SAS Publishing

SAS® Data Quality Server 9.1
Reference

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS® Data Quality Server 9.1: Reference*. Cary, NC: SAS Institute Inc.

SAS® Data Quality Server 9.1: Reference

Copyright © 2004, SAS Institute Inc., Cary, NC, USA

ISBN 1-59047-259-4

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, January 2004

Contents

What’s New

Overview

New capabilities, functions, and options in SAS Data Quality Server (formerly called SAS Data Quality – Cleanse) improve your ability to:

- analyze the quality of your data
- minimize defective data
- reduce redundancies
- transform and standardize data
- merge and recombine data.

Note: z/OS is the successor to the OS/390 operating system. SAS Data Quality Server 9.1 is supported on both OS/390 and z/OS operating systems. Throughout this document, any reference to z/OS also applies to OS/390. △

General Enhancements

- SAS Data Quality Server is now available under the z/OS (formerly OS/390) operating environment.
- The following locales are provided:
  - DEDEU, German language, for use with data from Germany
  - ENUSA, English language for the United States
  - ENGBR, English language for Great Britain
  - ENAUS, English language for Australia
  - ITITA, Italian language for Italy
  - NLNLD, Dutch language for the Netherlands.

- The software is now closely integrated with:
  - SAS ETL Studio for data cleansing as part of enterprisewide extract, transform, and load.
  - dfPower Studio and Blue Fusion from DataFlux (a SAS company) for locale editing and additional data cleansing functionality.
- The SAS Sample Library now contains samples for SAS Data Quality Server 9.1.

Functions

The following new functions interpret input character values that have been parsed (and therefore contain delimiters):

- DQMATCHPARSED returns a match code from a parsed character value.
- DQGENDERPARSED returns a gender determination from the parsed name of an individual.

The next set of new functions helps you prepare parsed character values:

- DQPARSETOKENPUT creates a new parsed value or adds a parsed value to an existing parsed value.
- DQMATCHINFOGET returns the name of the parse definition that is associated with a specified match definition.
- DQGENDERINFOGET returns the name of the parse definition that is associated with a specified gender definition.

The new function DQLOCALEINFOGET returns a list of the locales that are currently loaded into memory.

The new function DQPATTERN returns a pattern analysis of the words or characters in an input character value.

The existing function DQSCHEMEAPPLY and the existing CALL routine CALL DQSCHEMEAPPLY now accept the following new arguments: *scheme-lookup-method*, *match-definition*, *sensitivity*, and *locale*. These arguments implement the new scheme-apply capabilities that are also available in the DQSCHEME procedure for the APPLY statement.

System Options

The following system options are new or have changed:

- DQLOCALE= specifies the locales that are to be loaded into memory.
- DQSETUPLOC= specifies the location of the setup file for SAS Data Quality Server.

AUTOCALL Macros

The following AUTOCALL macros are new or have changed:

- %DQPUTLOC displays in the SAS log all the definitions and tokens in a specified locale.
- %DQLOAD loads specified locales. The new DQINFO parameter generates additional information in the SAS log for debugging purposes.
- %DQUNLOAD unloads all locales.

DQSCHEME Procedure

In the PROC DQSCHEME statement, the BFD and NOBFD options enable you to generate schemes in SAS format and in BFD format. BFD format schemes can be displayed and edited using dfPower Customize from DataFlux (a SAS company).

The new CONVERT statement enables you to convert existing schemes between SAS and BFD formats.

For the CREATE statement in the DQSCHEME procedure, the following options are new or have been changed:

- INCLUDE_ALL enables you to fully populate the scheme. The scheme includes all output values, including those that are not transformed.
- MODE= is stored in the scheme to specify that the scheme will, by default, be applied to the entirety of each value of the input variable (when MODE=PHRASE), or to each element in each value (when MODE=ELEMENT). The value of MODE= that is stored in the scheme can be overridden by the value of the MODE= option in the APPLY statement, or in the *mode* argument in the DQSCHEMEAPPLY function or CALL routine.
- SENSITIVITY= enables you to specify the degree of complexity in the match codes that are generated internally when the scheme is built. Higher sensitivity values generate match codes that are more complex. Complex match codes are useful when you want a higher degree of similarity between the DATA values that map to a given STANDARD value.
- MATCHDEF=, which was previously known as MATCHTYPE=, specifies the match definition that is referenced during the creation of match codes.
- LOCALE= specifies a locale.

For the APPLY statement in the DQSCHEME procedure, MODE= specifies whether to apply the scheme to the entire input value or to each element of the input value. The default value is determined by the value of MODE= that was stored in the scheme when the scheme was created.

DQMATCH Procedure

For the PROC DQMATCH statement, the options DELIMITER and NODELIMITER enable you to generate concatenated match codes with or without a delimiter.

For the CRITERIA statement in the DQMATCH procedure, the following options are new or have been changed:

- SENSITIVITY= now has a maximum value of 95, which equates with the maximum sensitivity in dfPower Studio.
- MATCHCODE= enables you to generate more than one match code in a single pass through the data.
- DELIMSTR= enables you to generate match codes for parsed input character variables.
- MATCHDEF=, which was previously known as MATCHTYPE=, specifies the match definition that will be referenced during the creation of match codes.
- LOCALE= specifies a locale.

CHAPTER 1

Introduction to SAS Data Quality Server

Overview of SAS Data Quality Server

The SAS Data Quality Server software enables you to analyze, cleanse, transform, and standardize your data. Data cleansing increases the accuracy and profitability of the knowledge that you extract from your data.

You can use the SAS Data Quality Server software to:

- *Analyze and transform data* by creating and applying schemes. For details, see Chapter 4, "The DQSCHEME Procedure," on page 29.
- *Create match codes* for analysis, reporting, and transformation. For details, see Chapter 3, "The DQMATCH Procedure," on page 19 and "About the Match Code Functions" on page 15.
- *Parse character values* to allow analysis and cleansing of the elements in character values. For details, see "About the Parsing Functions" on page 15.
- *Standardize* values for format, case (uppercase/lowercase), and punctuation. For details, see "DQCASE Function" on page 48.
- *Determine the gender* of an individual based on a name. For details, see "DQGENDER Function" on page 49.
- *Determine the locale* that best represents a given value. For details, see "DQLOCALEGUESS Function" on page 53.
- *Identify* the type of content in character values. For details, see "About the Gender Analysis, Locale Guessing, and Identification Functions" on page 16.
- *Analyze the patterns* in character data. For details, see "DQPATTERN Function" on page 64.
- *Integrate* the scheme-edit and locale-edit capabilities of the dfPower Studio and dfPowerCustomize software from DataFlux (a SAS company). For information on DataFlux products, see **`www.dataflux.com`**.
- *Apply schemes and create match codes* in data quality process templates that are available for purchase and use with the SAS ETL Studio software. For information on the SAS ETL Studio software, see **`support.sas.com/technologies/dw/`**.

CHAPTER 2

Using the SAS Data Quality Server Software

About the Quality Knowledge Base

The Quality Knowledge Base consists of a hierarchy of *locales* and a collection of other files. Specified locales are loaded into memory for reference during data cleansing.

Locales are specific to a spoken language and geographic region. For example, the ENAUS locale contains data cleansing *definitions* that are specific to the English language, for use with data from or pertaining to the region of Australia. Upper-level locales, such as EN for English, contain definitions that are inherited by region–specific locales. The hierarchy of locales, along with descriptions of the definitions in those locales, is provided in Chapter 7, "Locales," on page 77.

During data cleansing, SAS references a specified *definition* in a specified locale to perform a given task, such as standardization, the creation of match codes, or the parsing of input values. Definitions differ between locales because they accommodate regional differences in data.

To illustrate the regional differences in definitions, the parse definition ADDRESS in the ENUSA locale is referenced to associate parts of street addresses with the following tokens:

- STREET NUMBER
- PRE-DIRECTION
- STREET NAME
- STREET TYPE
- POST-DIRECTION
- ADDRESS EXTENSION
- ADDRESS EXTENSION NUMBER

In the ENGBR locale (English, Great Britain), the ADDRESS parse definition has the following tokens:

- PREFIX
- HOUSE NAME
- STREET NUMBER
- STREET NAME
- STREET TYPE
- NEIGHBOURHOOD

Data cleansing definitions are tailored to accommodate the data formats in specific geographic regions.

Global definitions enable consistent data cleansing across locales. For example, the parse definition PHONE (GLOBAL) provides a consistent set of parse tokens in all locales. The parsing operation that seeks values for the tokens differs between locales, but the results can be compared or combined across locales.

To summarize, the Quality Knowledge Base contains a hierarchy of locales that are language-specific and region-specific. Locales contain definitions that are usage-specific and content-specific. For further information on locales, see "About Locale Definitions" on page 11.

You can edit locales using the dfPower Customize software from DataFlux (a SAS company). For information on DataFlux software products, see **`www.dataflux.com`**.

About Locales

Before locales can be referenced for data cleansing, they must be loaded into memory using the AUTOCALL macro %DQLOAD. The macro sets the value of the system options DQSETUPLOC= and DQLOCALE= and loads the specified locales into memory.

The system option DQSETUPLOC= specifies the location of the data quality setup file; that file in turn specifies the location of the Quality Knowledge Base. For further information on the setup file, see "About the Data Quality Setup File" on page 5.

When you run %DQLOAD (see "%DQLOAD AUTOCALL Macro" on page 41), a specified list of locales is loaded into memory. That list of locales becomes the value of the DQLOCALE= system option.

To ensure that you use the intended locale from the intended Quality Knowledge Base, be sure to run %DQLOAD before data cleansing.

Note that you can change the values of the system options DQSETUPLOC= and DQLOCALE=, but doing so does not load different locales into memory. For this reason, it is recommended that you use %DQLOAD to change the values of the two data quality system options.

The first locale in the locale list is the default locale. The default locale is referenced when:

- A locale is not specified.
- The specified locale is not loaded into memory.
- Input data is insufficient for the DQLOCALEGUESS function (see“DQLOCALEGUESS Function” on page 53) to determine the best locale for that data.

If you change locale files in the Quality Knowledge Base, make sure that you reload macros into memory with %DQLOAD before data cleansing.

You can display information on a locale that is currently loaded into memory using the AUTOCALL macro %DQPUTLOC. For further information on that macro, see “%DQPUTLOC AUTOCALL Macro” on page 42.

After you submit your data cleansing programs, you can restore the memory by unloading locales using the AUTOCALL macro %DQUNLOAD (see “%DQUNLOAD AUTOCALL Macro” on page 43).

New locales and updates to existing locales are provided by DataFlux (a SAS company). You can download new locales from the following Web address:

```
www.dataflux.com/QKB
```

Downloading a locale also downloads new reference documentation for that locale.

About the Data Quality Setup File

The setup file for the SAS Data Quality Server software specifies the storage locations that comprise the Quality Knowledge Base. In the Windows and UNIX operating environments, the name of the setup file is DQSETUP.TXT. In the z/OS operating environment, the setup file is a member named DQSETUP in a SAS Data Quality Configuration PDS.

The location of the setup file is specified with the system option DQSETUPLOC=. If you move the setup file, you need to change the value of this system option. The value may be set by default when you initialize SAS. If you have more than one Quality Knowledge Base, the DQSETUPLOC= system option must be set to a value that points to the right one.

Editing the SAS Data Quality Server Setup File

The data quality setup file consists of a table with two columns. The first column lists file access methods, which must always be DISK. The second column provides fully-qualified paths to the contents of the Quality Knowledge Base.

When you edit the setup file, be sure to:

- Always include a semicolon at the end of each fully-qualified path.
- Specify the DISK access method only.
- In the Windows and UNIX operating environments, do not change the last directory or filename in any path. For the z/OS operating environment, do not

change the PDS names, and don't change the names of their contents. For example, in the following entry in the setup file, you would retain the GRAMMAR PDS name and not change any member names inside that PDS:

```
DISK SAS91.DQ.GRAMMAR;
```

If you would like to add comment lines to your setup file, specify an exclamation point (!) as the first non-blank character in each line that contains comments. When an exclamation point is detected in this position, the software ignores any other text on that line.

You can insert blank lines into the setup file.

In the setup file, the maximum record (line) length is 1024 characters.

About Input Requirements

The SAS Data Quality Server functions and the DQSCHEME procedure process character values up to a maximum length of 255 characters. The functions check the actual length of each input value. The DQSCHEME procedure logs an error if the defined length of a variable exceeds 255, regardless of the actual length of those values.

The DQMATCH procedure does not evaluate the length of input values, because match codes would almost certainly be the same for input values that diverged after the 255th character.

Transforming Data with Schemes

A scheme is a data set that is created for single input character variable. After a scheme is created, it is applied to that variable to transform its data. The purpose of the transformation is to replace values that are similar with a single most-common value.

The DQSCHEME procedure (see Chapter 4, "The DQSCHEME Procedure," on page 29) is used to create and apply schemes. You can create and apply multiple schemes in a single procedure step. The DQSCHEMEAPPLY function and CALL routine are also used to apply schemes (see "DQSCHEMEAPPLY CALL Routine" on page 65). You can also display, create, apply, and edit schemes in the dfPower Studio software from DataFlux (a SAS company).

Creating Schemes

Before you apply a scheme (see "Applying Schemes" on page 7), you first create the scheme data set. Schemes are created with the CREATE statement in the DQSCHEME procedure. When you submit a CREATE statement, PROC DQSCHEME creates match codes and assigns cluster numbers. A unique cluster number is assigned to each group of two or more input values that generate the same match code. After cluster numbers are assigned, the transformation value is determined. The transformation value is the one that is the most common value in each cluster.

Note: During scheme creation, the DQSCHEME procedure evaluates the definition of each input variable in each CREATE statement. An error message is generated if the defined length of an input variable exceeds 255 characters. △

Scheme data sets are created in SAS format or BFD format. BFD stands for Blue Fusion Data, which is a format that is recognized by SAS and by the dfPower Studio software. The SAS Data Quality Server software can create, apply, and display schemes in SAS format or BFD format. The dfPower Studio software can create, apply, display, and edit schemes in BFD format only. In the z/OS operating environment (formerly OS/390), the SAS Data Quality Server software can create, apply, and display schemes in SAS format; schemes in BFD format can be applied.

About Analysis Data Sets

Analysis data sets describe how a transformation would take place, without actually transforming the data; they enable you to experiment with different options to arrive at a scheme that provides optimal data cleansing. Analysis data sets are generated by specifying the ANALYSIS= option in the CREATE statement of the DQSCHEME procedure.

The key to optimizing a scheme is to choose the sensitivity value that best suits your data and your goal. You can create a series of analysis data sets using different sensitivity values to compare the results. Changing the sensitivity value changes the clustering of input values, as described in “About Sensitivity” on page 11.

When you decide on a sensitivity level you can create the scheme data set, by replacing the ANALYSIS= option with the SCHEME= option in the CREATE statement. For further information on sensitivity values and how they affect the creation of match codes and clusters, see “About Sensitivity” on page 11.

Viewing the analysis data set shows that it contains one observation for each unique input value. Any adjacent blank spaces are removed from the input values. The COUNT variable describes the number of occurrences of that value. The CLUSTER variable designates a cluster number for each input character value.

The CLUSTER variable contains a value only when two or more *different* input values generate the same match code. An input value that is repeated receives a cluster number only if another value generates the same match code.

You can specify the INCLUDE_ALL option in the CREATE statement to include all input values in the scheme, including the unique input values that did not receive a cluster number in the analysis data set.

Applying Schemes

After you create a scheme data set (see “Creating Schemes” on page 6), you apply it to an input variable to transform its values. You can apply a scheme with the APPLY statement in the DQSCHEME procedure (see “APPLY Statement” on page 33), or with the DQSCHEMEAPPLY function or CALL routine (see “DQSCHEMEAPPLY Function” on page 69). Use the CALL routine rather than the function if you want to return the number of transformations that occurred during the application of the scheme.

The scheme data set consists of the DATA and STANDARD variables. The DATA variable contains the input character values that were used to create the scheme. The STANDARD variable contains the transformation values. All of the DATA values in a given cluster have the same STANDARD value. The STANDARD values are the values that were the most common values in each cluster when the scheme was created.

When you apply a scheme to a SAS data set, an input value is transformed when the input value matches a DATA value in the scheme. The transformation replaces the input value with the transformation value.

The lookup method determines how the input value is matched to the DATA values in the scheme. The SCHEME_LOOKUP option or argument specifies that the match must be exact, case-insensitive, or consist of a match between the match codes of the input value and the match codes of the DATA values in the scheme. When a match occurs, any adjacent blank spaces in the transformation value are replaced with single blank spaces, then the value is written into the output data set. If no match is found for an input value, that exact value is written into the output data set.

You can specify the MODE argument or the MODE= option to apply schemes in one of two modes: *phrase* or *element*. Applying a scheme by phrase compares the entire input value (or the match code of the entire value) to the values (or match codes) in the scheme. Phrase is the default scheme apply mode.

When you apply a scheme by element, each element in the input value (or match code of each element) is compared to the values (or match codes) in the scheme. Applying schemes by element allows you to change one or more elements in an input value, without changing any of the other elements in that value.

The file format of a scheme is important when that scheme is applied. In the z/OS operating environment, schemes must be created and applied in SAS format. Schemes that are stored in a PDS in BFD format can be applied, and schemes in BFD format can be converted to SAS format using the CONVERT statement in the DQSCHEME procedure.

Note: Schemes in BFD format cannot be created or displayed in the z/OS operating environment. △

About Meta Options

Meta options are stored in the scheme when the scheme is created; they provide default values for certain options of the APPLY statement of the DQSCHEME procedure or default arguments for the DQSCHEMEAPPLY function or CALL routine. Default values are stored for the lookup mode (SCHEME_LOOKUP option or argument), apply mode (MODE option or argument), match definition, and sensitivity level. The values of the meta options are superseded when other values are specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine.

When the scheme is applied, the meta options for the match definition and sensitivity value are valid only when the scheme is applied with match-code lookup (when SCHEME_LOOKUP= USE_MATCHDEF).

The meta options are stored differently depending on the scheme format. For schemes in SAS format, the meta options are stored in the data set label. For schemes in BFD format, the meta options are stored within the scheme itself.

Note: In programs that create schemes in SAS format, do not specify a data set label; doing so deletes the meta options. △

The meta options are stored using the following syntax:

"*lookup-method*" "*apply-mode*" "*sensitivity-level*" "*match-definition*"

lookup-method

Valid values are:

EM	specifies that the default value of the SCHEME_LOOKUP option or argument is EXACT. To transform an input value, that value must exactly match a DATA value in the scheme.
IC	SCHEME_LOOKUP=IGNORE_CASE
UM	SCHEME_LOOKUP=USE_MATCHDEF. Match codes are created and compared for all input values and all DATA values in the scheme.

apply-mode
Valid values are:

E	specifies that the default value of the MODE option or argument is ELEMENT.
P	MODE=PHRASE

sensitivity-level
specifies the amount of information in the match codes that are generated when SCHEME_LOOKUP=USE_MATCHDEF. Valid values range from 50 to 95.

match-definition
specifies the name of the default match definition that will be used when the value of the SCHEME_LOOKUP option or argument is USE_MATCHDEF. Available match definitions are specified in the Locales chapter (see Chapter 7, "Locales," on page 77).

For example, the following meta options string specifies that, by default, the scheme uses match code lookup, applied by phrase, using the NAME match definition and a sensitivity level of 80:

```
"UM" "P" "80" "NAME"
```

Creating Match Codes

Match codes are encoded representations of input character variables that are used to cluster and cleanse input data. Match codes are created by the DQMATCH procedure and by the functions DQMATCH and DQMATCHPARSED.

The DQMATCH procedure creates an output data set that contains match codes and cluster numbers for one or more input character variables. Blank values can be removed from the output data set, or they can receive their own cluster number.

The functions DQMATCH and DQMATCHPARSED return one match code for one input character variable.

Note that match codes are created internally by the DQSCHEME procedure, the DQSCHEMEAPPLY function, and the DQSCHEMEAPPLY CALL routine. These match codes are used in the process of creating or applying a scheme, as described in "Transforming Data with Schemes" on page 6.

How Match Codes Are Created

You can create two types of match codes:

- *Simple match codes* are created from a single input character variable.
- *Compound match codes* consist of a concatenation of match codes from two or more input character variables. Match codes are created for each value in each variable. Then the separate match codes are concatenated into a compound match code. You have the option of specifying that a delimiter, in the form of an exclamation point (!), is to be inserted between the simple match codes that comprise the combined match code (via the DELIMITER option or argument).

To create simple match codes, you specify one CRITERIA statement, with one input variable identified in the VAR= option and one output variable identified with the MATCHCODE= option. Compound match codes are similar, except that you specify multiple CRITERIA statements for multiple variables, and all of those CRITERIA statements specify the same output variable in their respective MATCHCODE= options.

The SAS Data Quality Server software creates match codes using these general steps:

1 Parse the input character value to identify tokens.

2 Remove insignificant words.

3 Remove some of the vowels. Remove fewer vowels when a scheme-build match definition has been specified, as described in “About the Scheme Build Match Definitions” on page 13.

4 Standardize the format and capitalization of words.

5 Create the match code by extracting the appropriate amount of information from one or more tokens, based on the specified match definition and level of sensitivity.

Certain match definitions skip some of these steps.

Note: When you work with two or more data sets that you intend to analyze together or join using match codes, be sure to use identical sensitivities and match definitions when you create the match codes in each data set. △

About the Length of Match Codes

Match codes can vary in length between 1 and 255 characters. The length is determined by the specified match definition. If you receive a message in the SAS log that states that match codes have been truncated, you should extend the length of your match code variable. Truncated match codes will not produce accurate results.

About Clusters

Clusters are groups of differing input character values that have identical match codes. The number of clusters, and the number of values in each cluster, is determined by the match definition and the sensitivity value that were specified when the match codes were created, as described in “About Sensitivity” on page 11.

Clusters are used in the creation of schemes using the DQSCHEME procedure, as described in “Creating Schemes” on page 6.

In the DQMATCH procedure, cluster numbers are optionally delivered to the output data set when match codes are created. You can specify the following options in the PROC DQMATCH statement to configure the output data set based on clusters.

CLUSTER=
: specifies the name of the output variable that contains cluster numbers.

CLUSTERS_ONLY
: limits match code output to values that received a cluster number.

CLUSTER_BLANKS
: assigns a cluster number to blank values.

As an example of the use of the clustering options in the DQMATCH procedure, assume that a data cleansing program specifies the options CLUSTERS_ONLY and NO_CLUSTER_BLANKS. The resulting output would not contain input character values that were blank. Nor would the output data set contain input character values that did not receive a cluster number. In this case, a cluster number is assigned only when two or more input values have the same match code. When you specify CLUSTERS_ONLY, non-blank input character values are included in the output data set only when their match codes are held in common with at least one other input character value.

The output data set that is generated by the DQMATCH procedure is displayed by default in the Output window. The output data set contains the applicable values from the specified input character variable, along with the match codes that were created for those values.

For information on how match codes are created, see “How Match Codes Are Created” on page 9.

About Sensitivity

The amount of information contained in match codes is determined by a specified sensitivity level. Changing the sensitivity level allows you to change what is considered a match. Match codes that are created at lower levels of sensitivity capture little information about the input values. The result is more matches, fewer clusters (see “About Clusters” on page 10), and more values in each cluster. At higher sensitivity levels, input values must be more similar to receive the same match code. Clusters are more numerous, and the number of entries in each cluster is smaller.

In some data cleansing jobs, a lower sensitivity value is needed. For example, if you wanted to transform the following names to a single consistent value using a scheme, you would need to specify a lower sensitivity level:

```
Patricia J. Fielding
Patty Fielding
Patricia Feelding
Patty Fielding
```

In this, all four values would be assigned to the same cluster and would be transformed to the most-common value, **`Patty Fielding`**.

In other cases, a higher sensitivity level is needed. For example, if you were collecting customer data based on account numbers, you would want to cluster on individual account numbers. A high sensitivity value would be needed.

