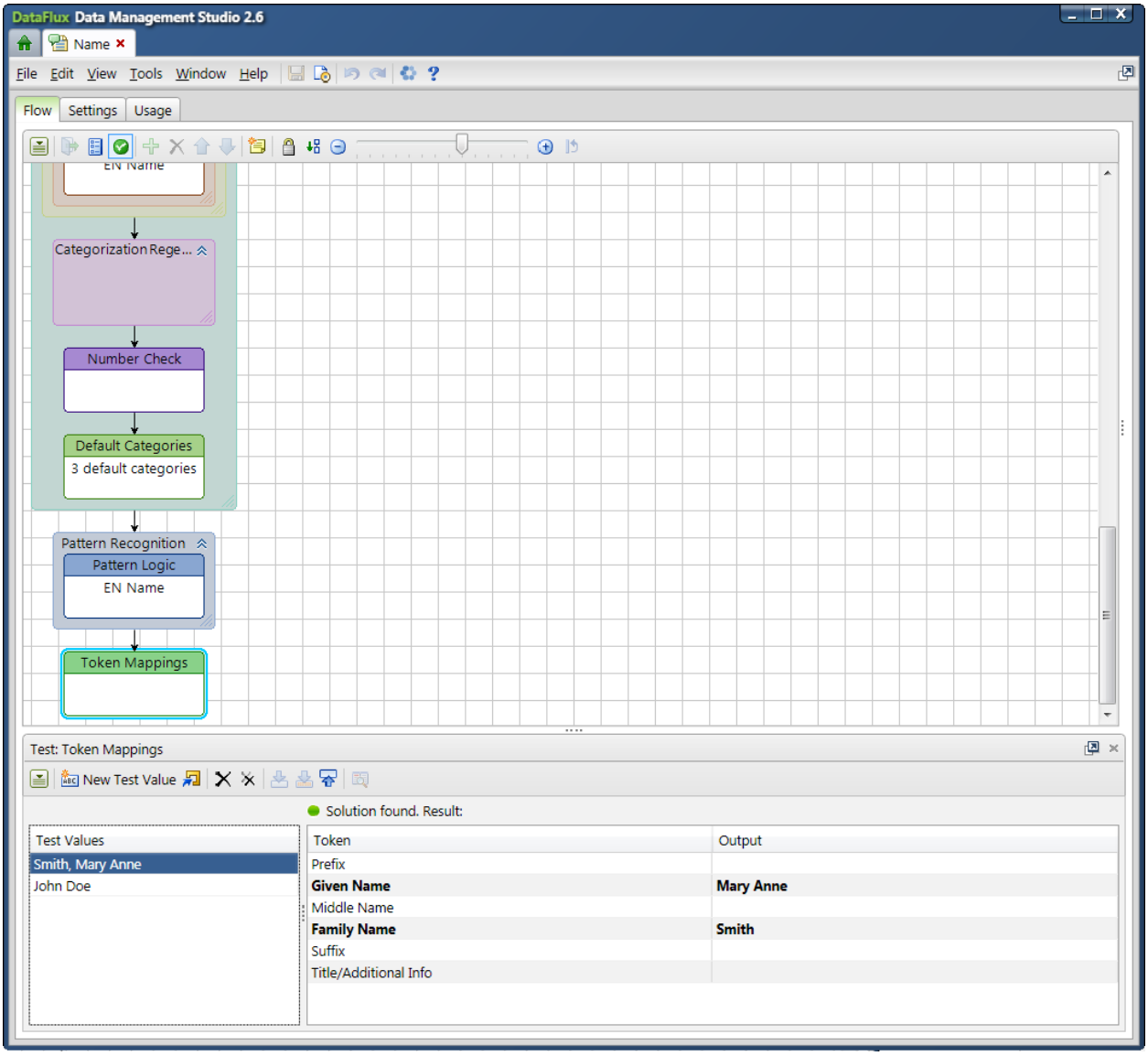
Display 7. Using a QKB in SAS Data Loader for Hadoop

Because all SAS data management products use the same data quality engine, results obtained with the same QKB are consistent across products and execution environments. This means, for example, that you could test QKB definitions on sample data using SAS Data Quality Server or DataFlux Data Management Studio, and then use those same definitions to process data in your production environment in Hadoop with SAS Data Loader for Hadoop – and you can be confident that results will be the same in both environments.

