SAS® Data Quality 3.5 and SAS® 9.4 Data Quality Server: Language Reference

SAS® Documentation
September 8, 2020

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *SAS® Data Quality 3.5 and SAS® 9.4 Data Quality Server: Language Reference*. Cary, NC: SAS Institute Inc.

SAS® Data Quality 3.5 and SAS® 9.4 Data Quality Server: Language Reference

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

August 2020

9.4_3.5-P2:dqclref

Contents

What's New in SAS Data Quality

Overview

SAS 9.4 Data Quality Server and SAS Data Quality 3.5 provide the following enhancements:

- Critical Changes
- DS2 support for some data quality functions
- Profiling capabilities through PROC DATAMETRICS

SAS 9.4M5 Data Quality Server and SAS Data Quality 3.4 provide the following enhancements:

- Profile and categorize data with new functions for identification analysis
- Combined documentation for SAS Data Quality 3.4 and SAS 9.4M5 Data Quality Server
- Perform data quality operations on data in CAS
- Execute programs in CAS that contain data quality functions
- Use a new function to obtain a confidence score for locale guessing
- Interoperate with SSL-enabled DataFlux Data Management Server
- List the locales supported by the SAS QKB with the DQLOCLST procedure
- Synchronize results with DataFlux Data Management software

Critical Changes

If you are using any of the following SAS products, see the corresponding topic for those products in the *SAS® 9.4 Guide to Software Updates and Product Changes*.

- SAS Business Data Network
- SAS Visual Process Orchestration
- SAS Data Management Console
- DataFlux Web Studio and Server
- DataFlux Authentication Server

DS2 Support for Some Data Quality Functions

In SAS Data Quality 3.5, the following functions are supported. See the documentation for "What Is DS2?" in *SAS DS2 Language Reference*.

DQCASE

DQEXTINFOGET

DQEXTRACT

DQEXTTOKENGET

DQEXTTOKENPUT

DQGENDER

DQGENDERINFOGET

DQGENDERPARSED

DQIDENTIFY

DQIDENTIFYIDGET

DQIDENTIFYINFOGET

DQIDENTIFYMULTI

DQLOCALEGUESS

DQLOCALEINFOGET

DQLOCALEINFOLIST

DQLOCALESCORE

DQMATCH

DQMATCHINFOGET

DQMATCHPARSED

DQPARSE

DQPARSEINFOGET

DQPARSETOKENGET

DQPARSETOKENPUT

DQPATTERN

DQSTANDARDIZE

DQTOKEN

DQVER

DQVERQKB

Profiling Capabilities through PROC DATAMETRICS

In SAS Data Quality 3.5, PROC DATAMETRICS implements data profile capabilities that help you recognize patterns, identify sparsity in the data, generate frequency distributions, and calculate basic statistics. Data profiling can also identify redundant data across tables and identify cross-column dependencies. All of this information is essential to achieving the goal of understanding your data. Data profiling can be applied to any data set that is accessible to SAS.

Profile and Categorize Data with New Functions for Identification Analysis

In SAS Data Quality 3.4, three new functions enable data discovery. The DQIDENTIFYINFOGET function returns the identities (content type categories) that are supported by a given identification analysis definition. The DQIDENTIFY function (not new) returns the name of the highest-scoring identity for an input character value. The DQIDENTIFYMULTI function returns a delimited string of identification analysis scores for an input character value. The DQIDENTIFYIDGET function returns an identification analysis score from a delimited list of identification analysis scores.

Combined Documentation for SAS Data Quality 3.4 and SAS 9.4 Data Quality Server

Starting in SAS 9.4M5 and SAS Viya 3.4, this document addresses all aspects of SAS 9.4M5 Data Quality Server and SAS Data Quality 3.4.

SAS Data Quality 3.4 provides full language element support in SAS, and adds data quality support for SAS Cloud Analytic Services (CAS). CAS runs in SAS Viya 3.4.

Perform Data Quality Operations on Data in CAS

Most of the data quality functions that can be used in DATA step programs run in CAS. The reference information for the data quality functions in this document clearly identifies the functions that are valid in CAS.

Execute Programs in CAS That Contain Data Quality Functions

With an appropriate caslib, data quality programs running on a SAS Workspace Server can read and write tables in SAS Cloud Analytic Services.

Use a New Function to Obtain a Confidence Score for Locale Guessing

The new function DQLOCALESCORE returns an integer confidence score for a source string and a locale.

Interoperate with SSL-enabled DataFlux Data Management Server

In SAS 9.4M4 and later, SAS Data Quality Server interoperates with SSL-enabled DataFlux Data Management Server 2.1 and later. Relevant language elements in SAS Data Quality Server can now use HTTPS URLs to communicate with the secured server software to run jobs and services.

List the Locales in the SAS QKB with the DQLOCLST Procedure

In SAS 9.4M4 and later, SAS Data Quality Server includes the DQLOCLST procedure. The DQLOCLST procedure creates a data set that includes the list of locales in the SAS Quality Knowledge Base that is named in the system option DQSETUPLOC=.

Synchronize Results with DataFlux Data Management Software

In SAS 9.4M3 and later, the results of data quality operations are synchronized with results obtained from DataFlux Data Management software. The synchronization occurs in SAS Data Management Studio 2.7 or later and in DataFlux Data Management Server 2.7 or later.

1

Overview of SAS Data Quality

Description

SAS Data Quality provides SAS language elements that perform data quality operations such as matching and standardization. The data quality operations analyze and transform data to improve accuracy and consistency. The result is higher quality data, which yields higher-value analytical results. To learn more about the data quality operations, see SAS Data Quality: Getting Started.

In addition to enabling data quality operations, SAS Data Quality can also build data quality into business processes across your enterprise:

- **Create and apply schemes** to continuously improve source tables. Scheme files are created and updated for individual source tables to maintain standardization across multiple columns.
- **Run and manage jobs and real-time services** on DataFlux Data Management Server. Jobs and real-time services are created in DataFlux Data Management Studio, and then uploaded for HTTP or HTTPS availability through the server. The jobs and real-time services can run data quality operations upon request from web applications, to cleanse data at the point of input.
- **Customize the SAS Quality Knowledge Base** to enhance data cleansing for site-specifc text formats.

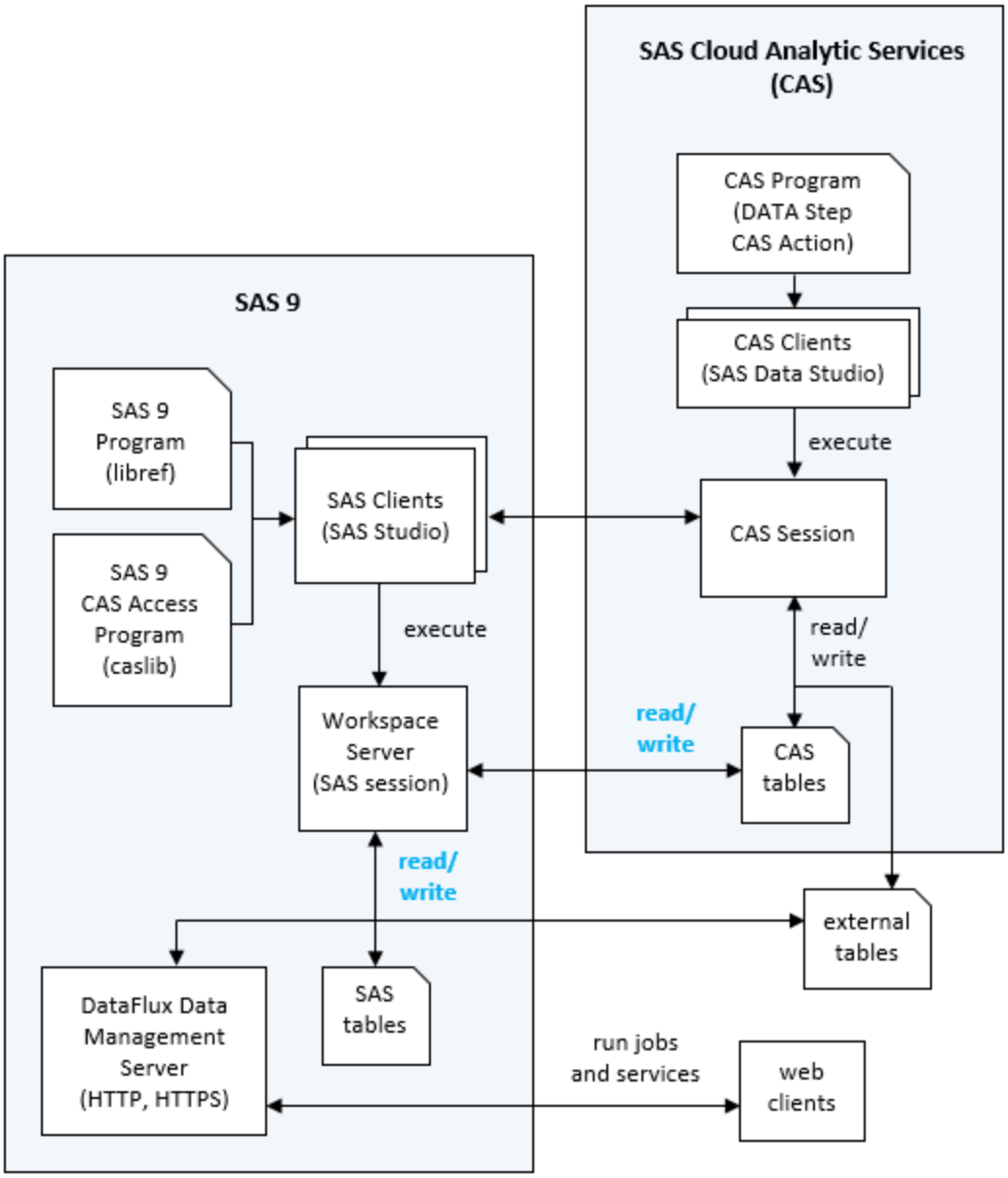
About SAS Data Quality in CAS

In the SAS 9 execution environment, data quality programs run on SAS Workspace Server. The programs use librefs to read and write SAS tables and external tables. If your site uses the SAS Viya execution environment, your data quality programs can read and write tables in SAS Cloud Analytic Services (CAS). To access CAS tables, SAS data quality programs specify caslibs instead of librefs.

The SAS DATA step can also run in CAS, as part of the SAS DATA step CAS action. SAS Data Quality 3.3 or later provides data quality functions that run in the SAS DATA step CAS action.

This document supports data quality programming in CAS and in SAS 9. The terms *CAS* and *SAS 9* are used in this document to indicate where language elements are valid, and to differentiate between the environments in prerequisites and examples.

The following diagram illustrates how data quality programs run and access data in SAS 9 and CAS.



2

Programming with SAS Data Quality

Prerequisites for SAS Data Quality

Introduction

These prerequisites establish configuration requirements for running data quality programs in CAS and SAS 9.

Prerequisites for Data Quality in CAS

Import a SAS Quality Knowledge Base into CAS

Before data quality programs can reference a SAS Quality Knowledge Base in CAS, one or more QKBs must first be imported into CAS. To import a QKB into CAS, administrators use SAS Environment Manager. Initial QKB import normally takes place during the deployment of CAS.

Specify a QKB and Default Locale in CAS

Data quality programs that run in CAS require values for the session options DQSETUPLOC= and DQLOCALE=.

Administrators can establish global CAS defaults for these session options, as described in Set the Default QKB. You can override the global CAS defaults by specifying session option values in your data quality programs.

Prerequisites for Data Quality in SAS 9

Specify a QKB and Locale in SAS 9

Data quality programs that run in SAS 9 require values for the system options DQSETUPLOC= and DQLOCALE=. In addition, the locale or locales that are specified in DQLOCALE= must be loaded into memory. These tasks are accomplished by asserting the AUTOCALL macro %DQLOAD.