In the SAS Data Quality Server software, sensitivity values range from 50 to 95, and the default value is 85.

To arrive at the sensitivity level that fits your data and your application, run tests with DQMATCH or create analysis data sets with PROC DQSCHEME.

About Locale Definitions

Using Parse Definitions

Parse definitions are referenced when you want to create parsed input values. Parsed input values are delimited so that elements in the those values can be associated with named tokens. After parsing, specific contents of the input values can be returned by specifying the names of tokens.

Parse definitions and tokens are referenced by the following functions:

- “DQPARSE Function” on page 59.
- “DQPARSEINFOGET Function” on page 60.
- “DQTOKEN Function” on page 74.
- “DQPARSETOKENGET Function” on page 61.

- "DQPARSETOKENPUT Function" on page 63

For a brief example of how tokens are assigned and used, see "About the Quality Knowledge Base" on page 3.

Parsing a character value assigns tokens only when the content in the input value meets the criteria in the parse definition. Parsed character values can therefore contain empty tokens. For example, three tokens are empty when you use the DQPARSE function to parse the character value **`Ian M. Banks`**, using the NAME parse definition in the ENUSA locale. The resulting token/value pairs are as follows:

NAME PREFIX	empty
GIVEN NAME	Ian
MIDDLE NAME	M.
FAMILY NAME	Banks
NAME SUFFIX	empty
NAME APPENDAGE	empty

Note: For parse definitions that work with dates, such as DATE (DMY) in the ENUSA locale, input values must be character data rather than SAS dates. △

About the Global Parse Definitions

Global parse definitions contain a standard set of parse tokens that enable the analysis of similar data from different locales. For example, the ENUSA locale and the DEDEU locale both contain the parse definition ADDRESS (GLOBAL). The parse tokens are the same in both locales. This global parse definition enables the combination of parsed character data from multiple locales.

All global parse definitions are identified by the (GLOBAL) suffix.

Using Match Definitions

Match definitions are referenced during the creation of match codes. Match codes provide a variable method of clustering similar input values as a basis for data cleansing jobs such as the application of schemes.

When you create match codes, you determine the number of clusters (values with the same match code) and the number of members in each cluster by specifying a sensitivity level. The default sensitivity level is specified by the procedure or function, rather than the match definition. For information on sensitivity levels, see "About Sensitivity" on page 11.

Match definitions are referenced by the following procedures and functions:

- Chapter 3, "The DQMATCH Procedure," on page 19.
- Chapter 4, "The DQSCHEME Procedure," on page 29.
- "DQMATCH Function" on page 56.
- "DQMATCHINFOGET Function" on page 57.
- "DQMATCHPARSED Function" on page 58.

When you create match codes for parsed character values, your choice of match definition depends on the parse definition that was used to parse the input character value. To determine the parse definition that is associated with a given match definition, use the "DQMATCHINFOGET Function" on page 57.

Note: For match definitions that work with dates, such as DATE (MDY) in the ENUSA locale, input values must be character data rather than SAS dates. △

About the Scheme Build Match Definitions

Locales contain certain match definitions that are recommended for use in the DQSCHEME procedure because they produce more desirable schemes. The names of these scheme-build match definitions always end with “(SCHEME BUILD)”.

Scheme-build match definitions are advantageous because they create match codes that contain more vowels. Match codes that contain more vowels result in more clusters with fewer members in each cluster, which in turn results in a larger, more specific set of transformation values.

When you are using the DQMATCH procedure or function to create simple clusters, it is better to have fewer vowels in the match code. For example, when using the CITY match definition in PROC DQMATCH, the values Baltimore and Boltimore receive the same match codes. The match codes would differ if you used the match definition CITY (SCHEME BUILD).

Using Case and Standardization Definitions

Case and standardization definitions are applied to character values to make them more consistent for the purposes of display or in preparation for transforming those values with a scheme.

Case definitions are referenced by the “DQCASE Function” on page 48. Standardization definitions are referenced by the “DQSTANDARDIZE Function” on page 73.

Case definitions transform the capitalization of character values. For example, the case definition Proper in the ENUSA locale takes as input any general text, capitalizes the first letter of each word, and uses lowercase for the other letters in the word, while recognizing and retaining or transforming various words and abbreviations into uppercase. Other case definitions, such as PROPER – ADDRESS, apply to specific text content.

Standardization definitions standardize the appearance of specific data values. In general, words are capitalized appropriately based on the content of the input character values. Also, adjacent blank spaces are removed, along with unnecessary punctuation. Additional standardizations may be made for specific content. For example, the standardization definition STATE (FULL NAME) in the locale ENUSA converts abbreviated state names to full names in uppercase.

About the Standardization of Dates in the EN Locale

In the EN locale, dates are standardized to two-digit days (00–31), two-digit months (01–12), and four-digit years (such as 2003). Input dates must be character values rather than SAS dates. Spaces separate (delimit) the days, months, and years, as shown in the following table.

Table 2.1 Examples of Date Standardizations

Input Date	Standardization Definition	Standardized Date
July04, 03	Date (MDY)	07 04 2003
July 04 04	Date (MDY)	07 04 1904

July0401	Date (MDY)	07 04 2001
04.07.02	Date (DMY)	04 07 2002
04-07-2004	Date (DMY)	04 07 2004
03/07/04	Date (YMD)	2003 07 04

Two-digit year values are standardized as follows. If an input year is greater than 00 and less than or equal to 03, the standardized year will be 2000, 2001, 2002, or 2003. Two-digit input year values that are greater than or equal to 04 and less than or equal to 99 will be standardized into the range of 1904–1999. For example, an input year of 03 is standardized as 2003. An input year of 04 is standardized as 1904. These standardizations are not affected by the value of the SAS system option YEARCUTOFF=.

Using Gender Analysis, Locale Guess, and Identification Definitions

Gender analysis, locale guess, and identification definitions enable you make determinations about character values. With these definitions you can determine:

- The gender of an individual based on a name value.
- The locale that is the most suitable for a given character value.
- The category of a value, which is chosen from a set of available categories.

Gender analysis definitions determine the gender of an individual based on that individual's name. The gender is determined to be unknown if the first name is used by both males and females, if no other clues are provided in the name or if conflicting clues are found. Gender analysis definitions are referenced by the "DQGENDER Function" on page 49.

Locale guess definitions allow the software to determine the locale that is most likely represented by a character value. All locales that are loaded into memory as part of the locale list are considered, but only if they contain the specified guess definition. If a definite locale determination cannot be made, the chosen locale is the first locale in the locale list. Locale guess definitions are referenced by the "DQLOCALEGUESS Function" on page 53.

Identification definitions are used to categorize character values. For example, using the Entity identification definition in the ENUSA locale, a name value can be determined to apply to an individual or an organization. Identification definitions are referenced by the "DQIDENTIFY Function" on page 52.

Using Pattern Analysis Definitions

Pattern analysis definitions enable you to determine if an input character value contains characters that are alphabetic, numeric, non-alphanumeric (punctuation marks or symbols), or a mixture of alphanumeric and non-alphanumeric. The ENUSA locale contains two pattern analysis definitions. The pattern analysis definition WORD is referenced by the "DQPATTERN Function" on page 64 to generate one character of analytical information for each word in the input character value. The CHARACTER definition generates one character of analytical information for each character in the input character value.

Using the SAS Data Quality Server Functions

The SAS Data Quality Server software provides a number of functions and a CALL routine that enable you to parse input values, return tokens, create match codes, analyze values, and return information on character values and locales. The functions are divided into the following categories:

- Parsing, see "About the Parsing Functions" on page 15.
- Creating Match Codes, see "About the Match Code Functions" on page 15.
- Applying Schemes, see "About the Scheme-Apply Functions" on page 15.
- Standardizing, see "About the Standardization Functions" on page 16.
- Gender Analysis, Locale Guessing, and Identifying, see "About the Gender Analysis, Locale Guessing, and Identification Functions" on page 16.
- Casing, see "About the Case Functions" on page 16.
- Analyzing Patterns, see "About the Pattern Analysis Functions" on page 16.
- Reporting, see "About the Reporting Functions" on page 17.

About the Parsing Functions

The parsing functions (see "Parsing Functions" on page 47) identify tokens in input character values, insert elements into input values based on token names, return elements based on token names, and return information on parse definitions. With these functions, you can assemble a parsed value from separate elements or create a match code for a single element in a character value.

About the Match Code Functions

The match code functions (see "Matching Functions" on page 47) return match codes or provide information that is pertinent to the creation of match codes. For information on the creation and use of match codes, see "Creating Match Codes" on page 9.

The DQMATCH and DQMATCHPARSED functions have an optional sensitivity argument that determines the amount of information in the resulting match code. Lower sensitivity levels generate fewer clusters, with more members in each cluster. Because higher sensitivities generate match codes with more information, input values must be more similar to generate the same match code. Resulting clusters are higher in number, with fewer members per cluster.

About the Scheme-Apply Functions

The scheme apply functions (see "Scheme Apply Functions and CALL Routines" on page 48) enable you to apply existing schemes to input character values. Each application seeks to apply a transformation value to one input character value. For information on schemes, see "Applying Schemes" on page 7.

In a scheme apply function, you specify the location of the scheme, the scheme format (SAS or BFD), the name of the variable that is to be transformed, and whether you want to apply the scheme to the entire input value or to elements in the input value. The return value is the transformed version of the input value. In addition to the transformed value, the CALL routine also returns the number of transformations that were performed on the input value.

You can create and store a scheme in a SAS data set or in file that uses BFD format. The BFD format allows schemes to be shared between SAS and the dfPower Studio software from DataFlux (a SAS company). In the z/OS operating environment, all schemes must be created and applied in SAS format.

About the Standardization Functions

The standardization functions (see "Standardization Functions" on page 48) allow you to standardize the case, punctuation, format, and content of character data. The functions return standardized values based on the specified standardization definition in the specified locale. For example, if the standardization definition ADDRESS in the ENUSA locale is specified for the DQSTANDARDIZE function, the return value contains appropriate uppercase letters, with all insignificant blank spaces and punctuation removed, and with certain recognized words changed to a standardized representation. Another group of standardization definitions are used to generate standardized dates in various combinations of year, month, and day.

About the Case Functions

The case functions (see "Case Functions" on page 47) allow you to transform the case and punctuation of your data, using uppercase and lowercase, as determined by the specified case definition in the specified locale. For example, in the ENUSA locale, if the DQCASE function references the case definition PROPER, the return value will contain capitalized letters at the start of each word, and lowercase letters for the rest. Other case definitions, such as PROPER – NAME, are used for specific content.

About the Gender Analysis, Locale Guessing, and Identification Functions

The gender analysis, locale guessing, and identification functions (see "Gender Analysis, Locale Guessing, and Identification Functions" on page 47) return information that is determined from the content of an input character value.

The DQGENDER function (see "DQGENDER Function" on page 49) determines gender based on the name of an individual. The function can enhance data analysis activities, such as customer analysis by gender, and also mass-mailing activities, such as customizing the title of the individual and the salutation in correspondence.

The DQLOCALEGUESS function (see "DQLOCALEGUESS Function" on page 53) determines the locale that is most likely represented by a character value. The function returns the name of the locale that best fits the character value. To be considered for selection, the locale must be loaded into memory as part of the locale list, and the locale must contain the specified guess definition. If more than one locale is selected, the function returns the name of the locale that appears first in the locale list.

The DQIDENTIFY function (see "DQIDENTIFY Function" on page 52) assigns a category to a character value, based on an identification definition in a specified locale. For example, the DQIDENTIFY function returns either INDIVIDUAL, ORGANIZATION, or UNKNOWN when a name is analyzed using the INDIVIDUAL/ORGANIZATION definition in the ENUSA locale.

About the Pattern Analysis Functions

The DQPATTERN function (see "DQPATTERN Function" on page 64) returns information on the presence of numeric, alphabetic, and non-alphanumeric characters in input character values. Pattern analysis definitions enable you to analyze by word or character.

About the Reporting Functions

The reporting functions (see “Reporting Functions” on page 48) return information on the locales, including the names of the locales in the locale list that is loaded into memory. For specified locales, you can return a list of definition names, token names for specified parse definitions, and parse definition names that are related to specified guess or match definitions.

CHAPTER 3 The DQMATCH Procedure

The DQMATCH Procedure in SAS Data Quality Server

The DQMATCH procedure creates match codes in an output data set for specified input character variables. The output optionally includes cluster numbers for values that have identical match codes. Missing values can be removed from the output data set or they can be retained and be given a cluster number.

Match codes are created based on a specified match definition in a specified locale. A specified sensitivity level determines the amount of information in the match codes. The amount of information in the match codes determines the number of clusters and the number of members in each cluster. Higher sensitivity levels produce fewer clusters, with fewer members per cluster. Use higher sensitivities when you need matches that are more exact. Use lower sensitivities to sort data into general categories or to capture all values that use different spelling to convey the same information.

For further usage information, see “Creating Match Codes” on page 9.

DQMATCH Procedure Syntax

Requirements: At least one CRITERIA statement is required. See “CRITERIA Statement” on page 21.

See Also:

- “DQMATCH Examples” on page 22.
- “Creating Match Codes” on page 9.
- “About Clusters” on page 10.

PROC DQMATCH
<**DATA**=*input-data-set*>
<**DELIMITER** | **NODELIMITER**>

<**CLUSTER**=*output-variable-name*>
<**CLUSTER_BLANKS** | **NO_CLUSTER_BLANKS**>
<**CLUSTERS_ONLY**>
<**LOCALE**=*locale-name*>
<**MATCHCODE**=*output-variable-name*>
<**OUT**=*data-set-name*>;

CRITERIA *options*;

The DQMATCH procedure enables the following options:

DATA=*input-data-set*
identifies the input SAS data set. The default input data set is the most recently created data set in the current SAS session.

CLUSTER=*output-variable-name*
specifies the name of the numeric variable in the output data set that contains the cluster number. If the CLUSTER= option is not specified and if the CLUSTERS_ONLY option is specified, then an output variable named CLUSTER is created.

CLUSTER_BLANKS | NO_CLUSTER_BLANKS
specifying the default value CLUSTER_BLANKS writes blank values into the output data set, without an accompanying match code. Specifying NO_CLUSTER_BLANKS removes blank values from the output data set.

CLUSTERS_ONLY
excludes from the output data set any input character values that are not found to be part of a cluster. A cluster number is assigned only when two or more input values produce the same match code. Specifying CLUSTERS_ONLY excludes input character values that have unique match codes and are not blank. This option is not asserted by default, so normally, all input values are included in the output data set.

DELIMITER | NODELIMITER
when multiple CRITERIA statements are specified, the default value DELIMITER specifies that exclamation points (!) separate the individual match codes that make up the concatenated match code. Match codes are concatenated in the order of appearance of CRITERIA statements in the DQMATCH procedure.

The NODELIMITER option specifies that multiple match codes are concatenated without the exclamation points.

Note: The default in SAS differs from the default in the dfPower Studio software from DataFlux (a SAS company). SAS uses a delimiter by default; DataFlux does not. Be sure to use delimiters consistently if you plan to analyze, compare, or combine match codes created in SAS and dfPower Studio. △

LOCALE=*locale-name*
optionally specifies the locale that will be used to create match codes. The value can be a locale name in quotation marks or the name of a variable whose value is a locale name or is an expression that evaluates to a locale name.

The specified locale must be loaded into memory as part of the locale list (see “About Locales” on page 4). If no value is specified, the default locale is used. The default locale is the first locale in the locale list.

Note that the match definition, which is part of the specified locale, is specified in the CRITERIA statement. This specification allows different match definitions to be applied to different variables in the same procedure.

MATCHCODE=*output-variable*
specifies a name for the output character variable that stores the match codes. The DQMATCH procedure defines a sufficient length for this variable, even if a variable with the same name already exists in the input data set.

A default match code variable named MATCH_CD is generated if the following statements are all true:

- No value is specified for the MATCHCODE= option in the PROC DQMATCH statement, and no values are specified for the MATCHCODE= option in subsequent CRITERIA statements.
- No value is specified for the CLUSTER= option.
- No value is specified for the CLUSTERS_ONLY option.

If the MATCHCODE= option is not specified in the PROC DQMATCH or in any CRITERIA statements, and if CLUSTERS= or CLUSTERS_ONLY is specified, then no match code output variable is created and no match codes are written into the output data set.

For further information on match codes, see "Creating Match Codes" on page 9.

OUT=*data-set-name*
specifies the name of the output data set. If the specified data set does not exist, PROC DQMATCH creates it. The default output data set is the input data set.

CRITERIA Statement

Creates a match code for an input variable.

Requirement: At least one CRITERIA statement is required in a DQMATCH procedure step.

CRITERIA
DELIMSTR=*parsed–input-variable* | **VAR**=*input-variable*
MATCHDEF=*match-definition*
<**SENSITIVITY**=*information-value*>
<**MATCHCODE**=*output-variable*>;

DELIMSTR= *parsed-input-variable*
specifies the name of a variable that has been parsed by the DQPARSE function, or contains tokens that were added with the DQPARSETOKENPUT function.

You cannot specify the DELIMSTR= option and the VAR= option in the same CRITERIA statement.

MATCHCODE=*output-variable*
optionally specifies the name of the variable that receives the match codes for the character variable that is specified in the VAR= or DELIMSTR= option.

In the CRITERIA statement, the value of the MATCHCODE= option is not valid if you also specify the MATCHCODE= option in the PROC DQMATCH statement.

If you are using multiple CRITERIA statements in a single procedure step, you must either specify the MATCHCODE= variable in each CRITERIA statement, or generate compound match codes by specifying the MATCHCODE= option only in the PROC DQMATCH statement.

MATCHDEF=*match-definition*
defines the match definition that will be used to create the match code for the specified variable. The match definition must exist in the locale that is specified in the LOCALE= option of the PROC DQMATCH statement. For information on match definitions, see the "ENUSA" on page 101.

SENSITIVITY=*information-value*
optionally determines the information of the resulting match codes. Higher sensitivity values create match codes that contain more information about the input values. Higher sensitivity levels result in a greater number of clusters, with fewer values in each cluster. Valid values range from 50 to 95. The default value is 85.

VAR=*input-variable*
specifies the name of the character variable that will be used to create match codes. The values of this variable cannot contain delimiters that were added with the functions DQPARSE or DQPARSETOKENPUT. If the variable contains delimited values, use the DELIMSTR= option instead of the VAR= option.

Details

Match codes are created for the input variables that are specified in each CRITERIA statement. The resulting match codes are stored in the output variables that are named in the MATCHCODE= option. The MATCHCODE= option can be specified in the PROC DQMATCH statement or the CRITERIA statement.

Simple match codes are created when the CRITERIA statements specify different values for their respective MATCHCODE= options. Compound match codes are created when two or more CRITERIA statements specify the same value for their respective MATCHCODE= options.

To create match codes for a parsed character variable, specify the DELIMSTR= option instead of the VAR= option. Then be sure to specify in the MATCHDEF= option the name of the match definition that is associated with the parse definition that was used to add delimiters to the character variable. To determine the parse definition that is associated with a match definition, use the DQMATCHINFOGET function (see "DQMATCHINFOGET Function" on page 57).

DQMATCH Examples

Example 1 on page 22.
Example 2 on page 24.
Example 3 on page 25.
Example 4 on page 26.

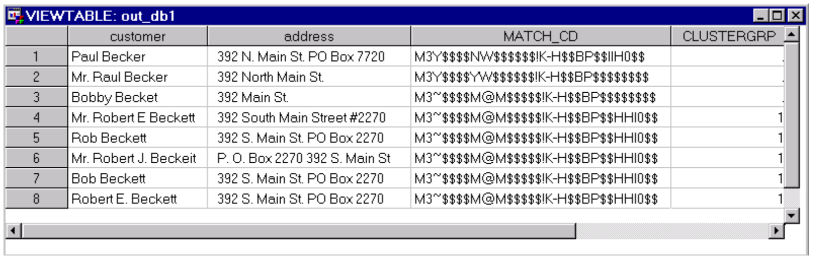
Example 1: Matching Values Using the Default Sensitivity

The following example uses the DQMATCH procedure to create match codes and cluster numbers. The default sensitivity level of 85 is used in both CRITERIA statements. The locale ENUSA is assumed to have been loaded into memory previously with the %DQLOAD AUTOCALL macro.

```
/* Create the input data set. */
data cust_db;
   length customer $ 22;
   length address $ 31;
   input customer $char22. address $char31.;

datalines;
Bob Beckett             392 S. Main St. PO Box 2270
Robert E. Beckett       392 S. Main St. PO Box 2270
Rob Beckett             392 S. Main St. PO Box 2270
Paul Becker             392 N. Main St. PO Box 7720
Bobby Becket            392 Main St.
Mr. Robert J. Beckeit   P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett    392 South Main Street #2270
Mr. Raul Becker         392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db1 matchcode=match_cd
   cluster=clustergrp locale='ENUSA';
   criteria matchdef='Name' var=customer;
   criteria matchdef='Address' var=address;
run;

/* Print the results. */
proc print data=out_db1;
run;
```

The PROC PRINT output is as follows.

Display 3.1 PROC DQMATCH Output



VIEWTABLE: out_db1

	customer	address	MATCH_CD	CLUSTERGRP
1	Paul Becker	392 N. Main St. PO Box 7720	M3Y$$$$NW$$$$$$!K-H$$BP$$IIH0$$	.
2	Mr. Raul Becker	392 North Main St.	M3Y$$$$YW$$$$$$!K-H$$BP$$$$$$$$	.
3	Bobby Becket	392 Main St.	M3~$$$$M@M$$$$$!K-H$$BP$$$$$$$$	.
4	Mr. Robert E Beckett	392 South Main Street #2270	M3~$$$$M@M$$$$$!K-H$$BP$$HHI0$$	1
5	Rob Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-H$$BP$$HHI0$$	1
6	Mr. Robert J. Beckeit	P. O. Box 2270 392 S. Main St	M3~$$$$M@M$$$$$!K-H$$BP$$HHI0$$	1
7	Bob Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-H$$BP$$HHI0$$	1
8	Robert E. Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-H$$BP$$HHI0$$	1

The output data set, OUT_DB1, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code is composed of two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that five of the character values are grouped in a single cluster and that the other three are not part of a cluster. The clustering is based on the values of the MATCH_CD variable. By looking at the values for MATCH_CD, you can see that five character values have identical match code values. Although the match code value for customer **`Bobby Becket`** is similar to the Cluster 1 match codes, the difference in the address caused it not to be included in Cluster 1.

Example 2 on page 24 shows how the use of non-default sensitivities increases the accuracy of the analysis.

This example is available in the SAS Sample Library under the name DQMCDFLT.

Example 2: Matching Values Using Mixed Sensitivities

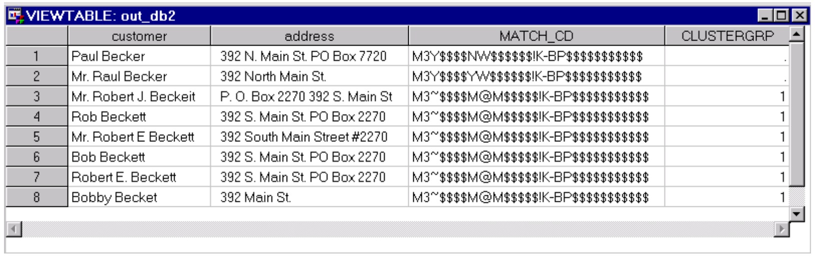
The following example is similar to Example 1 on page 22, in that it displays match codes and clusters for a simple data set. This example differs in that the CRITERIA statement for the ADDRESS variable uses a sensitivity of 50. The CRITERIA statement for the NAME variable uses the same default sensitivity of 85.

The use of mixed sensitivities allows you to tailor your clusters for maximum accuracy. In this case, clustering accuracy is increased when the sensitivity level of a less-important variable is decreased.