CUSTOMIZING A QKB

SAS releases regular updates to QKBs, introducing new definitions and enhancing the behavior of existing ones. Many users meet their data quality needs with an out-of-the-box QKB, perhaps periodically upgrading to a newer version. However, you might have data quality challenges unique to your business, for which you'd like enhanced support in the QKB. You can add this support yourself by customizing your QKB using DataFlux Data Management Studio.

In DataFlux Data Management Studio, you can open your QKB and browse its contents in a graphic navigator. You can edit individual definitions in a specialized editing tool, and test individual definitions with particular input strings before deciding whether you want to apply those definitions to your production data.



Display 8. Editing a QKB in DataFlux Data Management Studio

Any edits you make to your QKB can be used across all platforms in your enterprise with any of the SAS data quality products listed above.

While edits to some components of a QKB are simple, understanding the more detailed nuances of QKB definitions and the full capabilities of the QKB editing tools typically requires training. Once mastered, however, QKB customization skills are a valuable asset to any organization interested in enhancing the quality of its data resources. Contact your SAS account executive for information about QKB customization training.

FURTHER INFO

Craver (2015) and Rineer (2015) provide details about how to deploy and use a QKB in your enterprise. See the SAS Data Quality Server online documentation for a list of the various data quality operations that are supported by the QKB. You can also read QKB online documentation to see what definitions are available for use with your data.

Finally, contact your SAS account executive for information about how to license SAS Data Quality Server and other data management software that leverages the QKB.

CONCLUSION

The QKB is a powerful tool for users who wish to apply data quality operations to their data without spending months struggling with complex DATA step programs. You can leverage years of knowledge engineering work performed by SAS and encoded in the out-of-the-box QKB. Further, you can enhance the QKB with logic needed to solve quality problems unique to your business. DATA step functions and PROCs in SAS Data Quality Server and graphical user interfaces in other SAS data management products make it easy to obtain consistent data quality results across multiple processing environments.

REFERENCES

- DeShon, Joe. 2015. “Fixing Lowercase Acronyms Left Over from the PROPCASE Function”. *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/1752-2015.pdf.
- Cantu-Perez, Gabriela, MPH, Klekar, Christopher, MPH, MBA. 2015. “Data Management Techniques for Complex Healthcare Data”. *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/3104-2015.pdf.
- Craver, Mark. 2015. SAS® Data Management: Technology Options for Ensuring a Quality Journey Through the Data Management Process. *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/SAS1907-2015.pdf.
- Rineer, Brian. 2015. “Garbage In, Gourmet Out: How to Leverage the Power of the SAS® Quality Knowledge Base”. *Proceedings of the SAS Global Forum 2015 Conference.* Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/SAS1390-2015.pdf.

RECOMMENDED READING

- Rineer, Brian. 2015. “Garbage In, Gourmet Out: How to Leverage the Power of the SAS® Quality Knowledge Base.” *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/SAS1390-2015.pdf.
- Craver, Mark. 2015. SAS® Data Management: Technology Options for Ensuring a Quality Journey Through the Data Management Process. *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings15/SAS1907-2015.pdf.
- Quality Knowledge Base documentation. Available http://support.sas.com/documentation/onlinedoc/qkb/.
- SAS Data Quality Server documentation. Available http://support.sas.com/documentation/cdl/en/dqclref/68376/HTML/default/viewer.htm#titlepage.htm.

CONTACT INFORMATION

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

Brian Rineer
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
+1-919-677-8000
Brian.Rineer@sas.com
http://www.sas.com