Data Quality Configuration Requirements for Secured DataFlux Data Management Server

In SAS 9, when the DataFlux Data Management Server is running in secure mode (with SSL enabled), the server process communicates only with clients that are also in secure mode. Clients that request the execution of jobs and services are required to install and configure OpenSSL 1.0 or higher. In the Windows operating environment, OpenSSL must be installed in the Windows system directory, or in a directory that is in the system PATH.

In the UNIX and Linux operating environments, OpenSSL must be installed in one of the following host directories:

- AIX: openssl.base
- HP-UX: openssl
- Linux: openssl
- Solaris 10: SUNWopenssl-libraries
- Solaris 11: library/security/openssl

Authorization Requirements for DataFlux Data Management Server

Users who create, run, or manage data quality jobs and services must be fully authorized for access by the DataFlux Data Management Server. To learn more about those authorizations, see the DataFlux Data Management Server: Administrator's Guide.

Programming Notes

About Parse Definitions

Parse definitions are referenced to identify the component parts of source strings. The component parts are known as tokens. Token names enable the analysis or cleansing of specified tokens.

Parse definitions and tokens are referenced by the following functions:

- “CALL DQPARSE Routine” on page 130
- “DQPARSEINFOGET Function” on page 131
- “DQTOKEN Function” on page 146
- “DQPARSETOKENGET Function” on page 134
- “DQPARSETOKENPUT Function” on page 135

Parsing a source string assigns tokens only when the content meets the criteria in the parse definition. Therefore, parsed source strings can contain empty tokens. For example, three tokens are empty when you use the DQPARSE function, the Name parse definition, and the ENUSA locale to parse the source string `Ian M. Banks`. 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 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.

About Extraction Definitions

Extraction definitions are referenced to extract parts of source strings and assign them to corresponding named tokens. Extraction input values are delimited so that the elements in those values can be associated with named tokens. After extraction, specific contents of the input values can be returned by specifying the names of tokens.

Extraction definitions and tokens are referenced by the following functions:

- “DQEXTRACT Function” on page 109
- “DQEXTINFOGET Function” on page 108
- “DQEXTTOKENGET Function” on page 110
- “DQEXTTOKENPUT Function” on page 111

Applying an extraction definition to a source string assigns values to named tokens only when the content meets the criteria in the definition. Tokens that do not meet the criteria are empty. For example, applying the Description extraction definition in the ENUSA locale to the source string `"Mr John Smith, 100 SAS Campus Dr, PO Box 12345, Cary, NC 27513, 919-531-0000"` yields the following name/value pairs:

Name
    Mr John Smith

Organization

Address
    100 SAS Campus Dr, PO Box 12345, Cary, NC 27513

E-mail

Phone
    919-531-0000

Additional Info
    ,; ,

About Case and Standardization Definitions

Case and standardization definitions increase consistency for the purposes of display or in preparation for applying a scheme.

Case definitions are referenced by the “DQCASE Function” on page 107. Standardization definitions are referenced by the “DQSTANDARDIZE Function” on page 145.

Case definitions transform the capitalization of source strings. For example, the case definition Proper in the ENUSA locale takes as input any general text. The

Proper definition capitalizes the first letter of each word, and uses lowercase for the other letters in the word. It also recognizes and retains or transforms 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 might be made for specific content. For example, the standardization definition State/Province (Full Name) in the locale ENUSA converts abbreviated state names to full names in uppercase.

Standardization of Dates in the ENUSA Locale

In the ENUSA locale, dates are standardized to two-digit days (00–31), two-digit months (01–12), and four-digit years. 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 *Sample 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 is 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 are 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= .

About the Gender Analysis, Locale Guess, Locale Score, and Identification Analysis Definitions

Gender analysis, locale guess, locale score, and identification analysis definitions enable you to make determinations about source strings. With these definitions, you can determine the following:

- the gender of an individual based on a name value
- the locale that is the most suitable for a given character value
- the confidence score of a locale for a given character value
- the category of a character value, to determine, for example, if a name value applies to an individual or an organization

Gender analysis definitions determine the gender of an individual based on that individual's name. The gender is determined to be unknown if the name meets the following conditions:

- the first name is used by both males and females, and if
- no other clues are provided in the name, or if
- conflicting clues are found in the name.

Gender analysis definitions are referenced by the "DQGENDER Function" on page 113.

Locale guess definitions enable the software to determine the locale that is most likely represented by a character value. All of the locales that are specified in the DQLOCALE= system option (for SAS 9) or the DQLOCALE= session option (for CAS) are considered, but only if they contain the specified locale guess definition.

Locale guess definitions are referenced by the function DQLOCALEGUESS on page 120 and the DQLOCALESCORE function.

Identification analysis definitions are used to categorize character values. Categorization is used in data discovery and profiling. One example of categorization is to use the Entity identification analysis definition to determine if a name value applies to an individual or to an organization.

The "DQIDENTIFY Function" on page 116 references an identification definition to return the highest-scoring identity (category) that is supported by that definition. You can also return identification analysis scores using the DQIDENTIFYMULTI and DQIDENTIFYIDGET functions. To return the identities that are supported by a particular identification analysis definition, use DQIDENTIFYINFOGET.

Customize a QKB

You can customize a QKB so that you can analyze and transform specific types of data according to your individual preferences. You can even create new definitions to process unique business data. You can use your customized QKB to cleanse data in CAS or in SAS 9. To learn more about QKB customization, see QKB Management.

To learn more about QKB customization for SAS 9, see in the *DataFlux Data Management Studio User's Guide*.

Initial Delay Using a Non-Default QKB in CAS

If your site uses a non-default QKB, then you can experience slower execution immediately after you start your CAS session. The slower execution is caused by the required loading of the QKB CAS table onto the CAS worker nodes. The delayed start of the first data quality operation is not seen when you use the default QKB. The delay does not occur because the default QKB is automatically loaded on all nodes at session start.

To speed up your first data quality operation with a non-default QKB, you can customize your session start to load the non-default QKB CAS table on all worker nodes. To learn how to customize your session start, see SAS Cloud Analytic Services: Troubleshooting in *SAS Viya Administration: SAS Cloud Analytic Services*.

Memory Errors

If the SAS console displays error messages regarding insufficient memory, increase the amount of virtual memory that is allocated to your SAS session. To do so, increase the value of the MEMSIZE= system option.

3

DATAMETRICS Procedure

Overview: DATAMETRICS Procedure

What Does the DATAMETRICS Procedure Do?

PROC DATAMETRICS implements data profile capabilities that help you recognize patterns, identify sparsity in the data, generate frequency distributions, and calculate basic statistics. Data profiling analysis can be applied to any data set that is accessible to SAS. The following analysis capabilities are included:

- standard profile metrics
- frequency distribution per column, limiting results to top n, bottom n
- row and columns counts
- pattern analysis
- identification analysis

Syntax: DATAMETRICS Procedure

PROC DATAMETRICS DATA=*input-data-set-name*
OUT=*output-data-set*
<FREQUENCIES =*frequencies*> <MINMAX=*minmax*> <THREADS=*threads*>
<MEDIAN>
<FORMAT>;
IDENTITIES <QKB=qkb> <LOCALE=locale> <DEFINITION=definition>
<MULTIIDENTITY>;
<VARIABLES=list of variables of profile>;

PROC DATAMETRICS Statement

Apply data profiling analysis to a data set.

Syntax

PROC DATAMETRICS DATA=*input-data-set-name*
OUT=*output-data-set*
<FREQUENCIES=*frequencies*> <MINMAX=*minmax*> <THREADS=*threads*>
<MEDIAN>
<FORMAT>;
IDENTITIES <QKB=qkb> <LOCALE=locale> <DEFINITION=definition>
<MULTIIDENTITY>;
<VARIABLES=list of variables of profile>;

Optional Arguments

FREQUENCIES=frequencies
the maximum number of frequency values to publish in the output. The range is 0-2,147,483,647.

Default 1000.

MINMAX=minmax
The maximum number of min/max values to publish in the output. The range is 0-2,147,483,647.

Default 10.

THREADS=threads
The number of threads to be used in computing the results. The range is 0–32,768.

Default 4.

Note If a value of 0 is entered for the THREADS argument, the default number of threads is used.

MEDIAN
Calculates the median if present. This is optional because the median calculation requires more resources.

Default not specified.

FORMAT
Formatted values are used for profiling if present.

Default not specified.

IDENTITIES Statement

Performs identification anaylsis on character data if present.

Default: not specified.

Notes: If IDENTITIES is specified, then the DEFINITION argument is also required. Additionally, the following arguments can be specified in case the default values for the respective arguments are not sufficient: QKB, LOCALE, MULTIIDENTITY.

IDENTITIES can be shortened to ID.

Syntax

IDENTITIES
<QKB=*qkb*> <LOCALE=*locale*> <DEFINITION=*definition*> <MULTIIDENTITY>;

Optional Arguments

QKB=qkb
The fully qualified path of the root directory of the Quality Knowledge Base to use for identification analysis for more information. See "Quality Knowledge Base" in *SAS Data Quality: Getting Started*.

Default DQSETUPLOC option value.

LOCALE=locale
The ISO code name of the QKB locale to use for the specified identification analysis definition to use for identification analysis. See QKB Locale ISO Codes on the SAS support website.

Default DQLOCALE option value.

DEFINITION=definition
The name of the QKB identification analysis definition to use in the specified QKB locale for identification analysis. See "Introducing the Data Quality Operations" in *SAS Data Quality: Getting Started*.

Default not specified.

Notes If the IDENTITIES argument is specified, then this argument is required.

DEFINITION can be shortened to DEF.

MULTIIDENTITY

Calculates more than one identity for identification analysis. The QKB can sometimes return more than one identity for a given value, and each one can have a different score. The returned value is a delimited list of scores for each identity that is supported by the identification-analysis-definition.

Default not specified.

Note MULTIIDENTITY can be shortened to MULTI.

VARIABLES Statement

A space-separated list of variables to profile.

Default: not specified. All variables are profiled.

Results: DATAMETRICS Procedure

DATAMETRICS Procedure

The following table contains output data descriptions. Each row in the table represents a metric value. This table always contains the following columns:

- ColumnId – Indicates the ID of the column to which the metric applies. The data in this column is used to identify the columns of the input (source) data set that was profiled. It is a numeric value starting at 0. All rows that have the same value in this column refer to the same input column in the source data set. The number always starts at 0 for the source columns. For example, all rows that have a 0 in ColumnId contain profiling metrics for the first column in the source data set. This column could also contain the value -1. In this case, the row refers to the table as a whole.
- RowId – The data in this column identifies which profile metric the row contains. See Metric Values for a description of each metric.
- Count – This is a multipurpose column which, depending on the metric in the row, might contain the metric's value.
- CharValue – This is a multipurpose column which, depending on the metric in the row might contain the metric's value.
- DoubleValue – The data in this column is the numeric value for the respective metric.

To retrieve a metric value from the table for a particular column, first find the row that contains the desired ColumnId from the source table, and the desired metric. See Table 3.1 Metric Values. Then look in the subsequent fields in that row (depending on the metric), to get the value.

Table 3.1 *Metric Values*

Row ID Value	Description	Usage Notes
0	number of columns	The number of columns profiled. This is a table-level metric and the column ID is -1. The Count column contains the number of columns profiled.
1	number of rows	The number of rows read from the data source. This is a table-level metric and the column ID is -1. The Count column contains the number of rows read from the data source.
3	columns in data source	The number of columns in the data source. This is a table-level metric and the column ID is -1. The Count column contains the number of columns in the table.
4	date created	The data source creation date time. This is a table-level metric and the column ID is -1. The Count column contains the table creation date and time.
5	date modified	The data source modification date time. This is a table-level metric and the column ID is -1. The Count column contains the table modification date and time.
6	encoding	The data source encoding. This is a table-level metric and the column ID is -1. The Value column contains the table encoding.