This example primarily shows how to identify possible duplicate customers based on their name but using minimal sensitivity for their address to avoid false matches.

```
/* Create the input data set. */
data cust_db;
   length customer $ 22;
   length address $ 31;
   input customer $char22. address $char31.;

datalines;
Bob Beckett            392 S. Main St. PO Box 2270
Robert E. Beckett      392 S. Main St. PO Box 2270
Rob Beckett            392 S. Main St. PO Box 2270
Paul Becker            392 N. Main St. PO Box 7720
Bobby Becket           392 Main St.
Mr. Robert J. Beckeit  P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett   392 South Main Street #2270
Mr. Raul Becker        392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db2 matchcode=match_cd
cluster=clustergrp locale='ENUSA';
   criteria matchdef='Name' var=customer;
   criteria matchdef='Address' var=address sensitivity=50;
run;

/* Print the results. */
proc print data=out_db2;
run;
```

The PROC PRINT output is as follows.



VIEWTABLE: out_db2

	customer	address	MATCH_CD	CLUSTERGRP
1	Paul Becker	392 N. Main St. PO Box 7720	M3Y$$$$NW$$$$$$!K-BP$$$$$$$$$$$	.
2	Mr. Raul Becker	392 North Main St.	M3Y$$$$YW$$$$$$!K-BP$$$$$$$$$$$	.
3	Mr. Robert J. Beckeit	P. O. Box 2270 392 S. Main St	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1
4	Rob Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1
5	Mr. Robert E Beckett	392 South Main Street #2270	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1
6	Bob Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1
7	Robert E. Beckett	392 S. Main St. PO Box 2270	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1
8	Bobby Becket	392 Main St.	M3~$$$$M@M$$$$$!K-BP$$$$$$$$$$$$	1

The output data set, OUT_DB2, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code is composed of two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that six of the character values are grouped in a single cluster and that the other two are not part of any cluster. The clustering is based on the values of the MATCH_CD variable. This result is different than in Example 1 on page 22, where only five values were clustered based on NAME and ADDRESS. This difference is caused by the lower sensitivity setting for the ADDRESS criteria in the current example, which makes the matching less sensitive to variations in the address field. Therefore, the value **`Bobby Becket`** has now been included in Cluster 1 because **`392 Main St.`** is considered a match with **`392 S. Main St. PO Box 2270`** and the other variations, while this was not true at a sensitivity of 85.

Example 2 on page 24 uses minimum sensitivity levels to assign all values to clusters.

This example is available in the SAS Sample Library under the name DQMCMIXD.

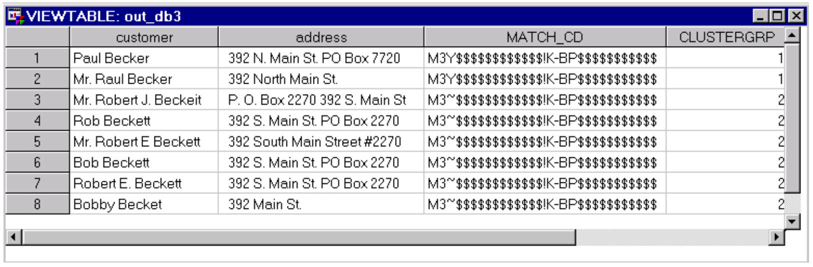
Example 3: Matching Values Using Minimal Sensitivity

The following example shows how minimal sensitivity levels can generate inaccurate clusters. A sensitivity of 50 is used in both CRITERIA statements, which is the minimum value for this argument.

```
/* Create the input data set. */
data cust_db;
   length customer $ 22;
   length address $ 31;
   input customer $char22. address $char31.;

datalines;
Bob Beckett             392 S. Main St. PO Box 2270
Robert E. Beckett       392 S. Main St. PO Box 2270
Rob Beckett             392 S. Main St. PO Box 2270
Paul Becker             392 N. Main St. PO Box 7720
Bobby Becket            392 Main St.
Mr. Robert J. Beckeit   P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett    392 South Main Street #2270
Mr. Raul Becker         392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
```

```
proc dqmatch data=cust_db out=out_db3 matchcode=match_cd
   cluster=clustergrp locale='ENUSA';
   criteria matchdef='Name' var=customer sensitivity=50;
   criteria matchdef='Address' var=address sensitivity=50;
run;

/* Print the results. */
proc print data=out_db3;
run;
```

The PROC PRINT output is as follows.


VIEWTABLE: out_db3

	customer	address	MATCH_CD	CLUSTERGRP
1	Paul Becker	392 N. Main St. PO Box 7720	M3Y$$$$$$$$$$$$!K-BP$$$$$$$$$$$	1
2	Mr. Raul Becker	392 North Main St.	M3Y$$$$$$$$$$$$!K-BP$$$$$$$$$$$	1
3	Mr. Robert J. Beckeit	P. O. Box 2270 392 S. Main St	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2
4	Rob Beckett	392 S. Main St. PO Box 2270	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2
5	Mr. Robert E Beckett	392 South Main Street #2270	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2
6	Bob Beckett	392 S. Main St. PO Box 2270	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2
7	Robert E. Beckett	392 S. Main St. PO Box 2270	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2
8	Bobby Becket	392 Main St.	M3~$$$$$$$$$$$$!K-BP$$$$$$$$$$$	2

The output data set OUT_DB3 includes the variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code is composed of two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that six of the values are grouped in one cluster and that the other two are grouped in another. The clustering is based on the values of the MATCH_CD variable. This example shows that, with a minimal sensitivity level of 50, the following values match and form a cluster.

```
Mr. Raul Beckett
Paul Becker
```

A higher sensitivity level would not cluster these observations.

This example is available in the SAS Sample Library under the name DQMCMIN.

Example 4: Creating Match Codes for Parsed Values

The following example creates match codes for parsed character data. The program loads locales, determines a parse definition, creates character elements, creates parsed character values, and creates match codes for the parse character elements.

```
/* load locales */
%dqload(dqlocale=(enusa), dqsetuploc="your-dqsetup-file-here")

/* Determine the parse definition associated with your */
/* match definition.                                   */
data _null_;
   parsedefn=dqMatchInfoGet('Name');
   call symput('parsedefn', parsedefn);
   put 'The parse definition for the NAME match definition is: ' parsedefn;
   tokens=dqParseInfoGet(parsedefn);
   put 'The ' parsedefn 'parse definition tokens are:' / @5 tokens;
run;

/* Create variables containing name elements. */
data parsed;
   length first last $ 20;
   first='Scott'; last='James';  output;
   first='James'; last='Scott';  output;
   first='Ernie'; last='Hunt';   output;
   first='Brady'; last='Baker';  output;
   first='Ben';   last='Riedel'; output;
   first='Sara';  last='Fowler'; output;
   first='Homer'; last='Webb';   output;
   first='Poe';   last='Smith';  output;
run;

/* Create parsed character values. */
data parsedview;
   set parsed;
   length delimstr $ 100;

   * Insert one token at a time;
   delimstr=dqParseTokenPut(delimstr, first, 'Given Name',  'Name');
   delimstr=dqParseTokenPut(delimstr, last,  'Family Name', 'Name');
run;

/* Generate match codes using the parsed character values. */
proc dqmatch data=parsedview
             out=mcodes;
   criteria matchdef='Name' delimstr=delimstr sensitivity=85;
run;

/* Print the match codes. */
proc print data=mcodes;
   title 'Look at the match codes from PROC DQMATCH';
run;
```

This example is available in the SAS Sample Library under the name DQMCPARS.

CHAPTER 4
The DQSCHEME Procedure

The DQSCHEME Procedure

The DQSCHEME procedure creates scheme data sets and analysis data sets and applies schemes to input data sets. You can also apply schemes with the DQSCHEMEAPPLY function or CALL routine (see “DQSCHEMEAPPLY CALL Routine” on page 65).

For further information on using the DQSCHEME procedure, see “Transforming Data with Schemes” on page 6.

DQSCHEME Procedure Syntax

Tips:

- □ CREATE and APPLY statements can appear in any order in the DQSCHEME procedure.
- □ All CREATE statements are processed before all APPLY statements regardless of their order of appearance.
- □ All CONVERT statements are processed last, regardless of their order of appearance in the procedure.

See also: “Transforming Data with Schemes” on page 6.

PROC DQSCHEME
<**DATA**=*input-data-set*>
<**BFD** | **NOBFD**>
<**OUT**=*output-data-set*>;

<**CREATE** *options*;>
<**APPLY** *options*;>
<**CONVERT** *options*;>

The DQSCHEME procedure enables the following options

BFD | NOBFD
specifying BFD indicates that all Blue Fusion Data that are created or applied in the current procedure step are in BFD format. Specifying NOBFD indicates that all schemes are in SAS format. The default value is BFD.

CAUTION:
Always specify NOBFD when creating schemes in the z/OS operating environment. Schemes in BFD format can be applied but not created. △

The DQSCHEME procedure can create and apply schemes in either format. Schemes in BFD format can be edited using the feature-rich graphical user interface of the dfPower Studio software from DataFlux (a SAS company).

Note: In schemes using SAS format, data set labels are used to store meta options (see "About Meta Options" on page 8). Therefore, you should not specify data set labels in scheme data sets that are stored in SAS format. Doing so overwrites the scheme metadata. △

DATA=*input-SAS-data-set*
identifies the SAS data set from which one or more schemes are built, when using the CREATE statement. When using the APPLY statement to apply existing schemes, the DATA= option specifies the data set against which the schemes are applied.

The default data set is the most recently created data set in the current SAS session.

OUT=*output-data-set*
specifies the name of the output data set. If the specified data set does not exist, PROC DQSCHEME creates it. The default output data set is the input data set.

If you use an APPLY statement, you must specify the OUT= option. If you use multiple APPLY statements, the results are written into the output data set after all schemes have been applied.

CREATE Statement

Creates a scheme or an analysis data set.

See also: "Applying Schemes" on page 7.

CREATE
ANALYSIS=*output-data-set*
SCHEME=*scheme-name*
VAR=*variable-name*
MATCHDEF=*match-definition*
<**SENSITIVITY**=*complexity-factor*>
<**LOCALE**=*locale-name*>
<**SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF**>

<MODE=PHRASE | ELEMENT>
<INCLUDE_ALL>;

ANALYSIS=*output-data-set*
names the output data set that will store analytical data, as described in "Creating Schemes" on page 6. This option is required if the SCHEME= option is not specified.

INCLUDE_ALL
specifies that the scheme is to contain all of the values of the input variable, including those with unique match codes that were not transformed and that did not receive a cluster number. INCLUDE_ALL is not specified by default.

LOCALE=*locale-name*
optionally specifies the locale that contains the specified match definition. The value can be a locale name in quotation marks or the name of a variable whose value is a locale name or is an expression that evaluates to a locale name.

The specified locale must be loaded into memory as part of the locale list (see "About Locales" on page 4). If no value is specified, the default locale is used. The default locale is the first locale in the locale list.

MATCHDEF=*match-definition*
specifies the match definition in the specified locale that is used to establish cluster numbers. For the ENUSA locale, information on available match definitions is provided in "ENUSA" on page 101. For other locales, consult the documentation that was downloaded with the locale.

Although you can specify any valid match definition, you are strongly encouraged to use definitions whose names end in (SCHEME BUILD) when using the ENUSA locale. These match definitions yield optimal results in the DQSCHEME procedure, as discussed in "About the Scheme Build Match Definitions" on page 13.

The value of the MATCHEDF= option is stored in the scheme as a meta option, which provides a default match definition when the scheme is applied. This meta option is used at apply time only when SCHEME_LOOKUP=MATCHDEF. The default value that is supplied by this meta option is superseded by a match defintion that is specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine. For more information on meta options, see "About Meta Options" on page 8.

MODE=PHRASE | ELEMENT
specifies a default mode of scheme application. This information is stored in the scheme as metadata. The default mode that you specify here can be superseded when the scheme is applied, as described in "Applying Schemes" on page 7.

Valid values for the MODE= option are defined as follows:

PHRASE
this default value specifies that the entirety of each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the match code for the entire input value is compared to the match codes that are generated for each data value in the scheme.

ELEMENT
specifies that each element in each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the match code for each element is compared to the match codes that are generated for each element in each value of the DATA variable in the scheme.

The value of the MODE= option is stored in the scheme as a meta option, which specifies a default mode when the scheme is applied. The default value that is supplied by this meta option is superseded by a mode that is specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine. For more information on meta options, see "About Meta Options" on page 8.

SCHEME=*scheme-name*

specifies the name of the scheme that will be created. To create a scheme data set in Blue Fusion Data format, specify the BFD option in the PROC DQSCHEME statement (see "DQSCHEME Procedure Syntax" on page 29), and specify an existing fileref as the value of the SCHEME= option. The fileref must reference a fully qualified path with a filename that ends in `.sch.bfd`. *Lowercase letters are required.*

CAUTION:

In the z/OS operating environment, specify only schemes using SAS format. BFD schemes can be applied but not created in the z/OS operating environment. △

To create a scheme in SAS format, specify the NOBFD option in the PROC DQSCHEME statement and specify a one-level or two-level SAS data set name.

The SCHEME= option is required if the ANALYSIS= option is not specified.

SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF

specifies one of three mutually exclusive methods of applying the scheme to the values of the input character variable. Valid values are defined as follows:

EXACT

this default value specifies that the values of the input variable are to be compared to the DATA values in the scheme without changing the input values in any way. The transformation value in the scheme is written into the output data set only when an input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input values are replaced with single blank spaces prior to comparison.

IGNORE_CASE

specifies that capitalization is to be ignored when input values are compared to the DATA values in the scheme. Any adjacent blank spaces in the input values are replaced with single blank spaces prior to comparison.

USE_MATCHDEF

specifies that comparisons are to be made between the *match codes* of the input values and the *match codes* of the DATA values in the scheme. A transformation occurs when the match code of an input value is identical to the match code of a DATA value in the scheme.

Specifying USE_MATCHDEF enables the options LOCALE=, MATCHDEF=, and SENSITIVITY=, which can be used to override the default values that may be stored in the scheme.

The value of the SCHEME_LOOKUP= option is stored in the scheme as a meta option, which specifies a default lookup method when the scheme is applied. The default value that is supplied by this meta option is superseded by a lookup method that is specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine. For more information on meta options, see "About Meta Options" on page 8.

SENSITIVITY=*information-factor*

optionally determines the amount of information that will be included in the match codes that are generated during the creation and perhaps the application of the scheme. Higher sensitivity values generate match codes that contain more information, which generally results in fewer matches, a greater number of clusters, and fewer values in each cluster. Valid values range from 50 to 95. The default value is 85.

The value of the SENSITIVITY= option is stored in the scheme as a meta option, which provides a default sensitivity value when the scheme is applied. This meta option is used at apply time only when SCHEME_LOOKUP=MATCHDEF. The default value that is supplied by this meta option is superseded by a sensitivity value that is specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine. For more information on meta options, see "About Meta Options" on page 8.

VAR=*variable-name*
identifies the input character variable that is analyzed and transformed.

APPLY Statement

Applies a scheme to a transform the values of a single variable.

See also: "Applying Schemes" on page 7.

APPLY
SCHEME=*scheme-name*
VAR=*variable-name*
<**SCHEME_LOOKUP**=**EXACT** | **IGNORE_CASE** | **USE_MATCHDEF**>
<**MATCHDEF**=*match definition*>
<**SENSITIVITY**=*complexity-factor*>
<**LOCALE**=*locale-name*>
<**MODE**=**PHRASE** | **ELEMENT**>;

LOCALE=*locale-name*
specifies the locale that contains the match definition that is specified in the MATCHDEF= option.

Note: This option is valid only when SCHEME_LOOKUP= USE_MATCHDEF. △

If USE_MATCHDEF is specified and LOCALE= is not specified, the locale that is used is the default locale, which is the first locale in the locale list.

If USE_MATCHDEF is not specified, then the locale that is used is the locale that is stored in the scheme or the default locale if a locale is not stored in the scheme.

The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

MATCHDEF=*match-definition*
specifies the name of the match definition in the specified locale that will be used to create match codes during the application of the scheme.

Note: This option is valid only when SCHEME_LOOKUP= USE_MATCHDEF. △

If USE_MATCHDEF is specified and the MATCHDEF= option is not specified, the match definition that is used is the one that is stored in the scheme. If USE_MATCHDEF is not specified and if a match definition is not stored in the scheme, then a value is required for the MATCHDEF= option.

For information on available match definitions, see the documentation for your locale, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

MODE=PHRASE | ELEMENT

specifies how the scheme is to be applied. The default value is the mode that is stored in the scheme. If no mode is stored in the scheme, then PHRASE is the default mode.

Valid values for the MODE= option are defined as follows:

PHRASE

specifies that the entirety of each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the match code for the entire input value is compared to the match codes that are generated for each DATA value in the scheme.

ELEMENT

specifies that each element in each value of the input character variable is compared to the DATA values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the match code for each element is compared to the match codes that are generated for each element in each data value in the scheme.

SCHEME=*scheme-name*

identifies the scheme to apply to the input data set. In all operating environments other than z/OS, schemes using BFD format are identified by specifying a fileref for a fully-qualified filename that ends in `.sch.bfd`. In the z/OS operating environment, no special naming conventions are required. For schemes with SAS format, specify a one-level or two-level SAS data set name.

SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF

specifies one of three mutually exclusive methods of applying the scheme to the values of the input character variable. Valid values are defined as follows:

EXACT

this default value specifies that the values of the input variable are to be compared to the DATA values in the scheme without changing the input values in any way. The transformation value in the scheme is written into the output data set only when an input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input values are replaced with single blank spaces prior to comparison.

IGNORE_CASE

specifies that capitalization is to be ignored when input values are compared to the DATA values in the scheme. Any adjacent blank spaces in the input values are replaced with single blank spaces prior to comparison.

USE_MATCHDEF

specifies that comparisons are to be made between the *match codes* of the input values and the *match codes* of the DATA values in the scheme. A transformation occurs when the match code of an input value is identical to the match code of a DATA value in the scheme.

Specifying USE_MATCHDEF enables the options LOCALE=, MATCHDEF=, and SENSITIVITY=, which can be used to override the default values that may be stored in the scheme.

Note: The options LOCALE=, MATCHDEF=, and SENSITIVITY= are valid only when SCHEME_LOOKUP= USE_MATCHDEF. △

SENSITIVITY=*complexity-factor*

specifies the amount of information in the match codes that are created during the application of the scheme. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, two values can receive the same match code despite dissimilarities.

Note: The SENSITIVITY= option is valid only when SCHEME_LOOKUP= USE_MATCHDEF. △

If USE_MATCHDEF is specified and SENSITIVITY= is not specified, the sensitivity value is 85. If USE_MATCHDEF is not specified, the sensitivity value is the value that is stored in the scheme. If USE_MATCHDEF is not specified and if a sensitivity value is not stored in the scheme, the sensitivity value is 85.

Valid values for the SENSITIVITY= option range from 50 to 95.

VAR=*variable-name*
identifies the character variable in the input data set that is to be transformed.

CONVERT Statement

Converts schemes between SAS and BFD formats.

Required: All options are required (IN, OUT, and BFDTOSAS or SASTOBFD).

See also: "Applying Schemes" on page 7.

CONVERT
IN=*file-specification***OUT**=*file-specification*
BFDTOSAS | SASTOBFD ;

BFDTOSAS | SASTOBFD
specify BDFTOSAS to convert a scheme in Blue Fusion Data format to SAS format. Specify SASTOBFD to convert a scheme in SAS format to Blue Fusion Data format. Schemes with SAS format are created with the CREATE statement using the NOBFD option on the PROC DQSCHEME statement.

CAUTION:
In the z/OS operating environment, specify BFDTOSAS only. In z/OS, schemes in BFD format can be applied but not created. △

IN=*file-specification*
identifies the existing scheme that is to be converted. If BFDTOSAS is specified, then the value must be the name of a fileref that references a fully-qualified path in lowercase that ends in `.sch.bfd`. (In the z/OS operating environment, the PDS specification has no special naming requirements.) If SASTOBFD is specified, then the value must be a one-level or two-level SAS data set name.

OUT=*file-specification*
specifies the name of the converted scheme. If BFDTOSAS is specified, then the value must be a one-level or two-level SAS data set name. If SASTOBFD is specified, then the value must be the name of a fileref that references a fully-qualified path in lowercase that ends in `.sch.bfd`. (In the z/OS operating environment, the PDS specification has no special naming requirements.)

PROC DQSCHEME Examples

Example 1: Creating an Analysis Data Set

The following example generates an analysis of the STATE variable in the VENDORS data set. Note that you do not have to create a scheme to generate the analysis data set. Also note that the locale ENUSA is assumed to have been loaded into memory as part of the locale list.

```
 /* Create the input data set. */
data vendors;
   input city $char16. state $char22. company $char34.;
datalines;
Detroit         MI                    Ford Motor
Dallas          Texas                 Wal-mart Inc.
Washington      District of Columbia  Federal Reserve Bank
SanJose         CA                    Wal mart
New York        New York              Ernst & Young
Virginia Bch    VA                    TRW INC - Space Defense
Dalas           TX                    Walmart Corp.
San Francisco   California            The Jackson Data Corp.
New York        NY                    Ernst & Young
Washington      DC                    Federal Reserve Bank 12th District
New York        N.Y.                  Ernst & Young
San Francisco   CA                    Jackson Data Corporation
Atlanta         GA                    Farmers Insurance Group
RTP             NC                    Kaiser Permantente
New York        NY                    Ernest and Young
Virginia Beach  VIRGINIA              TRW Space & Defense
Detroit         Michigan              Ford Motor Company
San Jose        CA                    Jackson Data Corp
Washington      District of Columbia  Federal Reserve Bank
Atlanta         GEORGIA               Target
;
run;

 /* Create the analysis data set. */
proc dqscheme data=vendors;
   create analysis=a_state
          matchdef='State (Scheme Build)'
          var=state
          locale='ENUSA';
run;

 /* Print the analysis data set. */
title 'Analysis of state name variations';
proc print data=a_state;
```

```
run;
```

The analysis data set WORK.A_STATE shows for each value for the STATE variable the number of occurrences and the associated cluster number. Variables that are not clustered with any other values have a blank value for the cluster number.

Example 2: Creating Schemes

The following example generates three schemes in SAS format. Note that the locale ENUSA is assumed to have been loaded into memory as part of the locale list.

```
/* Create the input data set. */
data vendors;
  input city $char17. state $char22. company $char36.;
datalines;
Detroit          MI                     Ford Motor
Dallas           Texas                  Wal-mart Inc.
Washington       District of Columbia   Federal Reserve Bank

/* See Example 1 on page 36 for the full data set. */

Washington       District of Columbia   Federal Reserve Bank
Atlanta          GEORGIA                Target
;
run;

proc dqscheme data=vendors nobfd;
  create matchdef='City (Scheme Build)' var=city
    scheme=city_scheme locale='ENUSA';
  create matchdef='State (Scheme Build)' var=state
    scheme=state_scheme locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
    var=company scheme=org_scheme locale='ENUSA';
run;

title 'City scheme';
proc print data=work.city_scheme;
run;

title 'State scheme';
proc print data=work.state_scheme;
run;

title 'Organization scheme';
proc print data=work.org_scheme;
run;
```

Notice that we did not create and then immediately apply one or more schemes within the same step. After you create schemes, it is important that someone familiar with the data reviews the results. In this particular example, the City scheme chose **`Dalas`** as the transformation value for the city of Dallas. Although the values **`Dalas`** and **`Dallas`** were correctly clustered together, you would probably prefer **`Dallas`** to be the transformation value.