Row ID Value	Description	Usage Notes
7	compressed	The value is nonzero if the data source is compressed. This is a table-level metric and the column ID is -1. The Count column contains a nonzero value if the table is compressed.
10	view	The value is nonzero if the data source is a view. This is a table-level metric and the column ID is -1. The Count column contains a nonzero value if the table is a view.
11	lib	The lib where the data source lives. This is a table-level metric and the column ID is -1. The Value column contains the LIBNAME where the table is located.
12	source	The loaded source of the data source. This is a table-level metric and the column ID is -1. The Value column contains the name of the table.
999	Format name	This metric is present when formats=true and when a column has a format. The CharValue contains the format name.
1000	Column name	The name of the column in the database table. The CharValue contains the value.

Row ID Value	Description	Usage Notes
1001	Data type	The Count column contains the number that represents the data type. This integer value represents the data type of the collected column. Here is a list of values and the data type each represents: 1 : CHARACTER (CHAR) 2 : VARCHAR 3 : DATE 4 : DATETIME 7 : DOUBLE 8 : INT32 9 : INT64
1002	Count of values	The Count column contains the number of values in the column.
1003	Name of the column containing the profiled column values	The CharValue column indicates which column contains data for certain metrics like frequency distributions, maximum, minimum, and mode. The value of the metric (for example, CharValue or DoubleValue) appears in this column. You must have the value from this metric before attempting to retrieve values for those other metrics.
1004	Frequency distribution (high)	The Count column contains the number of times that the value occurs and the Value column contains the actual value. This metric indicates the most frequently occurring values and displays the number of times the value occurs. **Note:** Missing values are counted as null values.

Row ID Value	Description	Usage Notes
1005	Frequency distribution (low)	The Count column contains the number of times that the value occurs and the Value column contains the actual value. This metric indicates the least frequently occurring values and displays the number of times that value occurs. **Note:** Missing values are counted as null values.
1006	Maximum value	The Value column contains the actual value.
1007	Maximum length	The Count column contains the value of the maximum length. This applies only to string types.
1008	Mean	The Value column contains the mean. This applies only to numeric types.
1009	Median	The DoubleValue column contains the median and is displayed only if MEDIAN is present in the DATAMETRICS statement. This applies only to numeric types.
1010	Minimum value	The Value column contains the actual value.
1011	Minimum length	The Count column contains the actual value. The value of the minimum length. This applies only to string types.
1012	Mode	The Count column contains the number of times that the value occurs, or -1 if multiple values have an equal number of occurrences.

Row ID Value	Description	Usage Notes
1013	Number of blank values	The count of the blank values. This applies only to string types. The Count column contains the value.
1014	Null count	The Count column contains the number of null values.
1016	Outlier high values	The maximum values. The Value column contains the value. For numeric data types, these are the values with the highest numeric value. For strings, these are values that sort toward the end of the alphabet (for example, Z).
1017	Outlier low values	The minimum values. The Value column contains the value. For numeric data types, these are the values with the lowest numeric value. For strings, these are values that sort toward the beginning of the alphabet (for example, A).
1018	Pattern frequency distribution (high)	Frequency distribution based on a generated pattern of the data. The Count column contains the number of times that the value occurs and the Value column contains the actual value.
1019	Pattern frequency distribution (low)	Frequency distribution based on a generated pattern of the data. The Count column contains the number of times that the value occurs and the Value column contains the actual value.
1020	Unique patterns	The count of unique patterns. The Count column contains the value.

Row ID Value	Description	Usage Notes
1021	Primary Key Candidate	If the Count column value is nonzero, then the column is a primary key candidate.
1022	Standard Deviation	The DoubleValue column contains the standard deviation value.
1023	Standard Error	The DoubleValue column contains the standard error value.
1025	Unique count	The count of unique values. The Count column contains the value.
1028	ID Analysis	The CharValue column contains the computed identity for the column. The Count contains the number of rows for which the identity value was computed. For MULTIIDENTITY, the score is returned in the DoubleValue column, and in this case the count is not reported.
1029	Actual bool	The number of string values identified as Boolean ("yes", "no", "true", "false", "t", "f", "y" and "n"). The Count column contains the number of values matching this type.
1030	Actual date	The number of string values identified as date. The Count column contains the number of values matching this type.
1031	Actual integer	The number of string values identified as integer. The Count column contains the number of values matching this type.

Row ID Value	Description	Usage Notes
1032	Actual real	The number of string values identified as real. The Count column contains the number of values matching this type.
1033	Actual string	The number of string values identified as string. The Count column contains the number of values matching this type.
1034	Actual designated	A number representing the designated type. If more than 80% of the values are a single type, it becomes the designated type. The Count column contains a number that represents the type.
1035	Determined type %	The percentage of values for designated type. The Count column contains the percentage of values for a designated type.
1037	Data length	The Count column contains the length value. The CharValue column displays a description of what the Count column value represents; for example: digits or bytes. This is retrieved from data source metadata.
1038	Ordinal	The position of the column to be profiled. For example, column 1, column 2, and column 5 are profiled first, so the row IDs are 0, 1, and 2; but the ordinals are 1, 2, and 5. This is retrieved from data source metadata. The Count column contains the ordinal position of the column.

Row ID Value	Description	Usage Notes
1039	Number of decimal places specified in format, if a format was associated with the column	This is retrieved from data source metadata. The Count column contains the decimal places.
1040	null percent	The percentage of values that are null. The value comes from the DoubleValue column.
1041	unique percent	The percentage of unique values. The value comes from the DoubleValue column.
1042	pattern unique percent	The percentage of unique pattern values. The value comes from the DoubleValue column.
1043	blank percent	The percentage of blank values. The value comes from the DoubleValue column.
1100	Error message	This metric is present if an error was encountered while calculating the results for this field. The CharValue contains the error message.
1101	Metadata allows nulls	This metric indicates that the metadata for this column allows NULLs. The Count contains 0 when NULLS are not allowed and 1 when NULLs are allowed.

Note: Depending on the data source, nulls and blanks are sometimes counted differently and might not be counted as a unique value.

If columns have no values (nulls, blanks, or empty strings) or no rows, then some metrics are by default not shown.

Examples: DATAMETRICS Procedure

Example 1: Basic Usage with Minimum Required Parameters

This statement runs PROC DATAMETRICS on the table my_data. Since no variable list is provided, the PROC generates metrics for all variables in the table. Metrics are written to a table named my_results. Because the IDENTITIES statement is not included, no identification analysis values are included in the output. Because no options are specified, default values are used for all options. The following example shows the minimum number of arguments in use.

```
proc datametrics data=my_data out=my_results;
```

Example 2: Advanced Usage with All Optional Parameters

This statement runs PROC DATAMETRICS on the table my_data. The procedure generates metrics for the variables 'name' and 'address'. If the input table is formatted, PROC DATAMETRICS uses the formatted values in the variables when generating metrics. Metrics are written to a table named my_results. Metrics include multiple identification analysis results generated using the 'English, United States' version of the 'Field Content' definition in the Quality Knowledge Base that is located in directory '/sas/dqc/QKBLoc'. The procedure generates the median value and ten minimum and maximum values for each variable. It generates the one hundred most frequently occurring values in each variable. The procedure uses eight threads for processing. The following example shows all arguments in use.

```
proc datametrics data=my_data out=my_results frequencies=100
   minmax=12 threads=8 median format;
   identities qkb='/sas/dqc/QKBLoc' locale='ENUSA'
   def='Field Content' multiidentity;
   variables name address;
   run;
```

Example 3: Reading Metrics from Profile Results Tables

Use the following steps to find the average value in the age field:

1. Using Table 3.1 Metric Values, find the row ID value for the Column Names (ROWID 1000).
2. In the Datametrics output, find the rows containing ROWID 1000. In the example below, these are age (Obs 1) and name (Obs 39). This example is looking for the average age, so find the Column ID for the age field (COLID 0).

3 Using Table 3.1 Metric Values again, find the row ID value for the mean metric ID (ROWID 1008).

4 To get the average value in the age field, look for the row where COLID is 0 (age) and ROWID is 1008 (mean metric) in the Datametrics output.

5 The mean value is in the DOUBLEVALUE column for that row (39.400).

```
data orig;
   infile datalines dsd delimiter=',';
   input age
      name : $255.
      ;
      datalines;
37,"James Johnson"
40,"Helen Franks"
51,"Robert McDonald"
27,"Patrick Doyles"
42,"Elizabeth Stallman"
;


run;
proc datametrics data=orig out=stage;
run;

proc print data = stage; TITLE "Datametrics output"; run;
run;
```

```
                                                          Datametrics
output                        11:10 Monday, November 4, 2019   1

                                Obs    COLID    ROWID    COUNT
CHARVALUE             DOUBLEVALUE

                                  1       0       1000        0
age                         .
                                  2       0       1038
1                                .
                                  3       0       1101
1                                .
                                  4       0       1039
0                                .
                                  5       0       1037       15
digits                      .
                                  6       0       1001        7
DOUBLE                      .
                                  7       0       1003        0
DOUBLEVALUE                 .
                                  8       0       1002
5                                .
                                  9       0       1025
5                                .
                                 10       0       1014
0                                .
                                 11       0       1010
0                            27.000
```

```
                                    12      0       1006
0                                 51.000
                                    13      0       1008
0                                 39.400
                                    14      0       1022
0                                  8.678
                                    15      0       1023
0                                  3.881
                                    16      0       1021
1                                   .
                                    17      0       1041
0                                100.000
                                    18      0       1040
0                                  0.000
                                    19      0       1004
1                                 51.000
                                    20      0       1005
1                                 51.000
                                    21      0       1004
1                                 42.000
                                    22      0       1005
1                                 42.000
                                    23      0       1004
1                                 40.000
                                    24      0       1005
1                                 40.000
                                    25      0       1004
1                                 37.000
                                    26      0       1005
1                                 37.000
                                    27      0       1004
1                                 27.000
                                    28      0       1005
1                                 27.000
                                    29      0       1016
1                                 51.000
                                    30      0       1016
1                                 42.000
                                    31      0       1016
1                                 40.000
                                    32      0       1016
1                                 37.000
                                    33      0       1016
1                                 27.000
                                    34      0       1017
1                                 27.000
                                    35      0       1017
1                                 37.000
                                    36      0       1017
1                                 40.000
                                    37      0       1017
1                                 42.000
                                    38      0       1017
1                                 51.000
                                    39      1       1000        0
name                                .
```

```
                                 40       1        1038
2                                .
                                 41       1        1101
1                                .
                                 42       1        1039
0                                .
                                 43       1        1037     255
characters                  .
                                 44       1        1001       1
CHARACTER                   .
                                 45       1        1003       0
CHARVALUE                   .
                                 46       1        1002
5                                .
                                 47       1        1025
5                                .
                                 48       1        1014
0                                .
                                 49       1        1013
0                                .
                                 50       1        1020
5                                .
                                 51       1        1042
0                             100.000
                                 52       1        1010       0
Elizabeth Stallman          .
                                 53       1        1006       0
Robert McDonald             .
                                 54       1        1011
12                                .
                                 55       1        1007
18                                .
                                 56       1        1021
1                                .
                                                                Datametrics
output                        11:10 Monday, November 4, 2019   2

                              Obs    COLID    ROWID        COUNT
CHARVALUE                 DOUBLEVALUE

                               57       1      1034
1033                                     .
                               58       1      1029
0                                      .
                               59       1      1033
5                                      .
                               60       1      1030
0                                      .
                               61       1      1031
0                                      .
                               62       1      1032
0                                      .
                               63       1      1035
100                                     .
                               64       1      1041
0                                  100
```

```
                              65       1      1040
0                                  0
                              66       1      1043
0                                  0
                              67       1      1004            1
Robert McDonald                .
                              68       1      1005            1
Robert McDonald                .
                              69       1      1004            1
Patrick Doyles                 .
                              70       1      1005            1
Patrick Doyles                 .
                              71       1      1004            1
James Johnson                  .
                              72       1      1005            1
James Johnson                  .
                              73       1      1004            1
Helen Franks                   .
                              74       1      1005            1
Helen Franks                   .
                              75       1      1004            1
Elizabeth Stallman             .
                              76       1      1005            1
Elizabeth Stallman             .
                              77       1      1018            1
Aaaaaaaaa Aaaaaaaa             .
                              78       1      1019            1
Aaaaaaaaa Aaaaaaaa             .
                              79       1      1018            1
Aaaaaaa Aaaaaa                 .
                              80       1      1019            1
Aaaaaaa Aaaaaa                 .
                              81       1      1018            1
Aaaaaa AaAaaaaa                .
                              82       1      1019            1
Aaaaaa AaAaaaaa                .
                              83       1      1018            1
Aaaaa Aaaaaaa                  .
                              84       1      1019            1
Aaaaa Aaaaaaa                  .
                              85       1      1018            1
Aaaaa Aaaaaa                   .
                              86       1      1019            1
Aaaaa Aaaaaa                   .
                              87       1      1016            1
Robert McDonald                .
                              88       1      1016            1
Patrick Doyles                 .
                              89       1      1016            1
James Johnson                  .
                              90       1      1016            1
Helen Franks                   .
                              91       1      1016            1
Elizabeth Stallman             .
                              92       1      1017            1
Elizabeth Stallman             .
```

```
                                93       1      1017            1
Helen Franks                     .
                                94       1      1017            1
James Johnson                    .
                                95       1      1017            1
Patrick Doyles                   .
                                96       1      1017            1
Robert McDonald                  .
                                97      -1         0
2                                      .
                                98      -1         1
5                                      .
                                99      -1         3
2                                      .
                               100      -1         4
1.8885E15                                      .
                               101      -1         5
1.8885E15                                      .
                               102      -1         6            0
latin1  Western (ISO)            .
                               103      -1         7
0                                      .
                               104      -1        10
0                                      .
                               105      -1        11            0
WORK                             .
                               106      -1        12            0
ORIG                             .
```

4

DMSRVADM Procedure

Overview: DMSRVADM Procedure

What Does the DMSRVADM Procedure Do?

The DMSRVADM procedure creates a job status data set for a specified DataFlux Data Management Server. The job status data set lists all jobs that have a log file on the server.

Syntax: DMSRVADM Procedure

PROC DMSRVADM

< HOST=*host-name*>
<OUT=*output-data-set*>
<PASSWORD=*password*>
<PORT= *job-port-number*>
<USERID= *userid*>;

PROC DMSRVADM Statement

The DMSRVADM procedure creates a status information data set.

Restriction: Not valid in CAS

Syntax

PROC DMSRVADM

<HOST='*host-name*'>
<OUT=*output-data-set*>
<PASSWORD='*password*'>
<PORT='*job-port-number*'>
<USERID='*userid*'>;

Optional Arguments

HOST=*host-name*

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The string or variable value is the URL of the server.

When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, https://myhost.unx.com).

Default *localhost*

OUT=*output-data-set*

specifies the storage location of the job status data set.

Default If the OUT option is not specified, the input data set _LAST_ is used.

PASSWORD=*password*

authenticates the user according to the registry in the DataFlux Data Management Server. The password can be plain text or encoded in SAS.

Note If security has not been configured on the server, the PASSWORD option is ignored.

PORT=*port-number*

identifies the port through which the *host* communicates with the DataFlux Data Management Server.

Default If the *port-number* is not specified, or if the value is 0, or a negative number, the default port number 21036 is used.

USERID=*userid*

authenticates the user according to the registry in the DataFlux Data Management Server.

Usage: DMSRVADM Procedure

The Job Status Data Set

The job status data set contains the following variables:

JOBID
specifies the job-id that was submitted to the DataFlux Data Management Server. The job-id is previously set by a function such as DMSRVBATCHJOB. The job-id is an argument in these functions:

- DMSRVCOPYLOG
- DMSRVDELETELOG
- DMSRVJOBSTATUS
- DMSRVKILLJOB

See "Functions Listed Alphabetically" on page 89 for a complete list of functions and call routines.

STATUS
specifies the following job status codes:

0
Job completed successfully.

1
Job failed.

2
Job still running.

3
Job in queue awaiting execution.

Descriptions and associated job types are listed in the table below.

JOBDESC	JOBTYPE
Data Service	0
Data Job	1
Profile Job	2
Process Job/Service	3
Repository Job	4
Unknown Type	Undetermined

Security

If security is implemented on a DataFlux Data Management Server, then you must register credentials with the function DMSRVUSER before you run PROC DMSRVADM. To learn more about registering credentials, see "DMSRVUSER Function" on page 105.

Examples: DMSRVADM Procedure

Example 1: Generate a Job Status Data Set

This example generates a data set that provides information about jobs that are running or have run on a DataFlux Data Management Server. The job status data set contains information about jobs that are represented by log files on the server.

```
proc dmsrvadm
   out=work.jobReport
   host='http://myhost.unx.com'   port=50001;
run;
```

Example 2: Clean Up Jobs and Logs

This example generates a job report and then uses the contents of the report to terminate all jobs and delete all log files on the DataFlux Data Management Server:

```
proc dmsrvadm
   out=work.jobReport
   host='http://myhost.unx.com'   port=50001;
run;

data _null_;
   set work.joblist;
   kjrc=dmsrvkilljob (jobid, 'http://myhost.unx.com' , 50001);
   dlrc=dmsrvdeletelog (jobid, 'http://myhost.unx.com' , 50001);
run;
```

5

DMSRVDATASVC Procedure

Overview: DMSRVDATASVC Procedure

What Does the DMSRVDATASVC Procedure Do?

The DMSRVDATASVC procedure runs a DataFlux Data Management Studio real-time service on a DataFlux Data Management Server. DataFlux Data Management real-time services cleanse smaller amounts of data at the point of data entry. Data processing is intended to be synchronous; a client application requests the service and awaits a response. The DMSRVDATASVC procedure authenticates you on the server, requests a service, delivers input data to the server, and delivers output data to a SAS DATA set.

To improve performance, large input data sets are delivered to the DataFlux Data Management Server in chunks of a specified size.

To cleanse or analyze larger amounts of data asynchronously, execute a DataFlux job using the function DMSRVBATCHJOB.

Syntax: DMSRVDATASVC Procedure

Restriction: Not valid in CAS

See: The documentation for your operating environment for specific DMSRVDATASVC procedure information.

PROC DMSRVDATASVC

<DATA=*input-data-set*>
<BLOCKSIZE=*rows-per-message*>
<HOST='*host-name*>'
<PARMLIST='*parameter-list*'>
<MISSINGVARSOK>
<NOPRINT>
<OUT=*output-data-set*>
<PASSWORD='*password-on-server*'>
<PORT=*port-number*>
<SERVICE=*service-name*>
<SERVICEINFO>
<TIMEOUT=*message-processing-limit*>
<TRIM>
<USERID='*user-name-on-server*'>;

PROC DMSRVDATASVC Statement

The DMSRVDATASVC procedure runs a real-time service on DataFlux Data Management Server.

Syntax

PROC DMSRVDATASVC

<DATA=*input-data-set*>
<BLOCKSIZE=*rows-per-message*>
<HOST='*host-name*'>
<PARMLIST=*parameter-list*>
<MISSINGVARSOK>
<NOPRINT>
<OUT=*output-data-set*>
<PASSWORD='*password-on-server*'>
<PORT=*port-number*>
<SERVICE='*service-name*'>
<SERVICEINFO>
<TIMEOUT=*message-processing-limit*>
<TRIM>
<USERID='*user-name-on-server*'>;

Actions

BLOCKSIZE= *rows-per-message*

specifies the maximum number of rows of source data that are transmitted in the messages that are sent to the DataFlux Data Management Server. If this option is not specified, then the entire data set is transmitted in a single message. Transmitting large data sets in a single message can restrict resources on the DataFlux Data Management Server. The server processes each message separately. Output is delivered in a single message.

The real-time service needs to be written to accommodate multiple messages. Real-time services are created in DataFlux Data Management Studio.

Restriction Real-time ervices that require the entire data set, such as those that calculate averages or frequencies, cannot use the BLOCKSIZE option.

DATA= *data-set-name*
identifies name of the input data set.

Default If the DATA option is not specified, the input data set _LAST_ is used.

HOST=*host-name*
identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server.

When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, https://myhost.unx.com).

Default *localhost*.

PARMLIST=*parameter-list*
specifies one or more input parameters for the real-time service, as a series of name and value pairs. If the real-time service uses a parameter list, and if a name in that list matches a name in the supplied*parameter-list*, then the name in the real-time service list is assigned the corresponding value in the *parameter-list*.

In the *parameter-list*, names and values must appear within single quotation marks. An equal sign must separate the name and value. A comma separates one pair from the next.

MISSINGVARSOK
indicates that the real-time service is to be allowed to continue to run when one or more variables (or table columns) are missing from the input data set. When the MISSINGVARSOK option is set, any data that is missing from the input data set is assumed to be non-critical.

Default MISSINGVARSOK is not set by default.

NOPRINT
if the SERVICEINFO option is specified, NOPRINT prevents the SERVICEINFO information from being written into the SAS log.

OUT=*output-data-set*
identifies the name of the output data set. Real-time services always create new data sets or overwrite existing data sets.

Default If the OUT option is not specified, then the name of the output data set is _LAST_.

PASSWORD=*password*
authenticates the user according to the registry in the DataFlux Data Management Server. The password can be plain text or encoded in SAS.

Note If security has not been configured on the server, then the PASSWORD option is ignored.

PORT=*port-number*
identifies the port number through which the named host communicates with the DataFlux Data Management Server. If this option is not specified, or if the value is zero or a negative number, then the default port number 21036 is used.

SERVICE=*service-name*
identifies the real-time service on the DataFlux Data Management Server.

SERVICEINFO
writes the input and output columns used by the real-time service into the data set that is specified by the OUT option.

The data set has four columns:

- Name is the column name.
- Type is the type of data in column -character(C) or numeric(N).
- Length is the length of column data.
- Class is the input, output, or macro.

Default The real-time service information is also written to the SAS log.

Restriction If SERVICEINFO is specified, then the real-time service is not run. Any options related to the execution of the real-time service, such as the BLOCKSIZE option, are ignored.

TIMEOUT=*message-processing-limit*
specifies a time in seconds after which the procedure terminates if the host that is running the DMSRVDATASVC procedure has not received a response from the DataFlux Data Management Server. If data is delivered to the server in multiple messages using the BLOCKSIZE option, then the TIMEOUT value is applied to each message.

Default 10 seconds.

Tip A value of zero or a negative number enables the procedure to run without a time limit.

TRIM
removes any blank spaces from the end of the input values.

Default TRIM is not set by default.

USERID=*user-name*
identifies the user according to the registry in the DataFlux Data Management Server.

Note If security has not been configured on the server, then the USERID option is ignored.

Usage: DMSRVDATASVC Procedure

The Input Data Set

The DMSRVDATASVC procedure acquires the names of the columns that the real-time service expects to receive as input from the DataFlux Data Management Server. In this case, the name of the input data set must match the name that is specified in the real-time service. If the expected column names do not match the column names in the input data set, then the DMSRVDATASVC procedure terminates.

Because real-time services can be created so that they can use any named data set for input, the name of a specific input data set is not required.

The Output Data Set

If the output data set exists, then new output data overwrites any existing data. The type of the output data is determined by the real-time service.