Example 3: Creating BFD Schemes

Blue Fusion Data schemes can be read by SAS and by the dfPower Studio software. Generating Blue Fusion Data schemes is advantageous when you wish to use dfPower to edit those schemes. The following example generates three schemes in Blue Fusion Data format. Also note that the locale ENUSA is assumed to have been loaded into memory as part of the locale list.

```
/* Create filerefs with required suffixes. */
filename city 'c:\my schemes\city.sch.bfd';
filename state 'c:\my schemes\state.sch.bfd';
filename org 'c:\my schemes\org.sch.bfd';

/* Create the input data set. */
data vendors;
  input city $char17. state $char22. company $char36.;
datalines;
Detroit          MI                      Ford Motor
Dallas           Texas                   Wal-mart Inc.
Washington       District of Columbia    Federal Reserve Bank

/* See Example 1 on page 36 for the full data set. */

Washington       District of Columbia    Federal Reserve Bank
Atlanta          GEORGIA                 Target
;
run;

proc dqscheme data=vendors bfd;
  create matchdef='City (Scheme Build)'
    var=city scheme=city locale='ENUSA';
  create matchdef='State (Scheme Build)'
     var=state scheme=state locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
     var=company scheme=org locale='ENUSA';
run;
```

Example 4: Applying Schemes

In this example, the APPLY statement generates cleansed data in the VENDORS_OUT data set. All schemes are applied before the result is written into the output data set. The locale ENUSA is assumed to be loaded into memory as part of the locale list.

```
/* Create filerefs with required suffixes. */
filename city 'c:\my schemes\city.sch.bfd';
filename state 'c:\my schemes\state.sch.bfd';
filename org 'c:\my schemes\org.sch.bfd';
```

```
/* Create the input data set. */
data vendors;
  input city $char17. state $char22. company $char36.;
datalines;
Detroit          MI                     Ford Motor
Dallas           Texas                  Wal-mart Inc.
Washington       District of Columbia   Federal Reserve Bank

/* See Example 1 on page 36 for the full data set. */

Washington       District of Columbia   Federal Reserve Bank
Atlanta          GEORGIA                Target
;
run;

proc dqscheme data=vendors out=vendors_out bfd;
  create matchdef='City (Scheme Build)'
    var=city scheme=city_scheme locale='ENUSA';
  create matchdef='State (Scheme Build)'
     var=state scheme=state_scheme locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
     var=company scheme=org_scheme locale='ENUSA';
  apply var=city scheme=city_scheme;
  apply var=state scheme=state_scheme;
  apply var=company scheme=org_scheme;
run;

title 'Result after applying all three SAS format schemes';
proc print data=work.vendors_out;
run;
```

Note that the APPLY statements do not specify a locale. Nor do they specify the scheme lookup method using the SCHEME_LOOKUP= option. This means that the schemes are applied with the ENUSA locale, which was stored in the schemes when they were created. Also, the default scheme lookup method (SCHEME_LOOKUP=EXACT) specifies that the value in the scheme replaces the input value in the output data set when an exact match is found between the input value and a DATA value in the scheme. The default scheme apply mode (MODE=PHRASE) is used, which means that the entirety of each input value is compared to the DATA values in the scheme.

CHAPTER
5

AUTOCALL Macros

AUTOCALL Macros for SAS Data Quality Server

The SAS Data Quality Server software provides the following AUTOCALL macros:

- □ "%DQLOAD AUTOCALL Macro" on page 41.
- □ "%DQPUTLOC AUTOCALL Macro" on page 42.
- □ "%DQUNLOAD AUTOCALL Macro" on page 43.

For more general information about the SAS AUTOCALL libraries and macros, see *SAS Macro Language: Reference* and *SAS Language Reference: Dictionary*.

%DQLOAD AUTOCALL Macro

Sets system option values and loads locales into memory.

Syntax

%**DQLOAD**(**DQLOCALE**=(*locale1 ...localeN*), **DQSETUPLOC**=*'file-specification'*,
<**DQINFO**=0 | 1>)

DQLOCALE=(*locale1 ...localeN*)
specifies a value for the DQLOCALE= system option to load an ordered list of locales into memory.

DQSETUPLOC=*'file-specification'*
specifies a value for the DQSETUPLOC= system option, which identifies the location of the setup file. The setup file in turn identifies the location of the Quality Knowledge Base, which contains the specified locales.

DQINFO=0 | 1
specifying DQINFO=1 generates additional information in the SAS log about the status of the locale load operation. The default value is DQINFO=0.

Details

The %DQLOAD AUTOCALL macro should be used at the beginning of each data cleansing program to ensure that the proper list and order of locales is loaded into memory prior to cleansing data. This loading prevents the use of an unintended default locale or locale list. Specifying the %DQLOAD macro before data cleansing, instead of at SAS invocation using an AUTOEXEC or configuration file, preserves memory and shortens the duration of the SAS invocation. Doing so is particularly beneficial when the SAS session will not be used to run data cleansing programs.

It is strongly suggested that you use only the %DQLOAD macro to set the value of the DQLOCALE= system option. Setting the value of this system option by the usual means (such as an OPTIONS statement) does not load the specified locales into memory. Not loading locales into memory can lead to the use of an unintended locale. For the same reason, it is not recommended that you set the DQLOCALE= system option at SAS invocation using a configuration file or AUTOEXEC.

In addition to setting the DQLOCALE= system option, the %DQLOAD macro also sets the DQSETUPLOC= system option (if that value is not set by default at your site). When SAS is installed, the value of the DQSETUPLOC= option is set to point to the location where the setup file is installed.

Example

The following example loads the ENUSA and DEDEU locales into memory in the UNIX operating environment:

```
%DQLOAD(DQLOCALE=(ENUSA DEDEU), DQSETUPLOC='/sas/dqc/dqsetup.txt');
```

%DQPUTLOC AUTOCALL Macro

Displays current information on a specified locale in the SAS log.

Requirement: At least one locale must be loaded into memory before this macro is called.

Tip: Specifying no parameters displays the full report for the default locale.

Syntax

%DQPUTLOC(*locale*, <**SHORT**=0 | 1>, <**PARSEDEFN**=0 | 1)>

locale

specifies the locale of interest. The value can be a locale name in quotation marks or the name of a variable whose value is a locale name or an expression that evaluates to a locale name.

The specified locale must have been loaded into memory as part of the locale list (see "About Locales" on page 4). If no value is specified, the default locale is used. The default locale is the first locale in the locale list.

SHORT= 0 | 1

optionally shortens the length of the entry in the SAS log. Specify SHORT=1 to remove the descriptions of how the definitions are used. The default value is SHORT=0, which displays the descriptions of how the definitions are used.

PARSEDEFN = 0 | 1
optionally lists with each match defintion the related parse definition, if such a parse definition exists. Specify PARSEDEFN=1 to list related parse defintion. The default value is PARSEDEFN=0, which does not display related parse definitions.

Details

The %DQPUTLOC AUTOCALL macro provides a quick means of displaying current information in the SAS log for the specified locale that is loaded into memory at that time. This display eliminates possible discrepancies that could be introduced by outdated documenation from other sources. Other sources of locale information, including the information provided for the ENUSA locale (see "ENUSA" on page 101), may have been superseded by a later download of a newer locale or by a new locale that was edited after download using the dfPower Customize software from DataFlux (a SAS company).

The available locale information includes a list of all definitions, parse tokens, related functions, and the names of the parse defintions that are related to each match definition. Knowing the related parse defintions enables the creation of parsed character values (see "DQPARSETOKENPUT Function" on page 63) and the creation of match codes for parsed character values (see "DQMATCHPARSED Function" on page 58).

Example

The following example displays in the SAS log the definitions, related parse definitions, and related SAS Data Quality Server functions for the ENUSA locale:

```
%dqputloc(enusa);
```

See Also

- □ "DQLOCALEINFOGET Function" on page 54.
- □ "DQLOCALEINFOLIST Function" on page 55.

%DQUNLOAD AUTOCALL Macro

Unloads all locales to increase the amount of free memory.

Syntax

%DQUNLOAD

Details

The %DQUNLOAD AUTOCALL macro unloads all locales that are currently loaded into memory. After unloading memory, be sure to load locales again with the %DQLOAD AUTOCALL macro before running any data cleansing programs.

CHAPTER 6
Functions and CALL Routines

Functions and CALL Routines in SAS Data Quality Server

The functions and CALL routines in the SAS Data Quality Server software enable you to cleanse data in specific and discrete steps. For lists of available functions and CALL routines, see "Functions Listed by Category" on page 46 and "Functions Listed Alphabetically" on page 46. For information on using these functions, see "Using the SAS Data Quality Server Functions" on page 15.

Note that the SAS Data Quality Server functions and CALL routines are also available in the Expression Builder of the SAS ETL Studio software.

Functions Listed Alphabetically

The "DQCASE Function" on page 48 returns a character value with standardized capitalization.

The "DQGENDER Function" on page 49 returns a gender determination from the name of an individual.

The "DQGENDERPARSED Function" on page 51 returns a gender determination from the parsed name of an individual.

The "DQIDENTIFY Function" on page 52 returns a category name from a character value.

The "DQLOCALEGUESS Function" on page 53 returns the name of the locale that is most likely represented by a character value.

The "DQLOCALEINFOGET Function" on page 54 returns information about locales.

The "DQLOCALEINFOLIST Function" on page 55 displays the names of the definitions in a locale and returns a count of those definitions.

The "DQMATCH Function" on page 56 returns a match codes from a character value.

The "DQMATCHINFOGET Function" on page 57 returns the name of the parse definition that is associated with a match definition.

The "DQMATCHPARSED Function" on page 58 returns a match code from a parsed character value.

The "DQPARSE Function" on page 59 returns a parsed character value.

The "DQPARSEINFOGET Function" on page 60 returns the token names for the specified parse definition.

The "DQPARSETOKENGET Function" on page 61 returns a token from a parsed character value.

The "DQPARSETOKENPUT Function" on page 63 inserts a token into a parsed character value and returns the updated parsed character value.

The "DQPATTERN Function" on page 64 returns a pattern analysis from an input character value.

The "DQSCHEMEAPPLY Function" on page 69 applies a scheme and returns a transformed value after applying a scheme.

The "DQSCHEMEAPPLY CALL Routine" on page 65 applies a scheme and returns a transformed value and a transformation flag.

The "DQSTANDARDIZE Function" on page 73 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.

The "DQTOKEN Function" on page 74 returns a token from a character value.

Functions Listed by Category

The functions in the SAS Data Quality Server software can be divided into the following categories.

Case Functions

The “DQCASE Function” on page 48 returns a character value with standardized capitalization.

Gender Analysis, Locale Guessing, and Identification Functions

The gender analysis, locale guessing, and identification functions return information that is determined from the conent of an input character value.

The “DQGENDER Function” on page 49returns a gender determination from the name of an individual.

The “DQGENDERINFOGET Function” on page 50 returns the name of the parse definition that is associated with a specified gender analysis definition.

The “DQGENDERPARSED Function” on page 51 returns a gender determination from the parsed name of an individual.

The “DQLOCALEGUESS Function” on page 53 returns the name of the locale that is most likely represented by a character value.

The “DQIDENTIFY Function” on page 52 returns a category name from a character value.

Matching Functions

The “DQMATCH Function” on page 56 returns a match code from a character value.

The “DQMATCHINFOGET Function” on page 57 returns the name of the parse definition that is associated with a match definition.

The “DQMATCHPARSED Function” on page 58 returns a match code from a parsed character variable.

Parsing Functions

The “DQPARSE Function” on page 59 returns a parsed character value.

The “DQPARSETOKENGET Function” on page 61 returns a token from a parsed character value.

The “DQPARSETOKENPUT Function” on page 63 inserts a token into a parsed character value and returns the updated parsed character value.

The “DQTOKEN Function” on page 74 returns a token from a character value.

Pattern Analysis Functions

The “DQPATTERN Function” on page 64 returns analytical information on the words or characters in an input character value.

Reporting Functions

The “DQGENDERINFOGET Function” on page 50 returns the name of the parse definition that is associated with a specified gender analysis definition.

The “DQLOCALEINFOGET Function” on page 54 returns information about locales.

The “DQLOCALEINFOLIST Function” on page 55 displays the names of the definitions in a locale and returns a count of those definitions.

The “DQMATCHINFOGET Function” on page 57 returns the name of the parse definition that is associated with a match definition.

The “DQPARSEINFOGET Function” on page 60 returns the token names for the specified parse definition.

Scheme Apply Functions and CALL Routines

The “DQSCHEMEAPPLY Function” on page 69 applies a scheme and returns a transformed value.

The “DQSCHEMEAPPLY CALL Routine” on page 65 applies a scheme and returns a transformed value and a transformation flag.

Standardization Functions

The “DQSTANDARDIZE Function” on page 73 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.

DQCASE Function

Returns a character value with standardized capitalization.

Valid: in the DATA step, PROC SQL, and SCL

Syntax

DQCASE(*char*, ‘*case-definition*’ <, ‘*locale*’>)

char
is the value that is transformed, according to the specified case definition. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

case-definition
specifies the name of the case definition that will be referenced during the transformation. If the content of the `char` value is represented by a case definition, that definition is recommended above a generic case definition. For example, if your

`char` value is a street address and if you are using the ENUSA locale, the recommended case definition would be PROPER – ADDRESS rather than the generic case definition PROPER.

The specified case definition must exist in the specified locale. For information on available case definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale

optionally specifies the name of the locale that contains the specified case definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The DQCASE function operates on any character content, such as names, organizations, and addresses. All instances of adjacent blank spaces are replaced with single blank spaces. For information on available case definitions, see "ENUSA" on page 101.

Example

The following example standardizes capitalization and spacing with the PROPER case definition in the ENUSA locale.

```
orgname=dqCase("BILL'S PLUMBING   &    HEATING", 'Proper', 'ENUSA');
```

After this function call, the value for the ORGNAME variable is **`Bill's Plumbing & Heating`**.

DQGENDER Function

Returns a gender determination from the name of an individual.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQGENDER(*char*, '*gender-analysis-definition*'<, '*locale*'>)

char

is the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

gender-analysis-definition

specifies the name of the gender analysis definition, which must exist in the specified locale. For information on available gender analysis definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified gender analysis definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The DQGENDER function evaluates the name of an individual to determine the gender of that individual. If the evaluation finds substantial clues that indicate gender, the function returns a value that indicates that the gender is female or male. If the evaluation is inconclusive, the function returns a value that indicates that the gender is unknown. The exact return value is determined by the specified gender analysis definition and locale.

For information on the available gender analysis definitions, see "ENUSA" on page 101.

Example

The following example returns the value `M` for the variable GENDER.

```
gender=dqGender('Mr. John B. Smith', 'Gender', 'ENUSA');
```

See Also

"DQGENDERPARSED Function" on page 51.

DQGENDERINFOGET Function

Returns the name of the parse definition that is associated with the specified gender definition.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQGENDERINFOGET ('*gender-analysis-definition*' <, '*locale*'>)

gender-analysis-definition
specifies the name of the gender analysis definition, which must exist in the specified locale. For information on available gender analysis definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified gender analysis definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Example

The following example displays the parse definition that is associated with the gender analysis definition in the ENUSA locale called GENDER. The parse definition that is returned is then used to display the names of the tokens that are enabled for that parse definition. The tokens are then used to construct a parsed value and return a match code for that parsed value.

```
/* display the parse definition associated with the */
/* GENDER definition and display the tokens in that */
/* parse definition.                                */
data _null_;
   parseDefn=dqGenderInfoGet('Gender', 'ENUSA');
   tokens=dqParseInfoGet(parseDefn, 'ENUSA');
   put parseDefn= / tokens=;
run;

/* build a parsed value from two tokens and display    */
/* in the log the gender determination for that value. */
data _null_;
   length parsedValue $ 200 gender $ 1;
   parsedValue=dqParseTokenPut(parsedValue, 'Sandi', 'Given Name', 'Name');
   parsedValue=dqParseTokenPut(parsedValue, 'Baker', 'Family Name', 'Name');
   gender=dqGenderParsed(parsedValue, 'Gender');
   put gender=;
run;
```

See Also

- "DQPARSE Function" on page 59.
- "DQPARSETOKENPUT Function" on page 63.
- "DQGENDERPARSED Function" on page 51.

DQGENDERPARSED Function

Returns a gender determination from the parsed name of an individual.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQGENDERPARSED(*parsed-char*, '*gender-analysis-definition*' <, '*locale*'>)

parsed-char
is a parsed value that contains the name of an individual. The value can be expressed as the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

gender-analysis-definition
specifies the name of the gender analysis definition that will be referenced to determine gender. The specified gender analysis definition has a related parse definition. To return an accurate gender determination, the related parse definition must be the same parse definition that was used to parse the **`parsed-char`**. To return the names of related parse definitions, use the DQGENDERINFOGET function (see "DQGENDERINFOGET Function" on page 50).

The specified gender analysis definition must exist in the specified locale. For information on available gender analysis definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified gender analysis definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The DQGENDERPARSED function returns a gender determination from a parsed character value that contains the name of an individual. If the analysis finds substantial clues that indicate the gender of the individual, the function returns a value that indicates that the gender is female or male. If the analysis is inconclusive, the function returns a value that indicates that the gender is unknown. The specific return value depends on the specified gender analysis definition and locale.

See Also

- □ "DQGENDERINFOGET Function" on page 50 for an example that uses DQGENDERPARSED.
- □ "DQGENDER Function" on page 49.

DQIDENTIFY Function

Returns a category name from a character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQIDENTIFY(*char*, *'identification-definition'*<, *'locale'*>)

char
is the value that is transformed, according to the specified identification definition. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

identification-definition
specifies the name of the identification definition, which must exist in the specified locale. For information on available identification definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified identification definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The DQIDENTIFY function returns a value that indicates the category of the content in an input character value. The available categories and return values depend on your choice of identification definition and locale.

Example

The following example determines if a character value represents an individual or an organization.

```
id=dqIdentify('LL Bean', 'Individual/Organization', 'ENUSA');
```

After this function call, the value for the ID variable is **`ORGANIZATION`**.

DQLOCALEGUESS Function

Returns the name of the locale that is most likely represented by a character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQLOCALEGUESS(*char*, '*locale-guess-definition*')

char
is the value that is analyzed to determine a locale, according to the specified guess definition. The value can be the name of a character variable, a character value in

quotation marks, or an expression that evaluates to a variable name or a quoted value.

locale-guess-definition
specifies the name of the guess definition. For information on available locale guess definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

Details

The DQLOCALEGUESS function evaluates the input character value using the specified locale guess definition in each of the locales that are loaded into memory. An applicability score is generated for each locale in the locale list. If the highest score is held by more than one locale or if none of the locales possess the specified locale guess definition, then the return value is the name of the first locale in the locale list.

For information on available locale guess definitions, see "ENUSA" on page 101.

Example

The following example returns the name of a locale as the value of the LOC variable:

```
loc=dqLocaleGuess('101 N. Main Street', 'Address');
```

The name of the locale that is returned depends on which locales are loaded into memory.

See Also

- □ "About Locales" on page 4.
- □ "ENUSA" on page 101.

DQLOCALEINFOGET Function

Returns information about locales.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQLOCALEINFOGET(<'*info-type*'>)

info-type
specifies the type of locale information that is to be returned. The only valid value is LOADED. If no parameter is specified, LOADED is used by default.

Details

The DQLOCALEINFOGET function returns a comma-delimited list of locale names. The ordered list contains the names of the locales that are currently loaded into memory. These locales are the ones that are available for use in data cleansing.

Example

The following example returns the locales that are currently loaded into memory.

```
loadedLocales=dqLocaleInfoGet('loaded');
put loadedLocales;
```

If the locales ENUSA and ENGBR locales are loaded in that order, the return value is **`ENUSA,ENGBR,`** and ENUSA is the default locale.

See Also

- "DQLOCALEINFOLIST Function" on page 55.
- "%DQPUTLOC AUTOCALL Macro" on page 42.
- "About Locales" on page 4.

DQLOCALEINFOLIST Function

Displays the names of the definitions in a locale and returns a count of those definitions.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQLOCALEINFOLIST('*definition-type*' <, '*locale*'>)

definition-type
specifies the definitions that are displayed. Valid values are as follows:
- ALL
- CASE
- GENDER
- GUESS
- IDENTIFICATION
- MATCH
- PARSE
- PATTERN
- STANDARDIZATION

locale
optionally specifies the name of the locale. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The DQLOCALEINFOLIST function displays in the SAS log the names of the definitions of the specified type in the specified locale. The return value of the function is the total number of definitions that are displayed in the log.

Examples

The following example displays in the SAS log a list of all of the definition names in the first locale in the locale list.

```
num=dqLocaleInfoList('all');
```

The following example displays a list of the parse definitions in the DEDEU locale.

```
num=dqLocaleInfoList('parse', 'DEDEU');
```

See Also

- "ENUSA" on page 101.
- "DQLOCALEINFOGET Function" on page 54.
- "%DQPUTLOC AUTOCALL Macro" on page 42.
- "About Locales" on page 4.

DQMATCH Function

Returns a match code from a character value.

Valid: in the DATA step, PROC SQL, and SCL

Syntax

DQMATCH(*char*, '*match-definition*' <, *sensitivity*, '*locale*'>)

char
is the value for which a match code is created, according to the specified match definition. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

match-definition
specifies the name of the match definition, which must exist in the specified locale. For information on available match definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

sensitivity
optionally specifies an integer value that determines the amount of information in the returned match code. Valid values range from 50 to 95. The default value is 85. A higher sensitivity value includes more information in the match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster, because matches require greater similarity between input values.

locale
optionally specifies the name of the locale that contains the specified match definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The DQMATCH function internally parses the input character value and creates and returns a match code. The match code represents a condensed version of the character value. The amount of information in the match code is determined by the sensitivity level. For higher sensitivities, two values must be very similar to produce the same match codes. At lower sensitivities, two values produce the same match codes despite their dissimilarities.

Example

The following example returns a match code that contains the maximum amount of information about the input value.

```
mcName=dqMatch('Dr. Jim Goodnight', 'NAME', 95, 'ENUSA');
```

See Also

- □ “Creating Match Codes” on page 9.
- □ Chapter 3, “The DQMATCH Procedure,” on page 19.
- □ “ENUSA” on page 101.

DQMATCHINFOGET Function

Returns the name of the parse definition that is associated with a match definition.

Valid in: in the DATA step, PROC SQL, or SCL

Syntax

DQMATCHINFOGET (‘*match-definition*’ <, ‘*locale*’>)

match-definition
specifies the name of the match definition, which must exist in the specified locale. For information on available match definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

locale
optionally specifies the name of the locale that contains the specified match definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The DQMATCHINFOGET function returns the name of the parse definition that is associated with the specified match definition. Obtaining the name of that parse definition enables you to create parsed character values with the DQPARSE or DQPARSETOKENPUT functions. If the specified match definition does not have an associated parse definition, the DQMATCHINFOGET function returns a missing value.

Example

The following example displays the name of the parse definition that is associated with the NAME match definition in the ENUSA locale. That parse definition is then used to display the tokens that are enabled for that parse definition. The tokens are then used to construct a parsed value, create and return a match code, and display the match code.

```
data _null_;
   parseDefn=dqMatchInfoGet('Name', 'ENUSA');
   tokens=dqParseInfoGet(parseDefn);
   put parseDefn= / tokens=;
run;

data _null_;
   length parsedValue $ 200 matchCode $ 15;
   parsedValue=dqParseTokenPut(parsedValue, 'Joel', 'Given Name', 'Name');
   parsedValue=dqParseTokenPut(parsedValue, 'Alston', 'Family Name', 'Name');
   matchCode=dqMatchParsed(parsedValue, 'Name');
   put matchCode=;
run;
```

DQMATCHPARSED Function

Returns a match code from a parsed character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQMATCHPARSED(*parsed-char*, '*match-definition*' <, *sensitivity*>, <, '*locale*'>)

parsed-char
is a parsed character value for which a match code will be created. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value. The value must have been parsed with the parse definition that is associated with the specified match definition. To determine the name of the associated parse definition, use "DQMATCHINFOGET Function" on page 57. To determine the tokens that are enabled by that parse definition, use "DQPARSEINFOGET Function" on page 60.

match-definition
specifies the name of the match definition, which must exist in the specified locale. For information on available match definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

sensitivity
optionally specifies an integer value that determines the amount of information in the returned match code. Valid values range from 50 to 95. The default value is 85. A higher sensitivity value inserts more information in the match code. In general, higher senstivity values result in a greater number of clusters, with fewer members per cluster; input values must be more similar to receive the same match codes.

locale
optionally specifies the name of the locale that contains the specified match definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Example

The following example returns a match code for the parsed name of an individual. The amount of information in the match code will be high.

```
data _null_;
   length nameIndividual matchCode $ 20 parsedName $ 200;
   nameIndividual='Susan B. Anthony';
   parsedName=dqParse(nameIndividual, 'name', 'enusa');
   matchCode=dqMatchParsed(parsedName, 'name', 90, 'enusa');
run;
```

See Also

- “Creating Match Codes” on page 9.
- Chapter 3, “The DQMATCH Procedure,” on page 19.
- “ENUSA” on page 101.