Examples: DMSRVDATASVC Procedure

Example 1: Send Input Data to Service and Output to Data Set

Features:

```
PROC DMSRVDATASVC
    HOST=
    PORT=
    SERVICE=
    DATA=
    OUT=
    TIMEOUT=
    USERID=
    PASSWORD=
```

This example reads a data file into a specified real-time service, and the output from the service appears in the indicated output data set. The TIMEOUT option specifies that the real-time service will terminate after 36 seconds if the service does not complete within that time. The real-time service is executed under the credentials

that are specified in the procedure. The real-time service was previously created and uploaded to the DataFlux Data Management Server.

```
/* send input data dqsio.dfsample to service analysis.ddf.        */
/* output from the data service appears in data set work.outsrv17 */

PROC DMSRVDATASVC
   HOST='http://myhost.unx.com'   PORT=21036
   SERVICE='analysis.ddf'
   DATA=dqsio.dfsample
   OUT=outsrv17
   TIMEOUT=36
   USERID='myname'
   PASSWORD='mypassword';
RUN;
```

Example 2: Run a DataFlux Data Management Studio Service

Features:

PROC DMSRVDATASVC
SERVICE=
DATA=
OUT=

This example runs a DataFlux Data Management Studio service on a DataFlux Data Management Server that is installed on the local host. The PORT option is not set, so the server communicates over the default port 21036. The DATA option specifies the input data set. The OUT option specifies the output data sets. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

```
PROC DMSRVDATASVC
   SERVICE='myService'
   DATA=work.insrv
   OUT=work.outsrv;
RUN;
```

6

DMSRVPROCESSSVC Procedure

Overview: DMSRVPROCESSSVC Procedure

What Does the DMSRVPROCESSSVC Procedure Do?

The DMSRVPROCESSSVC procedure runs a service on a DataFlux Data Management Server. The procedure executes a DM process service, or, when the SERVICEINFO option is specified, it produces a list of inputs and outputs managed by the service.

A process service generates a list of name-value pairs on output. When the SERVICEINFO option is not specified, the procedure writes these pairs to the output data set. This data set contains two character-type columns, with names "Name" and "Value."

When the SERVICEINFO option is specified, the procedure generates an output data set with two columns. The first column, named "Parameter", contains parameter names. The second column, named "Type", lists the parameter as "INPUT" or "OUTPUT".

Syntax: DMSRVPROCESSSVC Procedure

Restriction: Not valid in CAS.

PROC DMSRVPROCESSSVC

<HOST='*host-name*'>
<PARMLIST=*parameter-list*>
<NOPRINT>
<OUT=*output-data-set*>
<PASSWORD='*password-on-server*'>
<PORT=*port-number*>
<SERVICE='*service-name*'>
<SERVICEINFO>
<TIMEOUT=*message-processing-limit*>
<USERID='*user-name-on-server*'>;

PROC DMSRVPROCESSSVC Statement

The DMSRVPROCESSSVC procedure runs a service on a DataFlux Data Management Server.

Syntax

PROC DMSRVPROCESSSVC

<HOST='*host-name*'>
<PARMLIST=*parameter-list*>
<NOPRINT>
<OUT=*output-data-set*>
<PASSWORD='*password-on-server*'>
<PORT=*port-number*>
<SERVICE='*service-name*'>
<SERVICEINFO>
<TIMEOUT=*message-processing-limit*>
<USERID='*user-name-on-server*'>;

Actions

HOST=*host-name*

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server.

When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, https://myhost.unx.com).

Default *localhost*.

PARMLIST=*parameter-list*

specifies one or more input parameters for the process service, as a series of name and value pairs. If the process service uses a parameter list, and if a name in that list matches a name in the supplied*parameter-list*, then the name in the process service list is assigned the corresponding value in the *parameter-list*.

In the *parameter-list*, names and values must appear within single quotation marks. An equal sign must separate the name and value. A comma separates one pair from the next.

NOPRINT
if the SERVICEINFO option is specified, NOPRINT prevents the SERVICEINFO information from being written into the SAS log.

OUT=*output-data-set*
identifies the name of the output data set. Process services always create new data sets or overwrite existing data sets.

Default If the OUT option is not specified, then the name of the output data set is _LAST_ .

PASSWORD=*password*
authenticates the user according to the registry in the DataFlux Data Management Server. The password can be plain text or encoded in SAS.

Note If security has not been configured on the server, then the PASSWORD option is ignored.

PORT=*port-number*
identifies the port number through which the named host communicates with the DataFlux Data Management Server. If this option is not specified, or if the value is zero or a negative number, the default port number 21036 is used.

SERVICE=*service-name*
identifies the process service on the DataFlux Data Management Server.

SERVICEINFO
writes the input and output columns used by the process service into the data set that is specified by the OUT option.

The data set has four columns:

- Name is the column name.
- Type is the type of data in column -character(C) or numeric(N).
- Length is the length of column data.
- Class is the input, output, or macro.

Default The process service information is also written to the SAS log.

Restriction If SERVICEINFO is specified, the process service is not run. Any options related to the execution of the process service, such as the BLOCKSIZE option, are ignored.

TIMEOUT=*message-processing-limit*
specifies a time in seconds after which the procedure terminates if the host that is running the DMSRVPROCESSSVC procedure has has not received a response from the DataFlux Data Management Server. If data is delivered to the server in multiple messages using the BLOCKSIZE option, then the TIMEOUT value is applied to each message.

Tip A value of zero or a negative number enables the procedure to run without a time limit.

USERID=*user-name*
identifies the user according to the registry in the DataFlux Data Management Server.

Note If security has not been configured on the server, then the USERID option is ignored.

Usage: DMSRVPROCESSSVC Procedure

The Input Data Set

The DMSRVPROCESSSVC procedure acquires the names of the columns that the process service expects to receive as input from the DataFlux Data Management Server. Because process services can be created so that they can use any named data set for input, the name of a specific input data set is not required.

The Output Data Set

If the output data set exists, then new output data overwrites any existing data. The type of the output data is determined by the service.

Example: Run a DataFlux Data Management Service DMSRVPROCESSSVC Procedure

Example 1: Run a DataFlux Data Management Service

Features:

```
PROC DMSRVPROCESSSVC
   HOST=
   PORT=
   SERVICE=
   TIMEOUT=
   USERID=
   PASSWORD=
```

This example runs a process service on a DataFlux Data Management Server that is installed on the default port. The TIMEOUT option specifies that the process service will terminate after 360 seconds if it does not complete within that time. The process service is executed under the credentials that are specified in the procedure. The process service was previously created and uploaded to the DataFlux Data Management Server.

```
PROC DMSRVPROCESSSVC
```

```
   HOST='http://myhost.unx.com'   PORT=21036
   SERVICE='concatenate.djf'
   TIMEOUT=360
   USERID='myname'
   PASSWORD='mypassword'
RUN;
```

7

DQLOCLST Procedure

Overview: DQLOCLST Procedure

What Does the DQLOCLST Procedure Do?

The SAS procedure DQLOCLST generates a list of locales contained in the Quality Knowledge Base (QKB) that is named by the SAS option DQSETUPLOC. The list of locales is written into a SAS data set that the user selects. This data set contains a single column, LOCALE, whose value is a locale name. There is one row per locale found in the QKB.

Syntax: DQLOCLST Procedure

Restriction: Not valid in CAS.

PROC DQLOCLST
<OUT=*output-data-set*>;
run;

PROC DQLOCLST Statement

The DQLOCLST procedure creates a data set that includes the list of locales in the Quality Knowledge Base that is named by the system option DQSETUPLOC=.

Syntax

PROC DQLOCLST
<OUT=*output-data-set*>;
run;

Action

OUT=*output-data-set*
identifies the name of the output data set. The procedure follows standard SAS data set naming conventions.

Example: Create a Data Set of Locales DQLOCLST Procedure

Example 1: Create a Data Set of Locales

Features: PROC DQLOCLST
OUT=

This example creates a SAS data set with the name `loclist` in the `work` library. The DQSETUPLOC= system option specifies a QKB. For this example, the QKB is in the `/path/to/qkbci` directory. This example prints the result.

Note: For SAS Viya 4.0.1 and later, the path of the QKB is not required. Instead, specify the name of the QKB. QKBs are located in the QKB Repository. To get a list of all QKBs contained in the QKB repository, use the QKB procedure. For more information, see Overview: QKB Procedure on page 81.

```
options DQSETUPLOC="/path/to/qkbci";
proc dqloclst out=work.loclist;
run;
proc print data=work.loclist;
run;
```

8

DQMATCH Procedure

Overview: DQMATCH Procedure

What Does the DQMATCH Procedure Do?

PROC DQMATCH creates matchcodes as a basis for standardization or transformation. The matchcodes reflect the relative similarity of data values. Matchcodes are created based on a specified match definition in a specified locale. The matchcodes are written to an output SAS data set. Values that generate the same matchcodes are candidates for transformation or standardization.

The DQMATCH procedure can generate cluster numbers for input values that generate identical matchcodes. Cluster numbers are not assigned to input values that generate unique matchcodes. Input values that generate a unique matchcode (no cluster number) can be excluded from the output data set. Blank values can be retained in the output data set, and they can receive a cluster number.

A specified sensitivity level determines the amount of information in the matchcodes. The amount of information in the matchcode determines the number of clusters and the number of entries in each cluster. Higher sensitivity levels produce fewer clusters, with fewer entries per cluster. Use higher when you need matches that are more exact. Use lower sensitivity levels to sort data into general categories or to capture all values that use different spellings to convey the same information.

Syntax: DQMATCH Procedure

Restriction: Not valid in CAS

Requirement: At least one CRITERIA statement is required

Interaction: VARCHAR data is supported when data is in CAS

PROC DQMATCH DATA=*input-data-set-name*
CLUSTER=*output-numeric-variable-name*
CLUSTER_BLANKS | NO_CLUSTER_BLANKS
CLUSTERS_ONLY
DELIMITER | NODELIMITER
LOCALE=*locale-name*
MATCHCODE=*output-character-variable-name*
OUT=*output-data-set-name*;
CRITERIA *options*;

PROC DQMATCH Statement

Create match-codes as a basis for standardization or transformation.

Syntax

PROC DQMATCH <DATA=*input-data-set-name*>
<CLUSTER=*output-numeric-variable-name*>
< | >
<CLUSTERS_ONLY>
< | >
<LOCALE=*locale-name*>
<MATCHCODE=*output-character-variable-name*>
<OUT=*output-data-set-name*>;

Optional Arguments

CLUSTER=*variable-name*
specifies the name of the numeric variable in the output data set that contains the cluster number.

Interaction If the CLUSTER option is not specified and if the CLUSTERS_ONLY option is specified, an output variable named CLUSTER is created.

CLUSTER_BLANKS | NO_CLUSTER_BLANKS
specifies how to process blank values.

CLUSTER_BLANKS
specifies that blank values are written to the output data set. The blank values do not have accompanying matchcodes.

NO_CLUSTER_BLANKS
specifies that blank values are not written to the output data set.

Default CLUSTER_BLANKS

CLUSTERS_ONLY
specifies that input character values that are part of a cluster are written to the output data set. Excludes input character values that are not part of a cluster.

Default This option is not asserted by default. Typically, all input values are included in the output data set.

Note A cluster number is assigned only when two or more input values produce the same *match–code*.

DATA=*input-data-set-name*
specifies the name of the input SAS data set.

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

DELIMITER | NODELIMITER
specifies whether exclamation points (!) are used as delimiters.

DELIMITER
when multiple CRITERIA statements are specified, DELIMITER specifies that exclamation points (!) separate the individual *matchcodes* that make up the concatenated *matchcodes*. *Matchcodes* are concatenated in the order of appearance of CRITERIA statements in the DQMATCH procedure.

NODELIMITER
specifies that multiple *matchcodes* are concatenated without exclamation point delimiters.

Defaults (SAS) uses a delimiter.

(DataFlux Data Management Studio) does not use a delimiter.

Note Be sure to use delimiters consistently if you plan to analyze, compare, or combine matchcodes created in SAS and in DataFlux Data Management Studio.

LOCALE=*locale-name*
specifies the name of the locale that is used to create *matchcodes*. The *locale-name* can be a name in quotation marks, or an expression that evaluates to a *locale-name*. It can also be the name of a variable whose value is a *locale-name*.