DQPARSE Function

Returns a parsed character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQPARSE(*char*, ‘ *parse-definition*’ <, ’*locale*’>)

char
is the value that is parsed according to the specified parse definition. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

parse-definition
specifies the name of the parse definition, which must exist in the specified locale. For information on available parse definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

locale
optionally specifies the name of the locale that contains the specified parse definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The DQPARSE function returns a parsed character value. The return value contains delimiters that identify the elements in the value that correspond to the tokens that are enabled by the parse definition. The delimiters in the value allow functions such as DQPARSETOKENGET to access the elements in the value based on specified token names.

Note: Always use the DQPARSETOKENGET function to extract tokens from parsed values. To extract tokens from values that do not contain delimiters, use the “DQTOKEN Function” on page 74 function. △

Example

The following example parses the name of an individual. Then the DQPARSETOKENGET function returns the values of two of the tokens.

```
parsedValue=dqParse('Mrs. Sallie Mae Pravlik', 'NAME', 'ENUSA');
prefix=dqParseTokenGet(parsedValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqParseTokenGet(parsedValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of the PREFIX variable is **`Mrs.`** and the value of the GIVEN variable is **`Sallie`**.

See Also

- “DQPARSEINFOGET Function” on page 60.
- “DQTOKEN Function” on page 74.

DQPARSEINFOGET Function

Returns the token names in a parse definition.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQPARSEINFOGET(‘*parse-definition*’ <, ‘*locale*’>)

parse-definition
specifies the name of the parse definition, which must exist in the specified locale. For information on available parse definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

locale
optionally specifies the name of the locale that contains the specified parse definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The DQPARSEINFOGET function returns the names of the tokens that can be inserted into character values using the DQPARSETOKENPUT function. You can also use the token names to return token values with the functions DQPARSETOKENGET and DQTOKEN.

Example

The following example returns the token names for the parse definition E–MAIL in the locale ENUSA and displays the token names in the SAS log.

```
tokenNames=dqParseInfoGet('e-mail', 'ENUSA');
put tokenNames;
```

After this function call, the value of the TOKENNAMES variable is **`Mailbox,Sub-Domain,Top-Level Domain`**, which are the names of the three tokens in this parse definition.

See Also

- “DQPARSETOKENPUT Function” on page 63.
- “DQPARSETOKENGET Function” on page 61.
- “DQTOKEN Function” on page 74.

DQPARSETOKENGET Function

Returns a token from a parsed character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQPARSETOKENGET(*parsed-char*, ‘*token*’, ‘*parse-definition*’ <, ’*locale*’>)

parsed-char
is the parsed character value from which will be returned the value of the specified token. The **`parsed-char`** can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value. The value must contain delimiters that are consistent with the specified parse definition. To determine how the parse definition inserts delimiters, use “DQPARSEINFOGET Function” on page 60.

token
specifies the name of the token that is returned from the parsed value. The token must be enabled by the specified parse definition.

parse-definition
specifies the name of the parse definition that will be used to obtain the value of the token. The parse definition must be the same as the parse definition that originally parsed the **`parsed-char`** value. The specified parse definition must exist in the specified locale. For information on available parse definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

locale
optionally specifies the name of the locale that contains the specified parse definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The DQPARSETOKENGET function returns the value of the specified token from a character value that was previously parsed.

Note: Do not attempt to extract tokens from parsed values using any means other than the DQPARSETOKENGET function. △

Example

The following example parses a character value with the DQPARSE function and extracts two of the tokens with the DQPARSETOKENGET function.

```
parsedValue=dqParse('Mrs. Sallie Mae Pravlik', 'NAME', 'ENUSA');
prefix=dqParseTokenGet(parsedValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqParseTokenGet(parsedValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of the PREFIX variable is **`Mrs.`** and the value of the GIVEN variable is **`Sallie`**.

See Also

- "DQPARSE Function" on page 59.
- "DQPARSEINFOGET Function" on page 60.
- "DQTOKEN Function" on page 74.

DQPARSETOKENPUT Function

Inserts a token into a parsed character value and returns the updated parsed character value.

Valid: in the DATA step and SCL

Syntax

DQPARSETOKENPUT(*parsed-char*, *token-value*, '*token-name*', '*parse-definition*' <, '*locale*'>)

parsed-char
is the parsed character value that receives the new token value, according to the specified parse definition. The `parsed-char` value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

token-value
specifies the value of the token that is to be inserted into the parsed value.

token-name
specifies the name of the token that is to be inserted into the parsed value. The specified token must be enabled by the specified parse definition.

parse-definition
specifies the name of the parse definition, which must exist in the specified locale. The specified parse definition must be the same definition that was used to parse the `parsed-char` value. For information on available parse definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified parse definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

Use the DQPARSETOKENPUT function to insert into a parsed value a new value that is associated with a specified token. If a value exists for that token in the input value, the new value is inserted before the existing value, and the existing value is retained.

As shown in the example that is referenced below, you can specify a variable name as the value of the **`parsed-char`** argument, then assign the return value from the function to the same variable.

See Also

- □ "DQGENDERINFOGET Function" on page 50 for an example that uses the DQPARSETOKENPUT function.
- □ "DQPARSETOKENGET Function" on page 61.
- □ "DQMATCHPARSED Function" on page 58.
- □ "DQGENDERPARSED Function" on page 51.

DQPATTERN Function

Returns a pattern analysis from an input character value.

Valid: in the DATA step, PROC SQL, and SCL

Syntax

DQPATTERN (*char*, '*pattern-analysis-definition*' <, '*locale*'>);

char
is the name of the value that will be analyzed. The value can be a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

pattern-analysis-definition
specifies the name of the definition that will be referenced during the creation of the pattern analysis. The definition must exist in the specified locale. For information on available pattern–analysis definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified pattern analysis definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The pattern analysis that is returned by the DQPATTERN function identifies words or characters in the input value as numeric, alphabetic, non-alphanumeric, or mixed. The choice of pattern analysis definition determines the nature of the analysis. For information on pattern analysis definitions in the ENUSA locale, see "ENUSA Pattern Analysis Definitions" on page 107.

Available return values are defined as follows:

*	non-alphanumeric, such as punctuation marks or symbols.
A	alphabetic
M	mixture of alphabetic, numeric, and non-alphanumeric.
N	numeric

Example

The following example analyzes the words in an input character value and displays the results in the SAS log.

```
pattern=dqPattern('WIDGETS 5" 32CT', 'WORD', 'ENUSA');
put pattern;
```

The value of the PATTERN variable after this function call would be **`A N* M`**. Using the CHARACTER pattern analysis definition returns the value **`AAAAAAA N* NNAA`**.

DQSCHEMEAPPLY CALL Routine

Applies a scheme and returns a transformed value and a transformation flag.

Requirement: Schemes using SAS format are required in the z/OS operating environment.

Valid: in the DATA step and SCL

Syntax

CALL DQSCHEMEAPPLY(*char*, *output-variable*, '*scheme*' <, '*scheme-format*', '*mode*', *transform-count-variable*, '*scheme-lookup-method*', '*match-definition*', *sensitivity*, '*locale*'>)

char
is the value to which the specified scheme will be applied. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

output-variable
identifies the character variable that receives the transformed input value.

scheme
identifies the scheme that is applied to the input value. For schemes using SAS format, the *scheme* argument is a fully-qualified SAS data set name in quotation marks. For schemes using Blue Fusion Data format, the *scheme* argument is the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a fully-qualified path that ends in **`.sch.bfd`**. *Lowercase letters are required.* In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

scheme-format
optionally identifies the file format of the scheme. Valid values are as follows:

BFD
indicates that the specified scheme is stored in Blue Fusion Data format. This is the default value.

NOBFD
indicates that the specified scheme is stored in SAS format.

For further information, see "Applying Schemes" on page 7.

mode
specifies how the scheme is to be applied to the values of the input character variable. The default value of the *mode* argument is the mode that is stored in the scheme. If a mode is not stored in the scheme, then the default mode is PHRASE. If the value of the *scheme-lookup-method* argument is USE_MATCHDEF, and if a value is not specified for the *mode* argument, then the default mode is PHRASE.

Valid values for the *mode* argument are defined as follows:

PHRASE
compares the entire input character value to the entirety of each of the DATA values in the scheme. When the value of the *scheme-lookup-method* argument is USE_MATCHDEF, the match code of the entire input character value is compared to the match codes of the DATA values in the scheme. A transformation occurs when a match is found between the input character value (or match code) and a DATA value (or match code) in the scheme.

ELEMENT
compares each element in the input character value to each of the DATA values in the scheme. When the value of the *scheme-lookup-method* argument is USE_MATCHDEF, the match code of each element in the input character value is compared to the match codes of the DATA values in the scheme. A transformation occurs when a match is found between an element in the input character value (or match code of an element) and a DATA value (or match code) in the scheme.

transform-count-variable
optionally identifies the numeric variable that receives the returned number of transformations that were performed on the input value.

If the value of the mode argument is PHRASE and if the input value is not transformed, then the value of the transform-count variable is 0. If the input variable is transformed, then the value of the transform-count variable is 1.

If the value of the mode argument is ELEMENT and if the input value is not transformed, then the value of the transform-count variable is 0. If the input variable is transformed, then the value is a positive integer that represents the number of elements in the input value that were transformed.

Note: The transformation count may appear to be inaccurate if the transformation value in the scheme is the same as the input value (or element(s) in the input value). △

scheme-lookup-method
specifies one of three mutually-exclusive methods of applying the scheme. Valid values are as follows:

EXACT
this default value specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when

the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces prior to comparison.

IGNORE_CASE
: specifies that capitalization is to be ignored when the input value is compared to the DATA values in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces prior to comparison.

USE_MATCHDEF
: specifies that the *match code* of the input value is to be compared to the *match codes* of the DATA values in the scheme. A transformation occurs when the two match codes are identical.

Specifying USE_MATCHDEF enables the arguments *locale*, *match-definition*, and *sensitivity*, which can be used to override the default values that may be stored in the scheme.

Note: The arguments *locale*, *match-definition*, and *sensitivity* are valid only when the value of the *scheme-lookup-method* option is USE_MATCHDEF. △

match-definition

specifies the name of the match definition in the specified locale that will be used to create match codes during the application of the scheme.

Note: The *match-definition* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

If USE_MATCHDEF is specified and the *match-definition* argument is not specified, then the default match definition is the one that is stored in the scheme. If USE_MATCHDEF is specified and a match definition is not stored in the scheme, then a value is required for the *match-definition* argument.

For information on available match definitions, see Chapter 7, “Locales,” on page 77, or submit the AUTOCALL macro %DQPUTLOC (see “%DQPUTLOC AUTOCALL Macro” on page 42).

sensitivity

specifies the amount of information in the match codes that are created during the application of the scheme. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, two values receive the same match code despite their dissimilarities. Valid values range from 50 to 95.

Note: The *sensitivity* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

When USE_MATCHDEF is specified and the sensitivity argument is not specified, the default sensitivity is the sensitivity value that is stored in the scheme. When USE_MATCHDEF is specified and when a sensitivity value is not stored in the scheme, the default sensitivity value is 85.

locale

specifies the locale that contains the specified match definition that will be referenced during the application of the scheme. The value can be a locale name in quotation marks or the name of a variable whose value resolves to a locale name.

Note: The *locale* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see “About Locales” on page 4.

Details

The CALL routine DQSCHEMEAPPLY transforms an input value by applying a scheme. The scheme can be in SAS format or Blue Fusion Data format. Schemes using SAS format can be created with the DQSCHEME procedure (see “The DQSCHEME Procedure” on page 29). Schemes using Blue Fusion Data format can be created with the DQSCHEME procedure or with the dfPower Studio software from DataFlux (a SAS company).

Example

The following example generates a scheme using Blue Fusion Data format with the DQSCHEME procedure and then applies that scheme to a data set with CALL DQSCHEMEAPPLY. The example assumes that the ENUSA has been loaded into memory as the default locale.

```
/* Create the input data set. */
data suppliers;
   length company $ 50;
   input company $char50.;
datalines;
Ford Motor Company
Walmart Inc.
Federal Reserve Bank
Walmart
Ernest & Young
TRW INC - Space Defense
Wal-Mart Corp.
The Jackson Data Corp.
Ernest & Young
Federal Reserve Bank 12th District
Ernest and Young
Jackson Data Corp.
Farmers Insurance Group
Kaiser Permantente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;

/* Create the scheme. */
proc dqscheme data=suppliers bfd;
  create matchdef='Organization (Scheme Build)'
    var=company scheme=work.myscheme
    locale='ENUSA';
run;

/* Print the scheme. */
proc print data=work.myscheme;
title 'Organization Scheme';
run;
```

```
/* Apply the scheme and display the results. */
data suppliers;
   set suppliers;
   length outCompany $ 50;
   call dqSchemeApply(company, outCompany, 'work.myscheme', 'nobfd', 'phrase', numTrans);
   put 'Before applying the scheme: ' company /
       'After applying the scheme:  ' outCompany /
       'Transformation count:       ' numTrans /;
run;
```

The value of the NUMTRANS variable is 0 if the organization name is not transformed or 1 if the organization name is transformed. In the output of this example, a transformation count of 1 is shown in several instances, when no transformation appears to have been made, as shown in the following PROC PRINT output:

```
Before applying the scheme: Jackson Data Corp
After applying the scheme:  Jackson Data Corp
Transformation count:       1
```

Instances such as these are not errors because in these cases the transformation value is the same as the input value.

See Also

- “DQSCHEMEAPPLY Function” on page 69.
- Chapter 4, “The DQSCHEME Procedure,” on page 29.
- “Transforming Data with Schemes” on page 6.

DQSCHEMEAPPLY Function

Applies a scheme and returns a transformed value.

Requirement: Schemes using SAS format are required in the z/OS operating environment.

Valid: in the DATA step, PROC SQL, and SCL

Syntax

DQSCHEMEAPPLY(*char*, ‘*scheme*’ <, ‘*scheme-format*’, ‘*mode*’, ‘*scheme-lookup-method*’, ‘*match-definition*’, *sensitivity*, ‘*locale*’>)

char
is the value to which the specified scheme will be applied. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

output-variable
identifies the character variable that receives the transformed input value.

scheme
identifies the scheme that is applied to the input value. For schemes using SAS format, the *scheme* argument is a fully-qualified SAS data set name in quotation

marks. For schemes using Blue Fusion Data format, the *scheme* argument is the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a fully-qualified path that ends in **`.sch.bfd`**. *Lowercase letters are required.* In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

scheme-format

optionally identifies the file format of the scheme. Valid values are as follows:

BFD

indicates that the specified scheme is stored in Blue Fusion Data format. This is the default value.

NOBFD

indicates that the specified scheme is stored in SAS format.

For further information, see "Applying Schemes" on page 7.

mode

specifies how the scheme is to be applied to the values of the input character variable. The default value of the *mode* argument is the mode that is stored in the scheme (see "About Meta Options" on page 8). If a mode is not stored in the scheme, then the default mode is PHRASE. If the value of the *scheme-lookup-method* argument is USE_MATCHDEF, and if a value is not specified for the *mode* argument, then the default mode is PHRASE.

Valid values for the *mode* argument are defined as follows:

PHRASE

compares the entire input character value to the entirety of each of the DATA values in the scheme. When the value of the *scheme-lookup-method* argument is USE_MATCHDEF, the match code of the entire input character value is compared to the match codes of the DATA values in the scheme. A transformation occurs when a match is found between the input character value (or match code) and a DATA value (or match code) in the scheme.

ELEMENT

compares each element in the input character value to each of the DATA values in the scheme. When the value of the *scheme-lookup-method* argument is USE_MATCHDEF, the match code of each element in the input character value is compared to the match codes of the DATA values in the scheme. A transformation occurs when a match is found between an element in the input character value (or match code of an element) and a DATA value (or match code) in the scheme.

scheme-lookup-method

specifies one of three mutually-exclusive methods of applying the scheme. Valid values are as follows:

EXACT

this default value specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces prior to comparison.

IGNORE_CASE

specifies that capitalization is to be ignored when the input value is compared to the DATA values in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces prior to comparison.

USE_MATCHDEF

specifies that the *match code* of the input value is to be compared to the *match codes* of the DATA values in the scheme. A transformation occurs when the two match codes are identical.

Specifying USE_MATCHDEF enables the arguments *locale*, *match-definition*, and *sensitivity*.

Note: The arguments *locale*, *match-definition*, and *sensitivity* are valid only when the value of the *scheme-lookup-method* option is USE_MATCHDEF. △

The default value of *scheme_lookup_method* option is generally provided in the scheme (see "About Meta Options" on page 8). If not provided in the scheme, the default value is EXACT.

match-definition

specifies the name of the match definition in the specified locale that will be used to create match codes during the application of the scheme.

Note: The *match-definition* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

If USE_MATCHDEF is specified and the *match-definition* argument is not specified, then the default match definition is the one that is stored in the scheme (see "About Meta Options" on page 8). If USE_MATCHDEF is specified and a match definition is not stored in the scheme, then a value is required for the *match-definition* argument.

For information on available match definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

sensitivity

specifies the amount of information in the match codes that are created during the application of the scheme. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, values can receive the same match code despite their dissimilarities. Valid values range from 50 to 95.

Note: The *sensitivity* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

When USE_MATCHDEF is specified and the sensitivity argument is not specified, the default sensitivity is the sensitivity value that is stored in the scheme (see "About Meta Options" on page 8). When USE_MATCHDEF is specified and when a sensitivity value is not stored in the scheme, the default sensitivity value is 85.

locale

specifies the locale that contains the specified match definition that will be referenced during the application of the scheme. The value can be a locale name in quotation marks, or the name of a variable whose value resolves to a locale name.

Note: The *locale* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. △

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

The CALL routine DQSCHEMEAPPLY transforms an input value by applying a scheme. The scheme can be in SAS format or Blue Fusion Data format. Schemes using SAS format can be created with the DQSCHEME procedure (see "The DQSCHEME

Procedure" on page 29). Schemes using Blue Fusion Data format can be created with the DQSCHEME procedure or with the dfPower Studio software from DataFlux (a SAS company).

Note: To return a count of the number of transformations that take place during a scheme application, use "DQSCHEMEAPPLY CALL Routine" on page 65. △

Example

The following example generates a scheme with the DQSCHEME procedure and then applies that scheme to a data set with the DQSCHEME function. This example assumes that the ENUSA locale has been loaded into memory as part of the locale list.

```
/* Create the input data set. */
data suppliers;
   length company $ 50;
   input company $char50.;
datalines;
Ford Motor Company
Walmart Inc.
Federal Reserve Bank
Walmart
Ernest & Young
TRW INC - Space Defense
Wal-Mart Corp.
The Jackson Data Corp.
Ernest & Young
Federal Reserve Bank 12th District
Ernest and Young
Jackson Data Corp.
Farmers Insurance Group
Kaiser Permantente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;

/* Assign a fileref to the scheme file. */
filename myscheme 'c:\temp\company.sch.bfd';

/* Create the scheme. */
proc dqscheme data=suppliers bfd;
  create matchdef='Organization (Scheme Build)'
    var=company scheme=myscheme
    locale='ENUSA';
run;

/* Apply the scheme and display the results. */
data suppliers;
   set suppliers;
```

```
      length outCompany $ 50;
      outCompany=dqSchemeApply(company, 'myscheme', 'bfd', 'phrase', 'EXACT');
      put 'Before applying the scheme: ' company /
          'After applying the scheme:  ' outCompany;
   run;
```

See Also

- □ "DQSCHEMEAPPLY CALL Routine" on page 65.
- □ Chapter 4, "The DQSCHEME Procedure," on page 29.
- □ "Transforming Data with Schemes" on page 6.

DQSTANDARDIZE Function

Returns a character value after standardizing casing, spacing, and format, and applies a common representation to certain words and abbreviations.

Valid: in the DATA step, PROC SQL, and SCL

Syntax

DQSTANDARDIZE(*char*, ' *standardization-definition*' <, *locale*>)

char
is the value that will be standardized according to the specified standardization definition. The value can be the name of a character variable, a character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

standardization-definition
specifies the name of the standardization definition, which must exist in the specified locale. For information on available standardization definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified standardization definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

In the locales, standardization definitions are provided for character content such as dates, names, and ZIP codes. The available standardization definitions vary from one locale to the next.

The return value is provided in appropriate case, with insignificant blank space and punctuation removed. The standardization definition that was specified in the

DQSTANDARDIZE function may standardize certain words and abbreviations. The order of the elements in the return value may differ from the order of the elements in the input character value.

Example

The following example standardizes four names using the NAME standardization definition from the ENUSA locale. This example assumes that the ENUSA locale has been loaded into memory as part of the locale list.

```
data _null_;
   length name stdName $ 50;
   input name $char50.;
   stdName=dqStandardize(name, 'Name');
   put 'Name:' @10 name /
       'StdName:' @10 stdName /;
datalines;
HOUSE, KEN
House, Kenneth
House, Mr. Ken W.
MR. KEN W. HOUSE
;
run;
```

After this function call, the SAS log displays the following information:

```
Name:    HOUSE, KEN
StdName: Ken House

Name:    House, Kenneth
StdName: Kenneth House

Name:    House, Mr. Ken W.
StdName: Mr Ken W House

Name:    MR. KEN W. HOUSE
StdName: Mr Ken W House
```

DQTOKEN Function

Returns a token from a character value.

Valid: in the DATA step, PROC SQL, or SCL

Syntax

DQTOKEN(*char*, '*token*', '*parse-definition*'<, *locale*>)

char
is the value from which the specified token will be returned, according to the specified parse definition. The value can be the name of a character variable, a

character value in quotation marks, or an expression that evaluates to a variable name or a quoted value.

token
identifies the token that is returned.

parse-definition
specifies the name of the parse definition, which must exist in the specified locale. For information on available parse definitions, see Chapter 7, "Locales," on page 77, or submit the AUTOCALL macro %DQPUTLOC (see "%DQPUTLOC AUTOCALL Macro" on page 42).

locale
optionally specifies the name of the locale that contains the specified parse definition. The value can be a name in quotation marks, the name of a variable whose value is a locale name, or an expression that evaluates to a variable name or to a quoted locale name.

The specified locale must be loaded into memory as part of the locale list. If no value is specified, the default locale is used. The default locale is the first locale in the locale list. For information on the locale list, see "About Locales" on page 4.

Details

Use the DQTOKEN function to parse a value and return one token. If the DQTOKEN function does not find a value for that token, the return value for that token will be blank.

To return more than one token from a parsed value, use the combination of the "DQPARSE Function" on page 59 and the "DQPARSETOKENGET Function" on page 61.

Example

The following example parses a single token from a character value.

```
prefix=dqToken('Mrs. Sallie Mae Pravlik', 'Name Prefix', 'Name', 'ENUSA');
```

After this function call, the value for the PREFIX variable is **`Mrs.`**.

See Also

- □ "DQPARSE Function" on page 59.
- □ "DQPARSETOKENGET Function" on page 61.

CHAPTER 7 Locales

Overview

The following table lists the locales that are provided in the Quality Knowledge Base.

Table 7.1 Locales in the Quality Knowledge Base

Locale	See	National Language	Region
Global	"Global Definitions" on page 78		
L1	"L1" on page 78	ISO-8859 (Latin-1)	
DE	"DE" on page 79	German	
DEDEU	"DEDEU" on page 81	German	Germany
EN	"EN" on page 86	English	
ENAUS	"ENAUS" on page 90	English	Australia
ENGBR	"ENGBR" on page 95	English	Great Britain
ENUSA	"ENUSA" on page 101	English	United States
IT	"IT" on page 109	Italian	
ITITA	"ITITA" on page 111	Italian	Italy
NL	"NL" on page 115	Dutch	
NLNLD	"NLNLD" on page 116	Dutch	Netherlands

For information on using locales, see "About the Quality Knowledge Base" on page 3.
To download updated versions of locales such as ENUSA and to receive updated documentation on those locales, refer to the following Web site of DataFlux (a SAS company):

```
www.dataflux.com/QKB
```

The DataFlux Web site also enables you to obtain additional or updated locales.

Global Definitions

Provide information about locales.

Global Locale Guess Definitions

The Quality Knowledge Base includes the following locale guess definitions, which are not part of a locale:

ADDRESS
: determines the name of the locale that best fits an input street address.

COUNTRY
: determines the name of the locale that best fits an input country name.

LOCALE CODE
: determines the name of the locale that best fits an input locale code (such as ENUSA or DEDEU).

L1

Provides data cleansing definitions for the Latin-1 character encoding standard ISO-8859–1.

The L1 locale is the ancestor the following locales:

- "DE" on page 79.
- "EN" on page 86.
- "IT" on page 109.
- "NL" on page 115.

Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The L1 locale includes:

- Parse definitions (see "L1 Parse Definitions" on page 78).
- Pattern analysis definitions (see "L1 Pattern Analysis Definitions" on page 79).

L1 Parse Definitions

The L1 locale includes the following parse definitions:

E-MAIL
: parses the addresses of electronic mail. Values are sought for the following tokens:
  - MAILBOX
  - SUB-DOMAIN
  - TOP-LEVEL DOMAIN

For example, the token/value pairs for the e-mail address **`JohnDoe@hisIsp.com`** are:

MAILBOX	JohnDoe
SUB-DOMAIN	hisIsp
TOP-LEVEL DOMAIN	com

L1 Pattern Analysis Definitions

The L1 locale includes the following pattern analysis definitions:

CHARACTER
: generates one character of analytical information for each character in the input character value. Return values are an asterisk (`*`) for non-alphanumeric, **`A`** for alphabetic, or **`N`** for numeric.

WORD
: generates one character of analytical information for each word in the input character value. Return values are the same as those in the CHARACTER definition, with the addition of **`M`** for words that contain a mixture of two or more of the character types alphanumeric, numeric, and non-alphanumeric.

DE

Provides inheritable data cleansing definitions for the German language.

The DE locale is a descendant of the L1 locale (see “L1” on page 78). The DE locale is the ancestor of the DEDEU locale (see “DEDEU” on page 81). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The DE locale includes:

- Match definitions, see “DE Match Definitions” on page 79.
- Parse definitions, see “DE Parse Definitions” on page 80.
- Pattern analysis definitions, see “DE Pattern Analysis Definitions” on page 81.
- Standardization definitions, see “DE Standardization Definitions” on page 81.

DE Match Definitions

The DE locale includes the following match definitions:

CITY
: creates match codes for city names. For example, at certain levels of sensitivity, the same match codes are created for the following city names:

```
Doerfles-Esbach
Derfles Esbach
```

E-MAIL
creates match codes for the addresses of electronic mail. For example, at certain levels of sensitivity, the same match codes are created for the following e-mail addresses:

```
PMayer@T-Online.de
pmeier@t-online.de
```

NAME
creates match codes for the names of individuals. For example, at certain levels of sensitivity, the same match codes are created for the following names:

```
Herr Peter Kuehner
Peter Khner
```

TEXT
creates match codes for general text. For example, at certain levels of sensitivity, the same match codes are created for the following character values:

```
dfPower Studio
Das DfPower Studio
```

DE Parse Definitions

The DE locale contains the following parse definitions:

NAME
parses the names of individuals. Values are sought for the following tokens:

- PREFIX
- GIVEN NAME
- FAMILY NAME
- SUFFIX
- TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Herr Johann Scheuchenzuber, jun.`** are:

PREFIX	Herr
GIVEN NAME	Johann
FAMILY NAME	Scheuchenzuber
SUFFIX	jun.

NAME (GLOBAL)
parses the names of individuals. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- GIVEN NAME
- MIDDLE NAME
- FAMILY NAME
- SUFFIX
- TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Hans-Peter Khnel`** are:

GIVEN NAME	Hans-Peter
FAMILY NAME	Kühnel

DE Pattern Analysis Definitions

The DE locale contains the following pattern analysis definitions:

CHARACTER
: generates one character of analytical information for each character in the input character value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

WORD
: generates one character of analytical information for each word in the input character value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

DE Standardization Definitions

The DE locale contains the following standardization definitions:

CITY
: standardizes city names. For example, the city name **`Neustadt b.  Coburg`** is standardized as **`NEUSTADT BEI COBURG`**.

NAME
: standardizes the names of individuals. For example, the name **`vogelsang, sven`** is standardized as **`SVEN VOGELSANG`**.

DEDEU

Provides inheritable data cleansing definitions for the German language, for use with data from Germany.

The DEDEU locale is a descendant of the L1 and DE locales. Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The DEDEU locale includes:

- □ Match definitions, see "DEDEU Match Definitions" on page 81.
- □ Parse definitions, see "DEDEU Parse Definitions" on page 82.
- □ Pattern analysis definitions, see "DEDEU Pattern Analysis Definitions" on page 85.
- □ Standardization definitions, see "DEDEU Standardization Definitions" on page 85.

DEDEU Match Definitions

The DEDEU locale contains the following match definitions:

ADDRESS
: creates match codes for the "first line" (street address) of a mailing address. At certain levels of sensitivity, the same match codes are created for the following character values:

```
Viktoriastr. 35
Viktoriastraβe 35
```

CITY
creates match codes for city names. This definition is inherited from the DE locale (see “DE Match Definitions” on page 79).

CITY - STATE/PROVINCE - POSTAL CODE
creates match codes for the “last line” of mailing addresses. At certain levels of sensitivity, the same match codes are created for the following values:

```
06120 Halle/Salle
D–06120 Halle
```

E–MAIL
creates match codes for e-mail addresses. This definition is inherited from the DE locale (see “DE Match Definitions” on page 79).

NAME
creates match codes for the names of individuals. This definition is inherited from the DE locale (see “DE Match Definitions” on page 79).

ORGANIZATION
creates match codes for the names of organizations. At certain levels of sensitivity, the following values generate the same match codes:

```
Deutsches Inst. f. Wirtschaftsforschung
Institut fr Wirtschaftsforschung
```

ORGANIZATION (WITH LEGAL FORM ANALYSIS)
creates match codes for the names of organizations. The legal form of the name is represented in the match code, which results in a more exact match than ORGANIZATION. At certain levels of sensitivity, the following values generate the same match codes:

```
DIDIER WERKE AG
DIDIER WERKE AKTIENGESELLSCHAFT
```

PHONE
creates match codes for telephone numbers. At some levels of sensitivity, the following values generate the same match codes:

```
+49 6221 123--456
6221 123 - 456
```

TEXT
creates match codes for general text. This definition is inherited from the DE locale (see “DE Match Definitions” on page 79).

DEDEU Parse Definitions

The DEDEU locale contains the following parse definitions:

ADDRESS
parses the “first line” of mailing addresses. Values are sought for the following tokens:

- STREET PREFIX
- STREET NAME
- HOUSE NUMBER
- ADDRESS EXTENSION

For example, the token/value pairs for the value **`Bei den Kornschranned 1`** are:

STREET PREFIX	Bei den

STREET NAME	Kornschrannen
HOUSE NUMBER	1

ADDRESS (GLOBAL)
parses “first lines” of mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ BUILDING NAME
- □ BUILDING NUMBER
- □ STREET
- □ EXTENSION
- □ ADDITIONAL INFO

For example, token/value pairs for the value **`Aachenerstraβe 13, Gebude 2`** are:

BUILDING NUMBER	13
STREET	Aachenerstraβe
EXTENSION	Gebäude 2

CITY - STATE/PROVINCE - POSTAL CODE
Parses the “last line” of mailing addresses. Values are sought for the following tokens:

- □ CITY
- □ REGION
- □ NEIGHBORING CITY
- □ FEDERAL STATE
- □ POSTAL CODE

For example, the token/value pairs of the value **`D-85579 Gut Unterbiberg bei Mnchen, Bayern`** are:

CITY	Gut Unterbiberg
NEIGHBORING CITY	München
FEDERAL STATE	Bayern
POSTAL CODE	85579

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)
parses the “last line” of mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ CITY
- □ STATE/PROVINCE
- □ POSTAL CODE

For example, the token/value pairs for **`D-69118 Hiedelberg Baden-Wrttemberg`** are:

CITY	Hiedelberg
STATE/ PROVINCE	Baden-Württemberg
POSTAL CODE	69118

NAME
parses the names of individuals. This definition is inherited from the DE locale (see "DE Parse Definitions" on page 80).

NAME (GLOBAL)
parses the names of individuals. The tokens in this definition are shared by similar definitions in other locales. This definition is inherited from the DE locale (see "DE Parse Definitions" on page 80).

ORGANIZATION
parses the names of organizations. Values are sought for the following tokens:

- NAME
- LEGAL FORM

For example, the token/value pairs for **`Didier Werke AG`** are:

NAME	Didier Werke
LEGAL FORM	AG

PHONE
parses telephone numbers. Values are sought for the following tokens:

- COUNTRY CODE
- AREA CODE
- BASE NUMBER

For example, the token/value pairs for the value **`+49 6221 123--456`** are:

COUNTRY CODE	+49
AREA CODE	6221
BASE NUMBER	123-456

PHONE (GLOBAL)
parses telephone numbers. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for the value **`+49 6221 123--456`** are:

COUNTRY CODE	+49
AREA CODE	6221
BASE NUMBER	123–456

POSTAL CODE
Parses postal codes. Values are sought for the following tokens:

- COUNTRY CODE
- FIRST TWO POSTAL CODE DIGITS
- LAST THREE POSTAL CODE DIGITS

For example, the token/value pairs for the value **`D-69118`** are:

COUNTRY CODE	D
FIRST TWO POSTAL CODE DIGITS	69
LAST THREE POSTAL CODE DIGITS	118

DEDEU Pattern Analysis Definitions

The DEDEU locale contains the following pattern analysis definitions:

CHARACTER
: generates one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

WORD
: generates one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

DEDEU Standardization Definitions

The DEDEU locale contains the following standardization definitions:

ADDRESS
: standardizes the “first line” of mailing addresses. For example, the street address **`Victoria Straβe 12`** is standardized as **`VIKTORIA STR. 12`**.

CITY
: standardizes the names of cities. This definition is inherited from the DE locale (see “DE Standardization Definitions” on page 81).

CITY - STATE/PROVINCE - POSTAL CODE
: standardizes the “last line” of mailing addresses. For example, the location **`D-48465 Schttorf`** is standardized as **`48465 SCHTTORF`**.

NAME
: standardizes the names of individuals. This definition is inherited from the DE locale (see “DE Standardization Definitions” on page 81).

ORGANIZATION
: standardizes the names of organizations. For example, the name **`Didier Werke Aktg`** is standardized as **`DIDIER WERKE AG`**.

PHONE
: standardizes telephone numbers. For example, the phone number **`+49 6221 123 - 456`** is standardized as **`0049 6221 123-456`**.

EN

Provides inheritable data cleansing definitions for the English language.

The EN locale is a descendant of the L1 locale (see “L1” on page 78). The EN locale is the ancestor the following locales:

- “ENAUS” on page 90.
- “ENGBR” on page 95.
- “ENUSA” on page 101.

Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The EN locale includes:

- Gender analysis definitions, see “EN Gender Analysis Definitions” on page 86.
- Match definitions, see “EN Match Definitions” on page 86.
- Parse definitions, see “EN Parse Definitions” on page 87.
- Standardization definitions, see “EN Standardization Definitions” on page 89.

EN Gender Analysis Definitions

The EN locale includes the following gender analysis definitions:

GENDER

determines gender based on the name of an individual. The determination can be `M` for male, `F` for female, or `U` for unknown. The gender is unknown if gender clues conflict or if a name can be applied to either gender and no other gender clues are provided. For example, the gender determination for the name `Pat Bishop` is `U`, but the determination for the name `Mrs.  Pat Bishop` is `F`.

NAME is the parse definition that is associated with this gender analysis definition.

EN Match Definitions

The EN locale includes the following match definitions:

ACCOUNT NUMBER

creates match codes for account numbers, where the most significant information in those numbers appears in the final characters. Before the match code is created, all characters are reversed, then all non-alphanumeric characters are removed. For example, the input value `288-58-6390` is reversed to give `0936-85-882`. Next, all non-alphanumeric characters are removed to give `093685882`. The match code is then created from this value.

At all levels of sensitivity, the following account numbers create the same match codes:

```
288-58-6390
288 58 6390
#2 8 8 5 8 6 3 9 0
```

ACCOUNT NUMBER (MOST SIGNIFICANT FIRST)

creates match codes for account numbers, where the most significant information in those numbers appears in the initial characters. Before the match code is created, all non-alphanumeric characters are removed. For example, all non-alphanumeric characters are removed from the input value **`288-58-6390`** to give **`288586390`**. The match code is then created using the second value.

At all levels of sensitivity, the following account numbers generate the same match codes:

```
288-58-6390
288 58 6390
#2 8 8 5 8 6 3 9 0
```

DATE (DMY)
: creates match codes for dates that are specified by day, month, and year. For example, the date **`17March1960`** generates the same match code as the date **`17/3/60`**.

DATE (MDY)
: creates match codes for dates that are specified by month, day, and year.

DATE (YMD)
: creates match codes for dates that are specified by year, month, and day.

E-MAIL
: creates match codes for electronic mail addresses. For example, the e-mail address **`JohnQPublic@serviceProvider.com`** generates the same match code as the e-mail address **`JohnQPublic@serviceProvider.net`**.

NAME
: creates match codes for the names of individuals. For example, the name **`Mary Shafer`** generates the same match code, at lower sensitivity levels, as the name **`Mrs.  Marian V. Schaeffer`**.

EN Parse Definitions

The EN locale includes the following parse definitions:

DATE (DMY)
: parses dates that are specified by day, month, and year. Values are sought for the following tokens:

- DAY
- MONTH
- YEAR

For example, the token/value pairs for the date **`24July2004`** are:

DAY	24
MONTH	July
YEAR	2004

DATE (MDY)
: parses dates that are specified by month, day, and year. Values are sought for the following tokens:

- MONTH
- DAY
- YEAR

For example, the token/value pairs for the date **`July/24/2004`** are:

MONTH	July
DAY	24
YEAR	2004

DATE (YMD)
parses dates that are specified by year, month, and day. Values are sought for the following tokens:

- □ YEAR
- □ MONTH
- □ DAY

For example, the token/value pairs for the date **`2004.July.24`** are:

YEAR	2004
MONTH	July
DAY	24

E-MAIL
parses e-mail addresses. This parse definition is inherited from the L1 locale. See "L1 Parse Definitions" on page 78.

NAME
parses the names of individuals. This parse definition is inherited from the EN locale. See "EN Parse Definitions" on page 87.

NAME (GLOBAL)
parses the names of individuals. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ PREFIX
- □ GIVEN NAME
- □ MIDDLE NAME
- □ FAMILY NAME
- □ SUFFIX
- □ TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Dr. Alexander Graham Bell`** are:

NAME PREFIX	Dr.
GIVEN NAME	Alexander
MIDDLE NAME	Graham
FAMILY NAME	Bell

NAME (TWO NAME)
parses input values that contain the names of two individuals. Values are sought for the following tokens:

- □ NAME PREFIX 1
- □ GIVEN NAME 1
- □ MIDDLE NAME 1
- □ FAMILY NAME 1
- □ NAME SUFFIX 1

- NAME APPENDAGE 1
- NAME PREFIX 2
- GIVEN NAME 2
- MIDDLE NAME 2
- FAMILY NAME 2
- NAME SUFFIX 2
- NAME APPENDAGE 2

For example, the token/value pairs for the value **`Mr.  Jonathan Doe II DDS and Mrs.  Jane Alice Doe, MSSW`** are:

NAME PREFIX 1	Mr.
GIVEN NAME 1	Jonathan
FAMILY NAME 1	Doe
NAME SUFFIX 1	II
NAME APPENDAGE 1	DDS
NAME PREFIX 2	Mrs.
GIVEN NAME 2	Jane
MIDDLE NAME 2	Alice
FAMILY NAME 2	Doe
NAME APPENDAGE 2	MSSW

EN Pattern Analysis Definitions

The EN locale includes the following pattern analysis definitions:

CHARACTER
generates one character of analytical information for each character in an input value. This pattern analysis definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

WORD
generates one character of analytical information for each character in an input value. This pattern analysis definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

EN Standardization Definitions

The EN locale includes the following standardization definitions:

Date (DMY)
standardizes dates that are specified by day, month, and year.

Date (MDY)
: standardizes dates that are specified by month, day, and year.

Date (YMD)
: standardizes dates that are specified by year, month, and day.

ENAUS

Provides cleansing definitions for the English language and the region of Australia.

The ENAUS locale is a descendant of the EN locale (see "EN" on page 86). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

Global definitions can be referenced any time that two or more locales are loaded into memory (see "Global Definitions" on page 78).

The ENAUS locale includes:

- Gender analysis definitions, see "ENAUS Gender Analysis Definitions" on page 90.
- Match definitions, see "ENAUS Match Definitions" on page 90.
- Parse definitions, see "ENGBR Parse Definitions" on page 97.
- Pattern analysis definitions, see "ENAUS Pattern Analysis Definitions" on page 94.
- Standardization definitions, see "ENAUS Standardization Definitions" on page 94.

ENAUS Gender Analysis Definitions

The gender definition GENDER is inherited from the EN locale (see "EN Gender Analysis Definitions" on page 86).

ENAUS Match Definitions

The ENAUS locale includes the following match definitions:

ACCOUNT NUMBER
: creates match codes for account numbers that contain their most significant digits at the end. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ACCOUNT NUMBER (MOST SIGNIFICANT FIRST)
: creates match codes for account numbers that contain their most significant digits at the beginning. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ADDRESS
: creates match codes for street addresses. For example, at lower levels of sensitivity, the same match codes are created for the following addresses:

```
Suite 1, 300 Burns Bay Road
300 Byrnes Bay Rd, ste 1
```

ADDRESS is the parse definition that is associated with this match definition.

CITY
: creates match codes for city names. For example, at certain levels of sensitivity, the same match codes are created for the following city names:

```
North Sydney
N SYDN
```

CITY - STATE/PROVINCE - POSTAL CODE
: creates match codes for the “last line” of mailing addresses. A particular value need not contain all three. For example, at certain levels of sensitivity, the same match codes are created for the following values:

```
North Quay, Queensland 4002
N QUay, Qld 4002
```

DATE (DMY),
DATE (MDY),
DATE (YMD)
: these three separate match definitions create match codes for dates; they are inherited from the EN locale (see “EN Match Definitions” on page 86).

E-MAIL
: creates match codes for electronic mail addresses. This match definition is inherited from the EN locale (see “EN Match Definitions” on page 86).

NAME
: creates match codes for the names of individuals. This match definition is inherited from the EN locale (see “EN Match Definitions” on page 86).

ORGANIZATION
: creates match codes for the names of organizations such as corporations, associations, and partnerships. For example, at certain levels of sensitivity, the same match codes are created for the following organizations:

```
Moffat Limited
Moffet
```

PHONE
: creates match codes for telephone numbers. For example, the same match codes are created for the following telephone numbers:

```
Work: 61 02 94280410
61 02--94280410
```

ENAUS Parse Definitions

The ENAUS locale includes the following parse definitions:

ADDRESS
: parses street addresses. Values are sought for the following tokens:

- BUILDING NAME
- STREET NUMBER
- PRE-DIRECTION
- STREET NAME
- STREET TYPE
- POST-DIRECTION
- ADDRESS EXTENSION
- ADDRESS EXTENSION NUMBER

For example, the token/value pairs for the street address **`St George House 4--16 Montgomery Street ST, Basement`** are:

BUILDING NAME	St George House

STREET NUMBER	4–16
STREET NAME	Montgomery
STREET TYPE	Street
POST-DIRECTION	SE
ADDRESS EXTENSION	Basement

ADDRESS (GLOBAL)

parses street addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ BUILDING NAME
- □ BUILDING NUMBER
- □ STREET
- □ EXTENSION
- □ ADDITIONAL INFO

For example, the token/value pairs for the mailing address **`St George House 4--16 Montgomery Street ST, Basement`** are:

BUILDING NAME	St George House
BUILDING NUMBER	4–16
STREET	Montgomery Street SE
EXTENSION	Basement

CITY - STATE/PROVINCE - POSTAL CODE

parses “last-line” locations in mailing addresses. Values are sought for the following tokens:

- □ CITY
- □ STATE
- □ POSTCODE

For example, the token/value pairs for the location **`Lane Cove, NSW 2066`** are:

CITY	Lane Cove
STATE	NSW
POSTCODE	2066

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)

parses locations in mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ CITY
- □ STATE/PROVINCE
- □ POSTAL CODE

For example, the token/value pairs for the location **`North Quay, QLD 4002`** are:

CITY	North Quay
STATE/PROVINCE	QLD

POSTAL CODE	4002

DATE (DMY),
DATE (MDY),
DATE (YMD)
these parse definitions parse dates; they are inherited from the EN locale (see "ENAUS Parse Definitions" on page 91).

E-MAIL
parses electronic mail addresses. This parse definition is inherited from the L1 locale (see "L1 Parse Definitions" on page 78).

NAME
parses the names of individuals. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

NAME (GLOBAL)
parses the names of individuals. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- GIVEN NAME
- MIDDLE NAME
- FAMILY NAME
- SUFFIX
- TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Mr.  Peter Cullen (Justice of the Peace)`** are:

PREFIX	Mr.
GIVEN NAME	Peter
FAMILY NAME	Cullen
TITLE/ ADDITIONAL INFO	Justice of the Peace

NAME (TWO NAME)
parses input values that contain the names of two individuals. This parse definition is inherited from the EN locale (see "EN Parse Definitions" on page 87).

PHONE
parses telephone numbers. Values are sought for the following tokens:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION ID
- EXTENSION
- SUFFIX

For example, the token/value pairs for the telephone number **`Work:  61 02 94280410 ext 44`** are:

PREFIX	Work
COUNTRY CODE	61

AREA CODE	02
BASE NUMBER	94280410
EXTENSION ID	ext
EXTENSION	44

PHONE (GLOBAL)
parses Australian telephone numbers. Values are sought for the following tokens, which are shared by similar parse definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for **`Work:  61 02 94280410 ext 44`** are:

PREFIX	Work
COUNTRY CODE	61
AREA CODE	02
BASE NUMBER	94280410
EXTENSION	ext 44

ENAUS Pattern Analysis Definitions

The ENAUS locale contains the following pattern analysis definitions:

CHARACTER
returns one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

WORD
returns one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

ENAUS Standardization Definitions

The ENAUS locale includes the following standardization definitions:

ADDRESS
standardizes street addresses. For example, the input address **`4--16 Montgomery Street SE`** is standardized as **`4--16 MONTGOMERY ST SE`**.