The specified locale must be loaded into memory as part of the locale list. If you receive an out-of-memory error when you load the locale, you can increase the value in the MAXMEMQUERY system option. For more information, see your host-specific SAS 9.4 documentation, such as *SAS Companion for Windows*.

Default The first locale name in the locale list.

Restriction If no *locale-name* is specified, the first locale in the locale list is used.

Note The match definition, which is part of a 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-character-variable-name*
specifies the name of the output character variable that stores the matchcodes. The DQMATCH procedure defines a sufficient length for this variable, even if a variable with the same name exists in the input data set.

MATCH_CD is created if the following statements are all true:

- The MATCHCODE option is not specified in the DQMATCH procedure.
- The MATCHCODE option is not specified in subsequent CRITERIA statements.
- The CLUSTER option is not specified.
- The CLUSTERS_ONLY option is not specified.

Type The MATCHCODE variable always has the data type CHAR.

OUT=*output-data-set-name*
specifies the name of the output data set for *matchcodes* created with the DQMATCH procedure. The DQMATCH procedure creates *matchcodes* for specified character variables in an input data set.

Note If the specified output data set does not exist, the DQMATCH procedure creates it.

CRITERIA Statement

Creates matchcodes and optional cluster numbers for an input variable.

Syntax

CRITERIA <CONDITION=*integer*>
<*input-variable-name* | *input-variable-name*>
< | *match-definition*'>
<MATCHCODE=*output-character-variable*>
<SENSITIVITY=*sensitivity-level* >;

Optional Arguments

CONDITION=*integer*
groups CRITERIA statements to constrain the assignment of cluster numbers.

- Multiple CRITERIA statements with the same CONDITION value are all required to match the values of an existing cluster to receive the number of that cluster.
- The CRITERIA statements are applied as a logical AND.

- If more than one CONDITION option is defined in a series of CRITERIA statements, then a logical OR is applied across all CONDITION option values.
- In a table of customer information, you can assign cluster numbers based on matches between the customer name AND the home address.
- You can also assign cluster numbers on the customer name and organization address.
- All CRITERIA statements that lack a CONDITION option receive a cluster number based on a logical AND of all such CRITERIA statements.

Default 1

Restriction If you specify a value for the MATCHCODE option in the DQMATCH procedure, and you specify more than one CONDITION value, then SAS generates an error. To prevent the error, specify the MATCHCODE option in the CRITERIA statements only.

Note If you have not assigned a value to the CLUSTER option, then the cluster numbers are assigned to a variable named CLUSTER by default.

See "DQMATCHINFOGET Function" on page 125.

DELIMSTR= |
specifies the name of a variable.

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

VAR=*input-variable-name*
specifies the name of the character variable that is used to create matchcodes. If the variable contains delimited values, use the DELIMSTR option.

The type of the input variable can be CHAR or VARCHAR. When the variable is part of a table in SAS 9, VARCHAR is converted to CHAR at input. When the input variable is part of a table in CAS, VARCHAR is processed without conversion to CHAR. For more information, see VARCHAR Data Type in String Functions.

Restrictions The values of the VAR variable cannot contain delimiters that were added with the DQPARSE function or the DQPARSETOKENPUT function.

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

Note VAR variables of type VARCHAR are processed as VARCHARs, without conversion to CHAR at input.

See "DQPARSE Function" on page 128 for additional information.

"DQPARSETOKENPUT Function" on page 135 for additional information.

EXACT | MATCHDEF=
assigns a cluster number.

EXACT
assigns a cluster number based on an exact character match between values.

Restriction If you specify the EXACT= option, you cannot specify the MATCHDEF option, the MATCHCODE option, or the SENSITIVITY option.

MATCHDEF='*match-definition*'
specifies the match definition that is used to create the matchcodes for the specified source variable.

Restrictions The match definition must exist in the locale that is specified in the LOCALE option of the DQMATCH procedure.

If you specify the MATCHDEF option, you cannot specify the EXACT option, the MATCHCODE option, or the SENSITIVITY option.

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

Type The type of the MATCHCODE variable must be CHAR.

Restrictions The MATCHCODE option in the CRITERIA statement is not valid if you also specify the MATCHCODE option in the DQMATCH procedure.

If you are using multiple CRITERIA statements, either specify the MATCHCODE option in each CRITERIA statement, or specify the option only in the DQMATCH procedure. Specifying the MATCHCODE option in the DQMATCH procedure generates composite matchcodes.

SENSITIVITY=*sensitivity-level*
determines the amount of information in the resulting matchcodes. Higher sensitivity values create matchcodes that contain more information about the input values. Higher sensitivity levels result in a greater number of clusters, with fewer values in each cluster.

Default The default value is 85.

Range 50 to 95 inclusive.

Details

Matchcodes are created for the input variables that are specified in each CRITERIA statement. The resulting matchcodes are stored in the output variables that are named in the MATCHCODE option. The MATCHCODE option can be specified in the DQMATCH procedure or the CRITERIA statement.

Simple matchcodes are created when the CRITERIA statements specify different values for their respective MATCHCODE options. Composite matchcodes are created when two or more CRITERIA statements specify the same value for their respective MATCHCODE options.

To create matchcodes for a parsed character variable, specify the DELIMSTR option instead of the VAR option. In the MATCHDEF option, be sure to specify the name of the match definition. This definition 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.

Examples: DQMATCH Procedure

Example 1: Generate Composite Matchcodes

The following example uses the DQMATCH procedure to create composite matchcodes 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;
```

Output 8.1 *PROC Print Output*

Results Viewer - SAS Output

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

Details

The output data set, OUT_DB1, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the composite matchcode that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting matchcode contains two matchcode 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 matchcode values. Although the matchcode value for customer Bobby Becket is similar to the Cluster 1 matchcodes, the address difference caused it to be excluded in Cluster 1.

"Example 2: Matching Values Using Mixed Sensitivity Levels" on page 54 shows how the use of non-default sensitivity levels increases the accuracy of the analysis.

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

Example 2: Matching Values Using Mixed Sensitivity Levels

The following example is similar to "Example 1: Generate Composite Matchcodes" on page 53 in that it displays matchcodes 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 enables you to customize 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 names. To minimize false duplicates, minimal sensitivity is applied to the addresses.

```
/* 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;
```

Output 8.2 *PROC Print Output*

Results Viewer - SAS Output

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

Details

The output data set, OUT_DB2, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the matchcode that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting matchcode contains two matchcode 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: Generate Composite Matchcodes" on page 53, 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. This makes the matching less sensitive to variations in the address field. Therefore, the value Bobby Becket has now been included in Cluster 1.392 Main St. is considered a match with 392 S. Main St. PO Box 2270 and the other variations, this was not true at a sensitivity of 85.

Note: 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;
```

Output 8.3 *PROC Print Output*

Results Viewer - SAS Output

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

Details

The output data set OUT_DB3 includes the variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the matchcode that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting matchcode contains two matchcode 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 BeckettPaul Becker
```

A higher sensitivity level would not cluster these observations.

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

Example 4: Creating Matchcodes for Parsed Values

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

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

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

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

Output 8.4 *PROC Print Output*

Results Viewer - SAS Output

Look at the match codes from PROC DQMATCH

Obs	first	last	delimstr	MATCH_CD
1	Scott	James	/=/Scott/=//=/James/=//=/	CB4$$$$4J~$$$$$
2	James	Scott	/=/James/=//=/Scott/=//=/	43~$$$$C7B$$$$$
3	Ernie	Hunt	/=/Ernie/=//=/Hunt/=//=/	2P~$$$$_YP$$$$$
4	Brady	Baker	/=/Brady/=//=/Baker/=//=/	M3Y$$$$MY~$$$$$
5	Ben	Riedel	/=/Ben/=//=/Riedel/=//=/	Y8W$$$$M_P$$$$$
6	Sara	Fowler	/=/Sara/=//=/Fowler/=//=/	GWY$$$$4&Y&$$$$
7	Homer	Webb	/=/Homer/=//=/Webb/=//=/	LM$$$$$2BY$$$$$
8	Poe	Smith	/=/Poe/=//=/Smith/=//=/	4B~2$$$N$$$$$$$

Example 5: Clustering with Multiple CRITERIA Statements

The following example assigns cluster numbers based on a logical OR of two pairs of CRITERIA statements. Each pair of CRITERIA statements is evaluated as a logical AND. The cluster numbers are assigned based on a match between the customer name and address, or the organization name and address.

```
 /* Load the ENUSA locale. The system option DQSETUPLOC= is already
set.*/

   %dqload(dqlocale=(enusa))

   data customer;
      length custid 8 name org addr $ 20;
      input custid name $char20. org $char20. addr $char20.;

datalines;
   1  Mr. Robert Smith    Orion Star Corporation   8001 Weston Blvd.
   2                      The Orion Star Corp.     8001 Westin Ave
   3  Bob Smith                                   8001 Weston Parkway
   4  Sandi Booth         Belleview Software       123 N Main Street
   5  Mrs. Sandra Booth   Belleview Inc.           801 Oak Ave.
   6  sandie smith Booth  Orion Star Corp.         123 Maine Street
   7  Bobby J. Smythe     ABC Plumbing             8001 Weston Pkwy
   ;
   run;

   /* Generate the cluster data. Because more than one condition
      is defined, a variable named CLUSTER is created automatically */

   proc dqmatch data=customer
                out=customer_out;
      criteria condition=1 var=name sensitivity=85 matchdef='Name';
      criteria condition=1 var=addr sensitivity=70 matchdef='Address';
```

```
      criteria condition=2 var=org  sensitivity=85
matchdef='Organization';
      criteria condition=2 var=addr sensitivity=70 matchdef='Address';
   run;

   /* Print the result. */

   proc print data=customer_out noobs;
   run;
```

Output 8.5 *PROC Print Output*

Results Viewer - SAS Output

Look at the match codes from PROC DQMATCH

custid	name	org	addr	CLUSTER
3	Bob Smith		8001 Weston Parkway	1
2		The Orion Star Corp.	8001 Westin Ave	1
7	Bobby J. Smythe	ABC Plumbing	8001 Weston Pkwy	1
1	Mr. Robert Smith	Orion Star Corporation	8001 Weston Blvd.	1
6	sandie smith Booth	Orion Star Corp.	123 Maine Street	2
4	Sandi Booth	Belleview Software	123 N Main Street	2
5	Mrs. Sandra Booth	Belleview Inc.	801 Oak Ave.	.

Details

In the preceding output, the two rows in cluster 1 matched on name and address. The rows in cluster 2 matched on name and address as well as organization and address. The inclusion of Bobby J. Smythe in cluster 2 indicates either a data error or a need for further refinement of the criteria and conditions. The last row in the output did not receive a cluster number because that row did not match any other rows.

Note: This example is available in the SAS Sample Library under the name DQMLTCND.

Example 6: Generating Multiple Simple Matchcodes

The following example creates more than one simple matchcode with a single DQMATCH procedure step. The first example created a composite matchcode by specifying the MATCHCODE= option in the DQMATCH procedure statement. This example creates simple matchcodes by specifying the MATCHCODE= option in each CRITERIA statement. In addition, unlike the first example, which creates a cluster number, you cannot create a cluster number when generating multiple simple matchcodes.

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_db5 locale='ENUSA';
   criteria matchdef='Name' var=customer matchcode=mc_name;
   criteria matchdef='Address' var=address matchcode=mc_addr;
run;

/* Print the results. */
proc print data=out_db5;
run;
```

Output 8.6 *PROC Print Output*

Results Viewer - SAS Output

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

Details

The output data set, OUT_DB5, includes the new variables MC_NAME and MC_ADDR. Compare this to the result in example 1, where the same matchcode values were combined to form a composite matchcode in the MATCH_CD variable.

Using simple or composite matchcodes depends on the type of comparison that you need. If you want to compare names and addresses separately, generate separate matchcodes as shown in this example. If you want to do comparisons based on the combined Name and Address, generate a composite match code as shown in example 1.

See “Example 1: Generate Composite Matchcodes” on page 53 to compare the examples.