CITY
standardizes the names of cities. For example, the input value **`Adamstown HEIGHTS`** is standardized as **`ADAMSTOWN HTS`**.

City - State/Province - Postal Code
standardizes the “last line” of mailing addresses. For example, the location **`Lane Cove, New South Wales 2066`** is standardized as **`LANE COVE NSW 2066`**.

DATE (DMY),
DATE (MDY),
DATE (YMD)
standardize dates that are specified in various combinations of day, month, and year. These standardization definitions are inherited from the EN locale (see "EN Standardization Definitions" on page 89).

NAME
standardizes the names of individuals. Names themselves are not changed. For example, the name **`Mike`** is not changed to **`Michael`**. The name **`Cullen, Mister Peter C.`** is standardized as **`MR PETER C. CULLEN`**.

ORGANIZATION
standardizes the names of organizations. For example, the organization name **`Robe River Iron Associates`** is standardized as **`ROBE RIVER IRON ASSOC`**.

PHONE
standardizes telephone numbers. For example, the phone number **`02--94280410 ext 44`** is standardized as **`02 94280410 X 44`**.

ENGBR

Provides cleansing definitions for the English language and the region of Great Britain.

The ENGBR locale is a descendant of the EN locale (see "EN" on page 86). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

Global definitions can be referenced any time that two or more locales are loaded into memory (see "Global Definitions" on page 78).

The ENGBR locale includes:

- □ Gender analysis definitions, see "ENGBR Gender Analysis Definitions" on page 95.
- □ Match definitions, see "ENGBR Match Definitions" on page 95.
- □ Parse definitions, see "ENGBR Parse Definitions" on page 97.
- □ Pattern analysis definitions, see "ENGBR Pattern Analysis Definitions" on page 100.
- □ Standardization definitions, see "ENGBR Standardization Definitions" on page 101.

ENGBR Gender Analysis Definitions

The gender analysis definition GENDER determines gender based on the names of individuals. This definition is inherited from the EN locale (see "EN Gender Analysis Definitions" on page 86).

ENGBR Match Definitions

The ENGBR locale includes the following match definitions:

ACCOUNT NUMBER
creates match codes for account numbers that contain their most significant digits at the end. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ACCOUNT NUMBER (MOST SIGNIFICANT FIRST)
: creates match codes for account numbers that contain their most significant digits at the beginning. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ADDRESS
: creates match codes for street addresses. For example, at lower levels of sensitivity, the same match codes are created for the following addresses:

```
The Chapter House, 32 Upton Rd, Covent Garden
The Chapter House, 32 Upton Road
```

ADDRESS is the parse definition that is associated with this match definition.

CITY
: creation match codes for city names. For example, at certain levels of sensitivity, the same match codes are created for the following city names:

```
Birmingham
bham
```

CITY - STATE/PROVINCE - POSTAL CODE
: creates match codes for the "last line" of mailing addresses. A particular value need not contain all three. For example, at certain levels of sensitivity, the same match codes are created for the following values:

```
Sheffield, South Yorkshire, S8 8TJ
Sheffield South Yorks. S8 8TJ
```

DATE (DMY),
DATE (MDY),
DATE (YMD)
: these three separate match definitions create match codes for dates; they are inherited from the EN locale (see "EN Match Definitions" on page 86).

E-MAIL
: creates match codes for electronic mail addresses. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

NAME
: creates match codes for the names of individuals. For example, at certain levels of sensitivity, the same match codes are created for the following names:

```
Sir Paul McCartney
Paul McArtney
```

ORGANIZATION
: creates match codes for the names of organizations such as corporations, associations, and partnerships. For example, at certain levels of sensitivity, the same match codes are created for the following organizations:

```
The John Howard Centre
Jon Howard Ctr
```

PHONE
: creates match codes for telephone numbers. For example, the same match codes are created for the following telephone numbers:

```
01628 486933
01628--486933 (Work)
```

ENGBR Parse Definitions

The ENGBR locale includes the following parse definitions:

ADDRESS

parses street addresses. Values are sought for the following tokens:

- □ PREFIX
- □ HOUSE NAME
- □ STREET NUMBER
- □ STREET NAME
- □ STREET TYPE
- □ NEIGHBOURHOOD

For example, the token/value pairs for the street address **`The Chapter House, 32 Upton Rd, Covent Garden`** are:

HOUSE NAME	The Chapter House
STREET NUMBER	32
STREET NAME	Upton
STREET TYPE	Rd
NEIGHBOUR HOOD	Covent Garden

ADDRESS (FULL)

parses entire addresses. Values are sought for the following tokens:

- □ PREFIX
- □ HOUSE NAME
- □ STREET NUMBER
- □ STREET NAME
- □ STREET TYPE
- □ NEIGHBOURHOOD
- □ TOWN VILLAGE
- □ POST TOWN
- □ COUNTY
- □ COUNTRY
- □ POST OUTCODE
- □ POST INCODE

For example, the token/value pairs for the full mailing address **`38 Castle Roe Road, Coleraine Northern Ireland BT51 3RL`** are:

STREET NUMBER	33
STREET NAME	Castle Roe
STREET TYPE	Road
TOWN VILLAGE	Coleraine
COUNTRY	Northern Ireland
POST OUTCODE	BT51

POST INCODE	3RL

ADDRESS (GLOBAL)

parses street addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- BUILDING NAME
- BUILDING NUMBER
- STREET
- EXTENSION
- ADDITIONAL INFO

For example, the token/value pairs for the mailing address **`The Chapter House, 32 Upton Rd`** are:

BUILDING NAME	The Chapter House
BUILDING NUMBER	32
STREET	Upton Rd

CITY - STATE/PROVINCE - POSTAL CODE

parses locations in mailing addresses. Values are sought for the following tokens:

- TOWN VILLAGE
- POST TOWN
- COUNTY
- COUNTRY
- POST OUTCODE
- POST INCODE

For example, the token/value pairs for the location **`Sheffield, South Yorkshire, S8 8TJ`** are:

TOWN VILLAGE	Sheffield
COUNTY	South Yorkshire
POST OUTCODE	S8
POST INCODE	8TJ

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)

parses locations in mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- CITY
- STATE/PROVINCE
- POSTAL CODE

For example, the token/value pairs for the location **`Sheffield, South Yorkshire, S8 8TJ`** are:

CITY	Sheffield
STATE/ PROVINCE	South Yorkshire
POSTAL CODE	S8 8TJ

DATE (DMY),
DATE (MDY),
DATE (YMD)
: these parse definitions parse dates; they are inherited from the EN locale (see “EN Parse Definitions” on page 87).

E-MAIL
: parses electronic mail addresses. This parse definition is inherited from the L1 locale (see “L1 Parse Definitions” on page 78).

NAME
: parses the names of individuals. Values are sought for the following tokens:

- NAME PREFIX
- GIVEN NAME
- MIDDLE NAME
- FAMILY NAME
- NAME SUFFIX
- NAME APPENDAGE

For example, the token/value pairs for the name **`Mr Mervyn Allen`**are:

NAME PREFIX	Mr
GIVEN NAME	Mervyn
FAMILY NAME	Allen

NAME (GLOBAL)
: parses the names of individuals. This parse definition is inherited from the EN locale (see “EN Parse Definitions” on page 87).

NAME (TWO NAME)
: parses input values that contain the names of two individuals. If the input value contains only one name, only the first six tokens are used. Note that this definition supersedes the definition of the same name that would otherwise be inherited from the EN locale. Values are sought for the following tokens:

- NAME PREFIX 1
- GIVEN NAME 1
- MIDDLE NAME 1
- FAMILY NAME 1
- NAME SUFFIX 1
- NAME APPENDAGE 1
- NAME PREFIX 2
- GIVEN NAME 2
- MIDDLE NAME 2
- FAMILY NAME 2
- NAME SUFFIX 2
- NAME APPENDAGE 2

For example, the token/value pairs for the value **`Mr.  Jonathan Doe II DDS and Mrs.  Jane Alice Doe, MSSW`** are:

NAME PREFIX 1	Mr.
GIVEN NAME 1	Jonathan

FAMILY NAME 1	Doe
NAME SUFFIX 1	II
NAME APPENDAGE 1	DDS
NAME PREFIX 2	Mrs.
GIVEN NAME 2	Jane
MIDDLE NAME 2	Alice
FAMILY NAME 2	Doe
NAME APPENDAGE 2	MSSW

PHONE
parses telephone numbers. Values are sought for the following tokens:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- SUFFIX

For example, the token/value pairs for the telephone number **`01628 486933 (Work)`** are:

AREA CODE	01628
BASE NUMBER	486933
EXTENSION	Work

PHONE (GLOBAL)
parses telephone numbers. Values are sought for the following tokens, which are shared by similar parse definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for **`01628 486933`** are:

AREA CODE	01638
BASE NUMBER	486933

ENGBR Pattern Analysis Definitions

The ENGBR locale contains the following pattern analysis definitions:

CHARACTER
returns one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

WORD
returns one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

ENGBR Standardization Definitions

The ENGBR locale includes the following standardization definitions:

ADDRESS
standardizes street addresses. For example, the input address **`47 Marlow Road, Marlow, Bucks`** is standardized as **`47 MARLOW RD MARLOW BUCKINGHAMSHIRE`**.

CITY
standardizes the names of cities. For example, the input value **`nhampton`** is standardized as **`NORTHHAMPTON`**.

City - State/Province - Postal Code
standardizes locations, such as those that are found in mailing addresses. For example, the location **`Sheffield, South Yorks., S8 8TJ`** is standardized as **`SHEFFIELD SOUTH YORKSHIRE S8 8TJ`**.

DATE (DMY),
DATE (MDY),
DATE (YMD)
standardize dates that are specified in various combinations of day, month, and year. These standardization definitions are inherited from the EN locale (see "EN Standardization Definitions" on page 89).

NAME
standardizes the names of individuals. Names themselves are not changed. For example, the name **`Mike`** is not changed to **`Michael`**. The name **`Mr.  J.T. Webb, Junior`** is standardized as **`MR J T WEBB JR`**.

ORGANIZATION
standardizes the names of organizations. For example, the organization name **`st. John's Hospital`** is standardized as **`ST JOHNS HOSPITAL`**.

PHONE
standardizes telephone numbers. For example, the phone number **`01628--486933 (Work)`** is standardized as **`06128 486933 WORK`**.

ENUSA

Provides cleansing definitions for the English language and the region of the United States of America.

The ENUSA locale is a descendant of the EN locale (see "EN" on page 86). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

Global definitions can be referenced any time that two or more locales are loaded into memory (see "Global Definitions" on page 78).

The ENUSA locale includes:

- Case definitions, see "ENUSA Case Definitions" on page 102.
- Gender analysis definitions, see "ENUSA Gender Analysis Definitions" on page 102.
- Identification definitions, see "ENUSA Identification Definitions" on page 102.
- Match definitions, see "ENUSA Match Definitions" on page 103.
- Parse definitions, see "ENUSA Parse Definitions" on page 105.
- Pattern analysis definitions, see "ENUSA Pattern Analysis Definitions" on page 107.
- Standardization definitions, see "ENUSA Standardization Definitions" on page 107.

ENUSA Case Definitions

The ENUSA locale includes the following case definitions.

PROPER
: for general text. For example, the character value **`propercase general text`** becomes **`Propercase General Text`**.

PROPER (ADDRESS)
: for street addresses without city, state, and ZIP code. For example, **`420 PARK RIDGE RD`** becomes **`420 Park Ridge Rd`**.

PROPER (CITY - STATE/PROVINCE - POSTAL CODE)
: for mailing addresses with the street address. For example, **`cary, nc 27513`** becomes **`Cary, NC 27513`**.

PROPER (NAME)
: for the names of individuals. For example, **`john DUNHAM`** becomes **`John Dunham`**.

PROPER (ORGANIZATION)
: for the names of organizations. For example, **`sas institute`** becomes **`SAS Institute`**.

ENUSA Gender Analysis Definitions

The ENUSA locale contains the following gender analysis definitions:

GENDER
: generates gender determinations based on the names of individuals. This definition is inherited from the EN locale (see "EN Gender Analysis Definitions" on page 86).

ENUSA Identification Definitions

The ENUSA locale includes the following identification definitions:

CONTACT INFO
: identifies the element of an address that is contained in an input character value. Available return values are defined as follows:

ADDR	street address line 1
CSZ	city, state, and/or ZIP
UNK	unknown
ATTN	attention or addressee line

ADDR2	street address line 2
BLANK	empty input
ORG	organization
IND	individual
ACCT	account information

For example, an input value of **`1619 Lexington Dr.`** returns the value ADDR. An input value of **`New York City`** returns a value of CSZ. An input value of **`C/O Ken Wright`** returns a value of ATTN.

INDIVIDUAL/ORGANIZATION
identifies an input name as belonging to an individual or an organization. Return values can be INDIVIDUAL, ORGANIZATION, or UNKNOWN. For example, an input name of **`SAS Institute, Inc.`** returns the value ORGANIZATION. The input name **`Sandra K. Baker`** returns a value of INDIVIDUAL.

ENUSA Match Definitions

The ENUSA locale includes the following match definitions:

ACCOUNT NUMBER
creates match codes for account numbers that contain their most significant digits at the end. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ACCOUNT NUMBER (MOST SIGNIFICANT FIRST)
creates match codes for account numbers that contain their most significant digits at the beginning. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ADDRESS
creates match codes for street addresses. For example, at lower levels of sensitivity, the same match codes are created for the following addresses:

```
101 Oak Street
101 N. Oak Avenue, Apt. #4-B
```

ADDRESS is the parse definition that is associated with this match definition.

CITY
creates match codes for city names. For example, at all levels of sensitivity, the same match codes are created for the following city names:

```
North Myrtle Beach
n. mertl bch
```

CITY (SCHEME BUILD)
creates match codes for city names. When building schemes with the DQSCHEME procedure, this match definition provides better results than the CITY match definition.

CITY - STATE/PROVINCE - POSTAL CODE
creates match codes for city, state and/or ZIP Code values. A particular value need not contain all three. For example, at lower levels of sensitivity, the same match codes are created for the following values:

```
Raleigh NC 27614-9520
RALEIGH, NORTH CAROLINA
```

DATE (DMY),
DATE (MDY),
DATE (YMD)
: these three separate match definitions create match codes for dates; they are inherited from the EN locale (see "EN Match Definitions" on page 86).

E-MAIL
: creates match codes for electronic mail addresses. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

NAME
: creates match codes for the names of individuals. This match definition is inherited from the EN locale (see "EN Match Definitions" on page 86).

ORGANIZATION
: creates match codes for the names of organizations such as corporations, associations, and partnerships. For example, at all levels of sensitivity, the same match codes are created for the following organization names:

```
SAS Institute, Inc.
SAS
```

ORGANIZATION (SCHEME BUILD)
: creates match codes for organization names. When building schemes with the DQSCHEME procedure, this match definition provides better results than the ORGANIZATION match definition.

PHONE
: creates match codes for telephone numbers. For example, at lower levels of sensitivity, the same match codes are created for the following telephone numbers:

```
1.919.674.2153 extension 999
674-2153
```

STATE
: creates match codes for state names and abbreviations. For example, at all levels of sensitivity, the same match codes are created for the following state names:

```
North Carolina
N.C.
```

STATE (SCHEME BUILD)
: creates match codes for state names and abbreviations. When building schemes with the DQSCHEME procedure, this match definition provides better results than the STATE match definition.

TEXT
: creates match codes for general text. For example, at lower levels of sensitivity, the same match codes are created for the following text:

```
Honda Accord four door
Honda Accord, 4-door Sedan
```

TEXT (SCHEME BUILD)
: creates match codes for general text. When building schemes with the DQSCHEME procedure, this match definition provides better results than the TEXT match definition.

ZIP
: creates match codes for United States Postal Service ZIP Codes. For example, at lower levels of sensitivity, the same match codes are created for the following ZIP Codes:

```
53400
53403-4001
```

ENUSA Parse Definitions

The ENUSA locale includes the following parse definitions:

ADDRESS

parses street addresses. Values are sought for the following tokens:

- STREET NUMBER
- PRE-DIRECTION
- STREET NAME
- STREET TYPE
- POST-DIRECTION
- ADDRESS EXTENSION
- ADDRESS EXTENSION NUMBER

For example, the token/value pairs for the street address **`4001 E. Independence Blvd.  SW, Suite 301`** are:

STREET NUMBER	4001
PRE-DIRECTION	E.
STREET NAME	Independence
STREET TYPE	Blvd.
POST-DIRECTION	SW
ADDRESS EXTENSION	Suite
ADDRESS EXTENSION NUMBER	301

ADDRESS (GLOBAL)

parses street addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- BUILDING NAME
- BUILDING NUMBER
- STREET
- EXTENSION
- ADDITIONAL INFO

For example, the token/value pairs for the mailing address **`4001 E. Independence Blvd.  SW, Suite 301`** are:

BUILDING NUMBER	4001
STREET	E. Independence Blvd. SW
EXTENSION	Suite 301

CITY - STATE/PROVINCE - POSTAL CODE

parses locations in mailing addresses. Values are sought for the following tokens:

- CITY
- STATE
- ZIP

For example, the token/value pairs for the location **`Cary, NC 27513--0001`** are:

CITY	Cary
STATE	NC
ZIP	27513-0001

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)
parses locations in mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- CITY
- STATE/PROVINCE
- POSTAL CODE

For example, the token/value pairs for the location **`Cary, NC 27513-0001`** are:

CITY	Cary
STATE/ PROVINCE	NC
POSTAL CODE	27513-0001

DATE (DMY),
DATE (MDY),
DATE (YMD)
these parse definitions parse dates; they are inherited from the EN locale (see “EN Parse Definitions” on page 87).

E-MAIL
parses electronic mail addresses. This parse definition is inherited from the L1 locale (see “L1 Parse Definitions” on page 78).

NAME
parses the names of individuals. This parse definition is inherited from the EN locale (see “EN Parse Definitions” on page 87).

NAME (GLOBAL)
parses the names of individuals. This parse definition is inherited from the EN locale (see “EN Parse Definitions” on page 87).

PHONE
parses telephone numbers. Values are sought for the following tokens:

- PREFIX
- COUNTRY CODE
- AREA CODE
- EXCHANGE
- STATION
- EXTENSION ID
- EXTENSION
- SUFFIX

For example, the token/value pairs for the telephone number **`919.677.8000 ext. 111`** are:

AREA CODE	919
EXCHANGE	677
STATION	8000
EXTENSION ID	ext.
EXTENSION	111

PHONE (GLOBAL)

parses telephone numbers. Values are sought for the following tokens, which are shared by similar parse definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for `919.677.8000 ext.  111` are:

AREA CODE	919
BASE NUMBER	6778000
EXTENSION	ext.111

ZIP

parses ZIP codes. Values are sought for the following tokens:

- ZIP
- ZIP ADD-ON

The token/value pairs for the ZIP code `27614-9520` are:

ZIP	27614
ZIP ADD-ON	9520

ENUSA Pattern Analysis Definitions

The ENUSA locale includes the following pattern analysis definitions:

CHARACTER

generates one character of analytical information for each character in the input character value. This pattern analysis definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

WORD

generates one character of analytical information for each word in the input character value. This pattern analysis definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

ENUSA Standardization Definitions

The ENUSA locale includes the following standardization definitions:

ADDRESS

standardizes street addresses. Standardized addresses are changed to mixed-case, insignificant punctuation and extra blank spaces are removed, and certain abbreviations are changed to a standard representation. For example, the input address **`Suite 401, 1619 WEST LEXINGTON DRIVE`** is standardized as **`1619 W. Lexington Dr., Suite 401`**

CITY
: standardizes the names of cities. Standardized city names are changed to initial capital letters, insignificant punctuation and extra blank spaces are removed, and certain abbreviations are spelled out. For example, the input value **`NY`** is standardized as **`NEW YORK`**.

CITY - STATE/PROVINCE - POSTAL CODE
: standardizes locations, such as those that are found in mailing addresses. Standardized city names are changed to initial capital letters, insignificant punctuation and extra blank spaces are removed, certain abbreviations are spelled out, and certain abbreviations are retained. For example, the location **`cary nc 27513`** is standardized as **`Cary, NC 27513`**.

DATE (DMY),
DATE (MDY),
DATE (YMD)
: standardize dates. These standardization definitions are inherited from the EN locale (see "EN Standardization Definitions" on page 89).

NAME
: standardizes the names of individuals. Standardized names are changed to mixed case, insignificant punctuation is removed, prefixes, suffixes, and appendages are changed to a standard representation, and the format of the name is standardized. No attempt is made to alter any part of the name itself. For example, the name **`Mike`** is not changed to **`Michael`**.
For example, the name **`BAKER, MISTER MIKE R.`** is standardized as **`Mr Mike R Baker`**.

ORGANIZATION
: standardizes the names of organizations. Standardized names are changed to mixed case, insignificant punctuation is removed, and various organizational abbreviations are changed to a standard representation. For example, the organization name **`i.b.m.  corporation`** is standardized as **`IBM Corp`**.

PHONE
: standardizes telephone numbers. Standardized phone numbers are changed to uppercase and all punctuation and blank spaces, other than dashes (-) is excluded. For example, the phone number **`(919) 555 1234 ext.  9`** is standardized as **`919--555--1234 EXT 9`**.

STATE (FULL NAME)
: standardizes state names or abbreviations into a full state name. Insignificant punctuation is removed and initial letters are capitalized. For example, the state names **`N.C.`** and **`n.  carolina`** are both standardized as **`North Carolina`**.

STATE (TWO LETTER)
: standardizes state names or abbreviations into two-letter state postal codes. For example, the state names **`N.C.`** and **`n.  carolina`** are both standardized to **`NC`**.

ZIP
: standardizes ZIP codes. Any alphabetic characters are removed, any blank spaces are removed, and a dash is inserted if a ZIP add-on is supplied. For example, the ZIP code **`27513 0001`** is standardized as **`27513-0001`**.

IT

Provides inheritable data cleansing definitions for the Italian language.

The IT locale is a descendant of the L1 locale (see “L1” on page 78). The IT locale is the ancestor the ITITA locale (see “ITITA” on page 111). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The IT locale includes:

- □ Gender analysis definitions, see “IT Gender Analysis Definitions” on page 109.
- □ Match definitions, see “IT Match Definitions” on page 109.
- □ Parse definitions, see “IT Parse Definitions” on page 110.
- □ Pattern analysis definitions, see “IT Pattern Analysis Definitions” on page 111.
- □ Standardization definitions, see “IT Standardization Definitions” on page 111.

IT Gender Analysis Definitions

The IT locale contains the following gender analysis definitions:

GENDER
: determines gender based on the name of an individual. The determination can be **M** for male, **F** for female, or **U** for unknown. The gender is unknown if gender clues conflict or if a name can be applied to either gender and no other gender clues are provided. For example, the name **Stefania Barone** generates the gender determination **F**.

IT Match Definitions

The IT locale contains the following match definitions:

CITY
: creates match codes for the names of cities. At certain levels of sensitivity, the following city names generate the same match codes:

```
Roma
roma
```

E-MAIL
: creates match codes for e-mail addresses. At certain levels of sensitivity, the following e-mail addresses generate the same match codes:

```
itwg@itwg.com
ITWG@itwg.com
```

NAME
: creates match codes for the names of individuals. At certain levels of sensitivity, the following names generate the same match codes:

```
Stefania Barone
Stefanya Barona
```

ORGANIZATION
: creates match codes for the names of organizations. At certain levels of sensitivity, the following names generate the same match codes:

```
INDUSTRIA BIELLESE
INDUSTRIA BELLESE
```

TEXT
: creates match codes for general text. At certain levels of sensitivity, the following values generate the same match codes:

```
Crea la mappa
Crea a la mappe
```

IT Parse Definitions

The IT locale contains the following parse definitions:

NAME
: parses the names of individuals. Values are sought for the following tokens:

- PREFIX
- GIVEN NAME
- FAMILY NAME
- NAME APPENDAGE
- NAME EXTENSION

For example, the token/value pairs for the name **`Dr. Mario Rossi, Architetto`** are:

PREFIX	Dr.
GIVEN NAME	Mario
FAMILY NAME	Rossi
NAME APPENDAGE	Architetto

NAME (GLOBAL)
: parses the names of individuals. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- GIVEN NAME
- MIDDLE NAME
- FAMILY NAME
- SUFFIX
- TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Dr. Mario Rossi, Architetto`** are:

PREFIX	Dr.
GIVEN NAME	Mario
FAMILY NAME	Rossi
TITLE/ ADDITIONAL INFO	Architetto

ORGANIZATION
parses the names of organizations. Values are sought for the following tokens:

- NAME
- LEGAL FORM
- ADDITIONAL INFO

For example, the token/value pairs for **`MOC MEDITERRANEO S.R.L.`** are:

NAME	MOC MEDITERRANEO
LEGAL FORM	S.R.L.