Note: This example is available in the SAS Sample Library under the name DQMCDFL2.

9

DQSCHEME Procedure

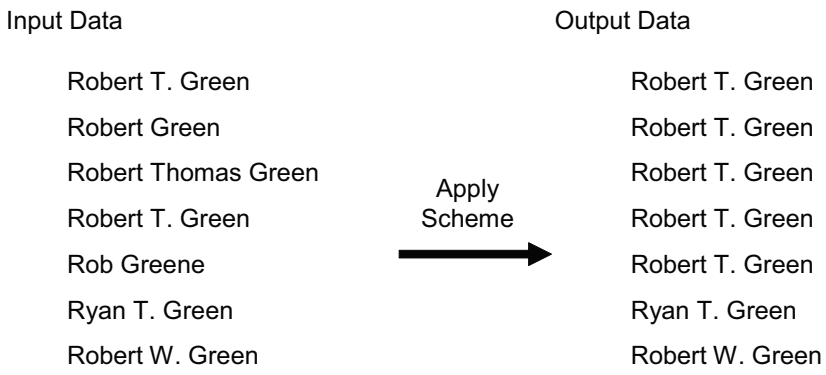
Overview: DQSCHEME Procedure

What Does the DQSCHEME Procedure Do?

PROC DQSCHEME 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 “CALL DQSCHEMEAPPLY Routine” on page 141.

The DQSCHEME procedure enables you to create and apply schemes that transform similar data values into the single most common value, as shown in the following diagram.

Figure 9.1 *Transform Similar Data Values*



The DQSCHEME procedure also analyzes and reports on the quality of your data.

Syntax: DQSCHEME Procedure

Restriction: Not valid in CAS

Interaction: VARCHAR data is supported when data is in CAS

```
PROC DQSCHEME DATA=input-data-set
   <QKB | NOQKB>
    OUT=output-data-set;
APPLY <options>;
CONVERT <options>;
CREATE <options>;
```

PROC DQSCHEME Statement

Creates scheme data sets and analysis data sets and applies schemes to input data sets.

Syntax

```
PROC DQSCHEME <DATA=input-data-set>
   < | >
    <OUT=output-data-set>;
```

Optional Arguments

QKB | NOQKB

QKB

specifies that all schemes are in QKB scheme file format. QKB scheme files can be edited using the feature-rich graphical user interface of the DataFlux Data Management Studio software.

NOQKB
specifies that all schemes are in SAS scheme file format.

Default QKB

Restrictions Always specify NOQKB when creating schemes in the z/OS operating environment.

In schemes stored in SAS format, data set labels are used to store meta options. Therefore, you should not specify data set labels in scheme data sets that are stored in SAS format. If you specify data set labels, you overwrite the scheme metadata.

See "Meta Options" on page 72.

DATA=*input-data-set*
When you use the CREATE statement to create schemes, the DATA option specifies the SAS data set from which one or more schemes are built. When you use the APPLY statement to apply existing schemes, the DATA option specifies the SAS data set that is transformed by the scheme.

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

OUT=*output-data-set*
specifies the output data set.

Interactions If the specified data set does not exist, the DQSCHEME procedure creates it.

If you use one or more APPLY statements, you must use the OUT option to specify the name of the output data set.

If you specify the OUT option without an APPLY statement, an empty output data set is created.

Results are written to the output data set after all schemes have been applied.

APPLY Statement

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

Syntax

APPLY <LOCALE=*locale-name*>
<MATCH-DEFINITION=*match-definition*>
<MODE= | >
<SCHEME=*scheme-name*>
<SCHEME_LOOKUP = | | >
<SENSITIVITY=*sensitivity-level*>
<VAR=*variable-name*>;

Optional Arguments

LOCALE=*locale-name*
specifies the ISO code of the locale that supports the specified match defintion. This value replaces the default value that could be present in the scheme.

MATCH-DEFINITION=*match-definition*
specifies the name of the match definition, in the specified locale, that is referenced to generate matchcodes for the input variable.

MODE= |
specifies a mode of scheme application. This information overrides any default mode that is specified in the CREATE statement for the scheme, or in the DQSCHEMEAPPLY function or CALL routine.

ELEMENT
specifies that each element in each value of the input variable is compared to the data values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the matchcode for each element is compared to matchcodes generated for each element, in each DATA variable value in the scheme.

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

SCHEME=*scheme-name*
identifies the scheme that transforms the input variable. In all the operating environments other than z/OS, schemes using QKB scheme file format are identified by specifying a fileref or a fully qualified name that ends in `.sch.qkb`. z/OS requires a SAS-format scheme file.

SCHEME_LOOKUP = | |
specifies the method of applying the scheme to the input variable, replacing any default method in the scheme file. Valid values are defined as follows:

EXACT
(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 before comparison.

IGNORE_CASE
specifies that capitalization is to be ignored when input values are compared to the DATA values in the scheme.

Interaction Any adjacent blank spaces in the input values are replaced with single blank spaces before comparison.

USE_MATCHDEF
specifies that comparisons are to be made between the matchcodes of the input values and the matchcodes in the scheme.

Interactions Specifying USE_MATCHDEF enables the options LOCALE, MATCHDEF, and SENSITIVITY, which can be used to override the default values that could be specified in the scheme.

A transformation occurs when the matchcode of an input value is identical to the matchcode of a DATA value in the scheme.

See "Meta Options" on page 72 for additional information.

SENSITIVITY=*sensitivity-level*
specifies the amount of information in the resulting matchcodes. This value replaces the sensitivity value that can be specified in the scheme.

Default 85

Range 50-95

Interaction This meta option is used at apply time only when SCHEME_LOOKUP=MATCHDEF.

See "Meta Options" on page 72 for additional information.

VAR=*input-variable-name*
specifies the name of the input variable or source column that is analyzed and transformed. The type of the input variable can be CHAR or VARCHAR. When the variable is part of a table in SAS 9, VARCHAR is converted to CHAR at input. When the input variable is part of a table in CAS, VARCHAR is processed without conversion to CHAR. For more information, see VARCHAR Data Type in String Functions.

CONVERT Statement

Converts schemes between SAS and QKB scheme file formats.

Requirement: All options are required.

Syntax

CONVERT < | >
<IN=*input-data-set*>
<OUT=*output-data-set*>;

Required Arguments

QKBTOSAS | SASTOQKB
specify QKBTOSAS to convert a scheme in QKB scheme file format to SAS format. Specify SASTOQKB to convert a scheme in SAS format to QKB scheme file format. Schemes in SAS format are created with the CREATE statement using the NOQKB option in the DQSCHEME procedure.

CAUTION **In the z/OS operating environment, specify QKBTOSAS only. In z/OS, schemes in QKB scheme file format can be applied but not created.**

IN=*scheme-data-set*
identifies the existing scheme data set that is to be converted.

If QKBTOSAS is specified, then the value must be the name of a fileref that references a fully qualified path in lowercase that ends in **.sch.qkb**.

If SASTOQKB is specified, then the value must be a one-level or two-level SAS data set name.

Note In the z/OS operating environment, the PDS specification has no special naming requirements.

OUT=*converted-scheme-data-set*
specifies the name of the data set with the converted scheme.

Requirements If SASTOQKB is specified, the value must be the name of a fileref. This fileref references a fully qualified path in lowercase that ends in `.sch.qkb`.

If QKBTOSAS is specified, the value must be a one-level or two-level SAS data set name.

Note The z/OS operating environment, the PDS specification has no special naming requirements.

CREATE Statement

Creates a scheme or an analysis data set.

Syntax

CREATE =<ANALYSIS=*analysis-data-set* >
<INCLUDE_ALL>
<LOCALE=*locale-name*>
<MATCHDEF=*match-definition* >
<MODE= | >
<SCHEME=*scheme-name*>
<SCHEME_LOOKUP= | | >
<SENSITIVITY=*sensitivity-level*>
<VAR=*input-character-variable*>;

Optional Arguments

ANALYSIS=*analysis-data-set*
Names the output data set that stores analytical data.

Restriction 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. This includes input variables with these conditions:

- with unique matchcodes
- that were not transformed
- that did not receive a cluster number

Note The INCLUDE_ALL option is *not* set by default.

LOCALE=*locale-name*
specifies the ISO code of the locale that supports the input data set and the specified match definition. This default is specified in the scheme. The default can be superseded by a different locale that is specified in the APPLY statement or in the DQSCHEMEAPPLY function or CALL routine. The value can be specified as a quoted string, a variable whose value is the locale name, or an expression that evaluates to the locale name.

The specified locale must be loaded into memory as the default locale.

Default The first locale that is specified in the DQLOCALE= system option.

MATCHDEF=*match-definition*
names the match definition in the specified locale that is used to establish cluster numbers. You can specify any valid match definition.

The value of the MATCHDEF option is stored in the scheme as a meta option. This provides a default match definition when a scheme is applied. This meta option is used only when SCHEME_LOOKUP=MATCHDEF. The default value that is supplied by this meta option can be superseded by match definitions that are specified in the APPLY statement or the DQSCHEMEAPPLY CALL routine.

Tip Use definitions whose names end in `(SCHEME BUILD)` when using the ENUSA locale. These match definitions yield optimal results in the DQSCHEME procedure.

See "Meta Options" on page 72 for additional information.

MODE= |
specifies a mode of scheme application. This information is stored in the scheme as metadata, which specifies a default mode when the scheme is applied. The default mode is superseded by a mode in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine.

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 matchcode for each element is compared to matchcodes generated for each element in each DATA variable value in the scheme.

PHRASE
(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 matchcode for the entire input value is compared to matchcodes that are generated for each data value in the scheme.

SCHEME=*scheme-name*
specifies the name or the fileref of the scheme that is created. The fileref must reference a fully qualified path with a filename that ends in `.sch.qkb`. Lowercase letters are required. To create a scheme data set in QKB scheme file format, specify the QKB option in the DQSCHEME procedure.

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

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

See "Syntax" on page 64 for additional information.

CAUTION **In the z/OS operating environment, specify only schemes that use SAS formats. QKB schemes can be applied, but not created in the z/OS operating environment.**

SCHEME_LOOKUP= | |

specifies the default method of applying the scheme to the input variable. Valid values are defined as follows:

EXACT

This default value specifies that the input variable will 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 before comparison.

IGNORE_CASE

specifies that capitalization is to be ignored when input values are compared to the DATA values in the scheme.

Interaction Any adjacent blank spaces in the input values are replaced with single blank spaces before comparison.

USE_MATCHDEF

specifies that comparisons are to be made between the matchcodes of the input values and the matchcodes of the DATA values in the scheme.

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

A transformation occurs when the matchcode of an input value is identical to the matchcode of a DATA value in the scheme.

The value of the SCHEME_LOOKUP option is stored in the scheme as a meta option. This specifies a default lookup method when the scheme is applied. The default supplied by this meta option can be superseded by a lookup method that is specified in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine.

See "Meta Options" on page 72 for additional information.

SENSITIVITY=*sensitivity-level*

determines the amount of information that is included in the matchcodes that are generated during the creation and perhaps the application of the scheme. The value of the SENSITIVITY option is stored in the scheme as a meta option. This provides a default sensitivity value when the scheme is applied.

Higher sensitivity values generate matchcodes that contain more information. These matchcodes generally result in the following:

- fewer matches
- greater number of clusters

- fewer values in each cluster

Default	85
Range	50-95
Interactions	The default value supplied by this meta option is superseded by a sensitivity value specified in the APPLY statement, or in the DQSCHEMEAPPLY CALL routine.
	This meta option is used at apply time only when SCHEME_LOOKUP=MATCHDEF.
See	"Meta Options" on page 72 for additional information.

VAR=*input-character-variable*
specifies the name of the input variable or source column that is analyzed to create the scheme. The type of the input variable can be CHAR or VARCHAR. When the variable is part of a table in SAS 9, VARCHAR is converted to CHAR at input. When the input variable is part of a table in CAS, VARCHAR is processed without conversion to CHAR. For more information, see VARCHAR Data Type in String Functions.