IT Pattern Analysis Definitions

The IT locale contains the following pattern analysis definitions:

CHARACTER
generates one character of analytical information for each character in the input character value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

WORD
generates one character of analytical information for each word in the input character value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

IT Standardization Definitions

The IT locale contains the following standardization definitions:

CITY
standardizes the names of cities. For example, **`Castel di Sangro`** is standardized as **`CASTEL DI SANGRO`**.

NAME
standardizes the names of individuals. For example, **`Roberto Benigni`** is standardized as **`ROBERTO BENIGNI`**.

ORGANIZATION
standardizes the names of organizations. For example, **`CONSULENZE AMBIENTALI S,P,A,`** is standardized as **`CONSULENZE AMBIENTALI SPA`**.

ITITA

Provides cleansing definitions for the Italian language and the region of Italy.

The ITITA locale is a descendant of the IT locale (see "IT" on page 109). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

Global definitions can be referenced any time that two or more locales are loaded into memory (see "Global Definitions" on page 78).

The ITITA locale includes:

- Gender analysis definitions, see "ITITA Gender Analysis Definitions" on page 112.

- □ Match definitions, see "ITITA Match Definitions" on page 112.
- □ Parse definitions, see "ITITA Parse Definitions" on page 113.
- □ Pattern analysis definitions, see "ITITA Pattern Analysis Definitions" on page 115.
- □ Standardization definitions, see "ITITA Standardization Definitions" on page 115.

ITITA Gender Analysis Definitions

The ITITA locale contains the following gender analysis definitions:

GENDER
: determines gender based on the names of individuals. This definition is inherited from the IT locale (see "IT Gender Analysis Definitions" on page 109).

ITITA Match Definitions

The ITITA locale contains the following match definitions:

ADDRESS
: creates match codes for the "first line" of mailing addresses. At certain levels of sensitivity, the following values generate the same match codes:

```
Via Dei Setaioli 17/10 Blocco 10
Via Dei Setaioli 17-10 Blocco 10
```

ADDRESS (STAND-ALONE)
: creates match codes for entire mailing addresses.

CITY
: creates match codes for the names of cities. This definition is inherited from the IT locale (see "IT Match Definitions" on page 109).

CITY - STATE/PROVINCE - POSTAL CODE
: creates match codes for the "last line" of mailing addresses. At certain levels of sensitivity, the following locations generate the same match code:

```
Sicilia, PA 90100
Sicillia, PA 90100
```

E-MAIL
: creates match codes for e-mail addresses. This definition is inherited from the IT locale (see "IT Match Definitions" on page 109).

NAME
: creates match codes for the names of individuals. This definition is inherited from the IT locale (see "IT Match Definitions" on page 109).

ORGANIZATION
: creates match codes for organization names. At certain levels of sensitivity, the following names generate the same match codes:

```
Servizi Tecnologie Ambientali S.P.A.
Servizi Tecnologie Ambientali SPA
```

PHONE
: creates match code for telephone numbers. At certain levels of sensitivity, the following values generate the same match codes:

```
02-809662
028 - 09662
```

TEXT
: creates match codes for general text. This definition is inherited from the IT locale (see "IT Match Definitions" on page 109).

ITITA Parse Definitions

The ITITA locale contains the following parse definitions:

ADDRESS

parses the "first line" of mailing addresses. Values are sought for the following tokens:

- STREET TYPE
- ADDITIONAL STREET TYPE
- STREET NAME
- STREET NUMBER
- STREET NUMBER ADDITIONAL INFO
- FIRST EXTENSION
- SECOND EXTENSION
- THIRD EXTENSION

For example, the token/value pairs for the value **`via Mario Rossi 5/A palazzo Martini angolo via Marinetti`** are:

STREET TYPE	via
STREET NAME	Mario Rossi
STREET NUMBER	5
TREE NUMBER ADDITIONAL INFO	A
FIRST EXTENSION	palazzo Martini
SECOND EXTENSION	angolo via Marinetti

ADDRESS (GLOBAL)

parses the "first line" of mailing addresses. Values are sought for the following tokens, which are shared by similar definition in other locales:

- BUILDING NAME
- BUILDING NUMBER
- STREET
- EXTENSION
- ADDITIONAL INFO

For example, the token/value pairs for **`VIA DEI SETAIOLI 17 BLOCCO 10`** are:

BUILDING NUMBER	17
STREET	VIA DEI SETAIOLI
EXTENSION	BLOCCO 10

CITY - STATE/PROVINCE - POSTAL CODE

parses the "last line" of mailing addresses. Values are sought for the following tokens:

- CITY

- PROVINCE
- STATE
- CAP

For example, the token/value pairs for **`Navelli, 67020 L'Aquila`** are:

CITY	Navelli
PROVINCE	L'Aquila
CAP	67020

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)
parses the "last line" of mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- CITY
- STATE/PROVINCE
- POSTAL CODE

For example, the token/value pairs for **`Roma, RM 00100`** are:

CITY	Roma
STATE/ PROVINCE	RM
POSTAL CODE	00100

NAME
parses the names of individuals. This definition is inherited from the IT locale (see "IT Parse Definitions" on page 110).

NAME (GLOBAL)
parses the names of individuals, using a set of tokens that are shared by similar definitions in other locales. This definition is inherited from the IT locale (see "IT Parse Definitions" on page 110).

PHONE
parses telephone numbers. Values are sought for the following tokens:

- COUNTRY CODE
- PREFIX
- BASE NUMBER

For example, the token/value pairs for **`39 347/09987676`** are:

COUNTRY CODE	39
PREFIX	347
BASE NUMBER	09987676

PHONE (GLOBAL)
parses telephone numbers. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for **`06-448991`** are:

AREA CODE	06
BASE NUMBER	448991

ITITA Pattern Analysis Definitions

The ITITA locale contains the following pattern analysis definitions:

CHARACTER
: returns one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see“L1 Pattern Analysis Definitions” on page 79).

WORD
: returns one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

ITITA Standardization Definitions

The ITITA locale contains the following standardization definitions:

ADDRESS
: standardizes the “first line” of mailing addresses. For example, **`v.  mario rossi 5`** is standardized as **`VIA MARIO ROSSI , 5`**.

CITY
: standardizes the names of cities. This definition is inherited from the IT locale (see “IT Standardization Definitions” on page 111).

CITY - STATE/PROVINCE - POSTAL CODE
: standardizes the “last line” of mailing addresses. For example, **`Navelli, 67020 L'Aquila`** is standardized as **`NAVELLI 67020 AQ`**.

NAME
: standardizes the names of individuals. This definition is inherited from the IT locale (see “IT Standardization Definitions” on page 111).

ORGANIZATION
: standardizes the names of organizations. This definition is inherited from the IT locale (see “IT Standardization Definitions” on page 111).

PHONE
: standardizes telephone numbers. For example, **`39 347/09987676`** is standardized as **`0039 347 09987676`**.

NL

Provides inheritable data cleansing definitions for the Dutch language.

The NL locale is a descendant of the L1 locale (see “L1” on page 78). The NL locale is the ancestor the NLNLD locale (see “NLNLD” on page 116). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

The NL locale includes:

- □ Parse definitions, see "NL Parse Definitions" on page 116.
- □ Pattern analysis definitions, see "NL Pattern Analysis Definitions" on page 116.

NL Parse Definitions

The NL locale contains the following parse definitions:

E-MAIL
: parses the addresses of electronic mail. This parse definition is inherited from the L1 locale (see "L1 Parse Definitions" on page 78).

NL Pattern Analysis Definitions

The NL locale contains the following pattern analysis definitions:

CHARACTER
: generates one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

WORD
: generates one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see "L1 Pattern Analysis Definitions" on page 79).

NLNLD

Provides cleansing definitions for the Dutch language and the region of The Netherlands.

The NLNLD locale is a descendant of the NL locale (see "NL" on page 115). Descendant locales inherit definitions from ancestor locales. Inherited definitions may be superseded by local definitions.

Global definitions can be referenced any time that two or more locales are loaded into memory (see "Global Definitions" on page 78).

The NLNLD locale includes:

- □ Match definitions, see "NLNLD Match Definitions" on page 116.
- □ Parse definitions, see "NLNLD Parse Definitions" on page 117.
- □ Pattern analysis definitions, see "NLNLD Pattern Analysis Definitions" on page 120.
- □ Standardization definitions, see "NLNLD Standardization Definitions" on page 120.

NLNLD Match Definitions

The NLNLD locale contains the following match definitions:

ADDRESS
: creates match codes for the "first line" of mailing addresses. At certain levels of sensitivity, the following street addresses generate the same match codes:

```
Strawinskylaan 3075
Srtawinskyln 3075
```

CITY
: creates match codes for the names of cities. At certain levels of sensitivity, the following city names generate the same match codes:

```
AMSTERDAM
z. amsterdam
```

E-MAIL
: creates match codes for electronic mail addresses. At certain levels of sensitivity, the following e-mail addresses generate the same match codes:

```
mail@filtermat.nl
MAIL@filtermat.nl
```

NAME
: creates match codes for the names of individuals. At certain levels of sensitivity, the following names generate the same match codes:

```
Peter von den Hoogenband
Pieter v Hoogenband
```

NAME (INITIALS)
: creates match codes based on the initial letters of the names of individuals. At certain levels of sensitivity, the following names generate the same match codes:

```
Peter von den Hoogenband
P von den Hoogenband
```

NAME (NO VOWEL DISTINCTION)
: creates match codes for the names of individuals by ignoring all of the vowels in those names. At certain levels of sensitivity, the following names generate the same match codes:

```
Wouter Arts
Wuter Erts
```

ORGANIZATION
: creates match codes for the names of organizations. At certain levels of sensitivity, the following organization names generate the same match codes:

```
Essent Energie Noord N.V.
ESSENT ENERGIE
```

PHONE
: creates match codes for telephone numbers. At certain levels of sensitivity, the following phone numbers generate the same match codes:

```
0317-477477
0317 477 477
```

NLNLD Parse Definitions

The NLNLD locale contains the following parse definitions:

ADDRESS
: parses the “first line” of mailing addresses. Values are sought for the following tokens:

- STREET
- NUMBER

- □ ADDRESS EXTENSION
- □ ADDITIONAL INFO

For example, the token/value pairs for **`Gebouw 3, Nachtwachtlaan 20`** are:

STREET	Natchtwachtlaan
NUMBER	20
ADDRESS EXTENSION	Gebouw 3

ADDRESS (GLOBAL)
parses the “first line” of mailing addresses. Values are sought for the following tokens, which are shared by similar definition in other locales:

- □ BUILDING NAME
- □ BUILDING NUMBER
- □ STREET
- □ EXTENSION
- □ ADDITIONAL INFO

For example, the token/value pairs for **`Strawinskylaan 3075`** are:

BUILDING NUMBER	3075
STREET	Strawinskylaan

CITY - STATE/PROVINCE - POSTAL CODE
parses the “last line” of mailing addresses. Values are sought for the following tokens:

- □ POSTAL CODE NUMBER
- □ POSTAL CODE CHARACTERS
- □ TOWN
- □ TOWN PART
- □ PROVINCE

For example, the token/value pairs for **`1270 EB Huizen`** are:

POSTAL CODE NUMBER	1270
POSTAL CODE CHARACTERS	EB
TOWN	Huizen

CITY - STATE/PROVINCE - POSTAL CODE (GLOBAL)
parses the “last line” of mailing addresses. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ CITY
- □ STATE/PROVINCE
- □ POSTAL CODE

For example, the token/value pairs for **`1077 ZX Amsterdam`** are:

CITY	Amsterdam
POSTAL CODE	1077 ZX

NAME

parses the names of individuals. Values are sought for the following tokens:

- □ PREFIX
- □ GIVEN NAME
- □ MIDDLE NAME
- □ FAMILY NAME
- □ SUFFIX
- □ TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Ruud Van Nistelrooy, Forward`** are:

GIVEN NAME	Ruud
FAMILY NAME	Van Nistelrooy
TITLE/ ADDITIONAL INFO	Forward

NAME (GLOBAL)

parses the names of individuals. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- □ PREFIX
- □ GIVEN NAME
- □ MIDDLE NAME
- □ FAMILY NAME
- □ SUFFIX
- □ TITLE/ADDITIONAL INFO

For example, the token/value pairs for the name **`Ruud Van Nistelrooy, Forward`** are:

GIVEN NAME	Ruud
FAMILY NAME	Van Nistelrooy
TITLE/ ADDITIONAL INFO	Forward

ORGANIZATION

parses the names of organizations. Values are sought for the following tokens:

- □ NAME
- □ LEGAL FORM
- □ NOISE WORD NAME

For example, the token/value pairs for **`Van Helden Reklameartikelen BV`** are:

NAME	Van Helden Reklameartikelen
LEGAL FORM	BV

PHONE

parses telephone numbers. Values are sought for the following tokens:

- □ PREFIX
- □ COUNTRY CODE
- □ REGION CODE

- BASE NUMBER
- EXTENSION WORD
- EXTENSION NUMBER
- ADDITIONAL INFO

For example, the token/value pairs for **`31493313006`** are:

COUNTRY CODE	31
REGION CODE	049
BASE NUMBER	3313006

PHONE (GLOBAL)
parses telephone numbers. Values are sought for the following tokens, which are shared by similar definitions in other locales:

- PREFIX
- COUNTRY CODE
- AREA CODE
- BASE NUMBER
- EXTENSION

For example, the token/value pairs for **`0317 477 477`** are:

AREA CODE	0317
BASE NUMBER	477477

NLNLD Pattern Analysis Definitions

The NLNLD locale contains the following pattern analysis definitions:

CHARACTER
generates one character of analytical information for each character in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

WORD
generates one character of analytical information for each word in the input value. This definition is inherited from the L1 locale (see “L1 Pattern Analysis Definitions” on page 79).

NLNLD Standardization Definitions

The NLNLD locale contains the following standardization definitions:

ADDRESS
standardizes the “first line” of mailing addresses. For example, the street address **`Strawinskyln 3075`** is standardized as **`STRAWINSKYLN 3075`**.

ADDRESS (SHORT STREET TYPE)
standardizes the “first line” of mailing addresses, using abbreviated street types. For example, the street address **`Markwegzuid 2a`** is standardized as **`MARKWEGZUID 2 A`**.

CITY - STATE/PROVINCE - POSTAL CODE
: standardizes the “last line” of mailing addresses. For example, the postal region **`2625 BV Delft Zuid Holland`** is standardized as **`2625 BV DELFT ZH`**.

NAME
: standardizes the names of individuals. For example, the name **`Marco v Basten`** is standardized as **`MARCO VAN BASTEN`**.

ORGANIZATION
: standardizes the names of organizations. For example, **`SAS Institute b.v.`** is standardized as **`SAS INST BV`**.

PHONE
: standardizes telephone numbers. For example, the phone number **`31493313006`** is standardized as **`31 3313006 049`**.

CHAPTER 8 System Options

System Options for SAS DATA Quality Server

The SAS Data Quality Server software provides two system options, both of which must be specified before you begin to cleanse data. The DQLOCALE= system option specifies an ordered list of locales. The DQSETUPLOC= system option specifies the location of the SAS Data Quality Server setup file.

To specify values for the DQLOCALE= and DQSETUPLOC= system options, use the %DQLOAD AUTOCALL macro, as described in "%DQLOAD AUTOCALL Macro" on page 41.

Note: It is not recommended that you specify these system options by any means other than invoking the AUTOCALL macro %DQLOAD. Failure to use %DQLOAD or misapplied use of default settings for these system options can result in data that is cleansed with inappropriate locales. △

The data quality system options can be referenced by the OPTIONS procedure by specifying GROUP=DATAQUALITY.

DQLOCALE=

Specifies an ordered list of locales.

Requirements: You must specify at least one locale.

Syntax

DQLOCALE=(*locale1 <locale2 ...localeN>*)

locale1 <locale2 ...localeN>
specifies an ordered list of locales. The list determines how the data is cleansed. Locales are applied to the data in the order in which they are specified. All locales in the list must exist in the Quality Knowledge Base.

Details

The DQLOCALE= system option identifies the locales that will be referenced during data cleansing. The order of the locales in the list affects the locale matching scheme of the DQMATCH procedure.

Unlike other system options, the value of the DQLOCALE= system option must be loaded into memory. Normally, system option values go into the system options table only. Since the locales that are specified with this option must also be loaded into memory, always set the value of this system option by invoking the AUTOCALL macro %DQLOAD. This macro takes as its arguments the values for the DQLOCALE= and DQSETUPLOC= system options.

Note: It is recommended that you invoke the AUTOCALL macro %DQLOAD at the beginning of each data cleansing program or session. Failure to do so may generate unintended output. △

SAS specifies no default value for the DQLOCALE= system option.

It is recommended that you *not* use an AUTOEXEC to load default locales when you invoke SAS. Loading default locales can allow you to apply the wrong locales to your data, which generates unintended output. Loading default locales also wastes resources when you are not cleansing data. Instead of loading default locales, invoke the %DQLOAD macro at the beginning of each data cleansing program or session, as described in "%DQLOAD AUTOCALL Macro" on page 41.

DQSETUPLOC=

Specifies the location of the SAS Data Quality Server setup file.

Syntax

DQSETUPLOC="*file-specification*"

Details

The DQSETUPLOC= system option specifies the location of the SAS Data Quality Server setup file. By default, that setup file is named DQSETUP.TXT. The setup file identifies the location of the Quality Knowledge Base, which contains the locales that are used to cleanse data. Be sure to change the value of this option with the AUTOCALL macro %DQLOAD if you:

- □ Use a different Quality Knowledge Base.
- □ Move the Quality Knowledge Base.
- □ Change the location of the setup file.

If you move the Quality Knowledge Base you need to change the contents of the setup file. Note that if you move the Quality Knowledge Base, you must not change the name of the last storage locations or the name of the files themselves. The contents of the setup file must be updated accordingly.

To ensure that you are using the correct Quality Knowledge Base, always set the DQSETUPLOC= system option by invoking the AUTOCALL macro %DQLOAD. This macro takes as its arguments the values for the DQSETUPLOC= and DQLOCALE= system options, as described in "%DQLOAD AUTOCALL Macro" on page 41.

Note: It is recommended that you invoke the AUTOCALL macro %DQLOAD at the beginning of each data cleansing program or session. Failure to do so may generate unintended output. △

APPENDIX 1 Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *SAS ETL Studio: User's Guide*
- *SAS Companion for UNIX Environments*
- *SAS Companion for z/OS*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **`sasbook@sas.com`**
Web address: **`support.sas.com/pubs`**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

analysis data set
in SAS data quality, a SAS output data set that provides information on the degree of divergence in specified character values.

Blue Fusion Data format
a file format for schemes that can be created and applied in data quality software from SAS and from DataFlux (a SAS company).

blank value
in SAS data quality, an empty or missing value in an input character variable.

case definition
a part of a locale that is referenced during data cleansing to impose on character values a consistent usage of uppercase and lowercase letters.

cluster
in SAS data quality, a set of character values with the same match code.

compound match code
a match code that consists of a concatentation of match codes from values from two or more input character variables in the same observation. A delimiter can be specified to separate the individual match codes in the concatenation. See also single-word match code and multiple-word match code.

data analysis
in SAS data quality, a process that evaluates and reports on the degree of divergence in specified character values. See also analysis data set.

data cleansing
a process that increases the value of character data through data standardization, data transformation, and removal of blank values.

data quality
refers to the relative value of data, based on the accuracy of the knowledge that can be generated using that data. High quality data is consistent, accurate, unambiguous, and efficient when processed. See also SAS data quality.

data standardization
a data cleansing process that references standardization definitions to change character values to a single specified format.

data transformation
in SAS data quality, a data cleansing process that applies a scheme to a specified character variable. The scheme creates match codes internally to create clusters. All values in each cluster are then transformed to the standardization value that is specified in the scheme for each cluster.

definition
in SAS data quality, a part of a locale that is referenced during data cleansing. See also case, guess, gender, identification, match, parse, and standardization definition.

delimited character value
a value that contains delimiters that were inserted by a SAS Data Quality Server function, where that function referenced a parse definition in a locale.

delimiter
in SAS data quality, a character value that separates words in a character value that has been parsed. A delimiter character can also be optionally inserted between individual match codes in a compound match code.

gender definition
a part of a locale that is referenced during data analysis and data cleansing to determine gender based on the names of individuals.

guess definition
a part of a locale that is referenced during the selection of the locale in the locale list that is the best choice for use in the analysis or cleansing of the specified character values.

identification definition
a part of a locale that is referenced during data analysis or data cleansing to determine categories for character values.

locale
in SAS data quality, a set of definitions in a quality knowledge base that are specific to a national language and region. The definitions are referenced during data analysis and data cleansing.

locale definition
See definition.

locale list
an ordered list of locales that is loaded into memory prior to data analysis or data cleansing. The first locale in the list is the default locale.

match code
an encoded version of a character value that is created as a basis for data analysis and data cleansing. Match codes are used to cluster and compare character values. See also sensitivity.

match definition
a part of a locale that is referenced during the creation of match codes. Each match definition is specific to a category of data content. In the ENUSA locale, for example, match definitions are provided for names, e-mail addresses, and street addresses, among others. See also sensitivity.

multiple-word match code
a match code that is created for a single input character value that contains blank spaces. Separate match codes are created for each word. The separate match codes are concatenated into a single match code. See also single-word match code and compound match code.

parse
in SAS data quality, a process that inserts into a character value a series of delimiters, as dictated by a specified parse definition.

parse definition
a part of a locale that is referenced during the parsing of character values. The parse definition specifies the number and location of the delimiters that are inserted during parsing. The location of the delimiters depends on the content of the character values. See also token.

quality knowledge base
a collection of locales and other information that is referenced during data analysis and data cleansing.

SAS data quality
refers to a collection of SAS software products that are used to increase the value of data through data analysis and data cleansing.

scheme
in SAS data quality, a reusable collection of match codes and standardization values that is applied to input character values for the purposes of transformation or analysis. Schemes can be created in Blue Fusion Data format or SAS format.

sensitivity
in SAS data quality, a value that specifies the degree of complexity in newly created match codes. A higher sensitivity value results in greater match code complexity, which in turn results in a larger number of clusters, with fewer members in each cluster.

single–word match code
a match code that is created for a single input character value that contains no blank spaces. See also multiple-word match code and compound match code.

standardize
in SAS data quality, to impose a specified format on character values using a standardization definition.

standardization definition
a part of a locale that is referenced during data cleansing to impose a specified format on character values.

token
in SAS data quality, identifies by name a word in a delimited character value. Parsing functions in the SAS Data Quality Server software can insert or remove a word from a character value by specifying a token name. The software locates the word by counting delimiters, up to the number of delimiters that is associated with the specified token.

standardization value
in SAS data quality, the value in a scheme that is applied to the input character value(s) that have a match code that is identical to the match code in the scheme that is associated with the standardization value.

Index

Your Turn

If you have comments or suggestions about *SAS® Data Quality Server 9.1: Reference*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: **`yourturn@sas.com`**

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: **`suggest@sas.com`**