Usage: DQSCHEME Procedure

Creating Schemes

Schemes are data sets that are first created from a source table, and then later applied to that source table to standardize the values in that table.

Scheme data sets are created with the CREATE statement. The CREATE statement uses matching codes to define groups of similar data values. The values that occur most frequently in each group become the survivor values for each group. When the scheme is applied, the survivor values replace the other values in the group.

Note: An error message is generated if the length of an input value exceeds 1024 bytes.

Scheme data sets are created in SAS format or in QKB scheme file format. QKB scheme file format is recognized by SAS and by the DataFlux Data Management software.

The differences between schemes in SAS format and QKB format are defined as follows:

- Schemes can be created and applied in in SAS format or QKB format.
- Schemes in SAS format and QKB format can be displayed with the SAS table viewer.

- DataFlux Data Management Studio software can create, apply, and edit schemes in QKB format.
- In the z/OS operating environment, the SAS Data Quality Server software can create, apply, and display schemes in SAS format. Schemes in QKB file format can be applied.

To convert scheme data sets from one format to the other, use the CONVERT statement in PROC DQSCHEME.

Generating Analysis Data Sets

Analysis data sets show the groupings of like data values in the scheme-building process. These are the groupings from which the standard value is selected. The data sets are generated by specifying the ANALYSIS option in the CREATE statement of the DQSCHEME procedure. The analysis data sets enable you to experiment with different options to create a scheme that provides optimal data cleansing.

The key to optimizing a scheme is to choose a sensitivity value that is most suitable for 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.

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.

The analysis data set 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 represents the groupings of data values that are similar based on the selected sensitivity. One standard value is selected from each cluster, based on the value with the highest COUNT (frequency).

Specify the INCLUDE_ALL option in the CREATE statement to include all input values in the scheme. This includes the unique input values that did not receive a cluster number in the analysis data set.

See “Creating Schemes” on page 71 for additional information.

Meta Options

Meta options are stored in the scheme when the scheme is created. The options provide default values for certain options of the DQSCHEME procedure's APPLY statement. The meta options also store 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.

Meta options for the match definition and sensitivity value are valid only when the scheme is applied with matchcode lookup and 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 QKB scheme file 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:

'lookup-method' 'apply-mode' 'sensitivity-level' 'match-definition'

lookup-method

EM specifies that the default value of the SCHEME_LOOKUP option or argument is EXACT. In order for an input value to be transformed, that value must exactly match a DATA value in the scheme.

IC specifies that SCHEME_LOOKUP=IGNORE_CASE.

UM specifies that SCHEME_LOOKUP=USE_MATCHDEF. Matchcodes are created and compared for all input values and all DATA values in the scheme.

apply-mode

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

P specifies that MODE=PHRASE.

sensitivity-level

the amount of information in the matchcodes that is generated when SCHEME_LOOKUP=USE_MATCHDEF.

Valid values range from 50 to 95.

match-definition

the name of the default match definition that is used when the value of the SCHEME_LOOKUP option is USE_MATCHDEF.

For example, the meta options string, `'UM' 'P' '80' 'NAME'`, specifies that the scheme:

- lookup method is matchcode
- the apply-mode is by phrase
- the sensitivity-level is 80
- the match-definition is NAME

Examples: DQSCHEME Procedure

Example 1: Creating an Analysis Data Set

Overview

This example generates an analysis of the STATE variable in the VENDORS data set.

Note: You do not have to create a scheme to generate the analysis data set.

Note: The locale ENUSA is assumed to have been loaded into memory as part of the locale list.

For each value of the STATE variable, the analysis data set WORK.A_STATE shows 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.

Note: This example is available in the SAS Sample Library under the name DQANALYZ.

```
 /* 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
Dallas            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 Permanente
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;
```

Output 9.1 *PROC PRINT Output for DQANALYZ*

Results Viewer - SAS Output

Analysis of state name variations

Obs	COUNT	state	CLUSTER
1	2	District of Columbia	1
2	1	DC	1
3	1	MI	2
4	1	Michigan	2
5	1	GA	3
6	1	GEORGIA	3
7	3	CA	4
8	1	California	4
9	1	NC	.
10	2	NY	5
11	1	N.Y.	5
12	1	New York	5
13	1	VA	6
14	1	VIRGINIA	6
15	1	TX	7
16	1	Texas	7

Example 2: Creating Schemes

The following example generates three schemes in SAS format. The match definition for Organization is assumed to be in the QKB used for this code. 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: Creating an Analysis Data Set for the full data set.
*/

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

proc dqscheme data=vendors noqkb;
  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'
    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;
```

Details

Notice that this example did not create and immediately apply one or more schemes within the same step. After you create schemes, it is important that someone familiar with the data review 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, you would probably prefer Dallas to be the transformation value.

Note: This example is available in the SAS Sample Library under the name DQSASSCH.

Example 3: Creating Schemes for the QKB

Transformation schemes can be read by SAS and by the DataFlux Data Management Platform software. Generating QKB schemes is advantageous when you want to use DataFlux Data Management Studio to edit the schemes. The following example generates three schemes in QKB scheme file format. Note that the locale ENUSA is assumed to be loaded into memory as part of the locale list.

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

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

/* 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: Creating an Analysis Data Set for the full data set.
*/

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

proc dqscheme data=vendors qkb;
  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'
     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 match definition for Organization is assumed to be in the QKB used for this code. The locale ENUSA is assumed to be loaded into memory as part of the locale list.

Note: This example is available in the SAS Sample Library under the name DQAPPLY.

```
/* Create filerefs with required suffixes, and place them in an
   existing subdirectory. */
filename city 'c:\my schemes\city.sch.qkb';
filename state 'c:\my schemes\state.sch.qkb';
filename org 'c:\my schemes\org.sch.qkb';

 /* 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;

proc dqscheme data=vendors out=vendors_out qkb;
   create matchdef='City'
          var=city
          scheme=city
          locale='ENUSA';
   create matchdef='State/Province'
          var=state
          scheme=state
          locale='ENUSA';
   create matchdef='Organization'
          var=company
          scheme=org
          locale='ENUSA';
   apply var=city scheme=city;
   apply var=state scheme=state;
   apply var=company scheme=org;
run;

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

Output 9.2 *PROC PRINT Output for DQAPPLY*

Result after applying all three QKB format schemes

Obs	city	state	company
1	Detroit	MI	Ford Motor
2	Dallas	TX	Wal mart
3	Washington	District of Columbia	Federal Reserve Bank
4	San Jose	CA	Wal mart
5	New York	NY	Ernst & Young
6	Virginia Bch	VA	TRW INC - Space Defense
7	Dalas	TX	Wal mart
8	San Francisco	CA	Jackson Data Corp
9	New York	NY	Ernst & Young
10	Washington	District of Columbia	Federal Reserve Bank
11	New York	NY	Ernst & Young
12	San Francisco	CA	Jackson Data Corp
13	Atlanta	GA	Farmers Insurance Group
14	RTP	NC	Kaiser Permantente
15	New York	NY	Ernest and Young
16	Virginia Bch	VA	TRW Space & Defense
17	Detroit	MI	Ford Motor
18	San Jose	CA	Jackson Data Corp
19	Washington	District of Columbia	Federal Reserve Bank
20	Atlanta	GA	Target

Details

Note that the SCHEME arguments in this case reference the three filerefs assigned above. All of the schemes will be created, and then they will all be applied, to produce the result in the VENDORS_OUT data set.

Note that you do not specify a locale when you apply a scheme. The application of a scheme simply compares an input value from your input data set against the values in the scheme, so no locale processing is done.

Also note that the schemes are being created and applied within the same step, but only for sample purposes. It is not recommended that you create and then immediately apply a scheme without reviewing the scheme first. If you are using QKB format schemes, the most common approach would be to create, review, and edit your schemes from within SAS Data Integration Studio or DataFlux Data Management Studio. After you review the schemes, you can apply them with the APPLY statement of PROC DQSCHEME (shown here), or with the DQSCHEMAPPLY function or CALL routine.

SCHEME_LOOKUP=EXACT (the default) specifies that the value in the scheme replaces the input value in the output data set. This occurs when an exact match is found between the input value and a DATA value in the scheme. When you use the default scheme apply mode MODE=PHRASE, each input value is compared to the DATA values in the scheme.

10

QKB Procedure

Overview: QKB Procedure

What Does the Quality Knowledge Base (QKB) Procedure Do?

The QKB procedure lists the name of each QKB that is stored in the QKB Repository, either in the console or in a data set.

Syntax: QKB Procedure

PROC QKB

<*NOPRINT*> ;

LIST <OUT=*output-data-set*>;

PROC QKB Statement

The QKB procedure lists each QKB that is stored in the QKB Repository.

Restriction: Valid for SAS Viya 4.0.1 and later.

Syntax

PROC QKB

<*NOPRINT*>
LIST <OUT=*output-data-set*>
;

Optional Arguments

NOPRINT
stops the output from being displayed in the console.

OUT=*output-data-set*
specifies the storage location of the QKB data set.

Default If OUT is not specified, then the output is written only to the SAS log/console.

Examples: QKB Procedure

Example 1: List Available QKBs

This example displays the contents of the QKB Repository in the console.

```
proc QKB;
   LIST;
   run;
```

Example 2: Generate a Data Set with Available QKBs

This example displays the contents of the QKB Repository in a data set designated optionally by OUT.

```
proc QKB
   LIST OUT=output-data-set;
  run;
```

11

AUTOCALL Macros

Dictionary

%DQLOAD AUTOCALL Autocall Macro

Sets system option values and loads locales into memory.

Restriction: Not valid in CAS

Syntax

%DQLOAD(DQSETUPLOC=" *path-specification*", DQLOCALE=(*locale* <, *locale2, locale3...*>), <DQINFO=*0*|*1*>);

Summary of Optional Arguments

status information

DQINFO=0 | 1

Required Arguments

DQSETUPLOC=*"path-specification"*

specifies the fully qualified path of the root directory of the Quality Knowledge Base. The Quality Knowledge Base contains the locales that are specified in the DQLOAD argument.

DQLOCALE=(*locale-1* < *locale2, locale3…*>)

specifies an ordered list of locales to load into memory. All locales must be present in the QKB that is specified by the DQSETUPLOC argument.

Optional Argument

DQINFO=0 | 1

DQINFO=1 specifies that the SAS log receives additional information about the locale load operation.

Default 0

Details

Specify the %DQLOAD AUTOCALL macro at the beginning of each data cleansing program. This ensures that the proper list and order of locales is loaded into memory before you cleanse data.

Specify the %DQLOAD macro before data cleansing, instead of at SAS invocation, using an AUTOEXEC or configuration file, to preserve memory and shorten the duration of the SAS invocation. Doing so is particularly beneficial when the SAS session is not 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. When SAS is installed, the value of the DQSETUPLOC= option is set to point to the default location of the sample Quality Knowledge Base. To cleanse data, it is important to point to a production QKB rather than the sample QKB. The sample QKB consists of a limited set of locales and definitions.

In CAS, use the DQLOAD= session option to list locales and load locales into memory.

Example

In the following example, DQLOCALE specifies an ordered list of locales to load into memory. DQSETUPLOC specifies the location of the Quality Knowledge Base.

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

%DQPUTLOC AUTOCALL Autocall Macro

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

Restriction: Not valid in CAS

Syntax

%DQPUTLOC (*locale*);

Summary of Optional Arguments

lists related parse definition

PARSEDEFN=0 | 1

shortens length of log

SHORT=0 | 1

specifies the local of interest

locale

Optional Arguments

locale

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

Default	If no locale is specified, information is logged for the default locale.
Requirement	The specified locale must be loaded into memory before this macro is called.

PARSEDEFN=0 | 1

PARSEDEFN=1 lists the related parse definition with each gender analysis definition and each match definition , if a parse definition exists.

Default 1

SHORT=0 | 1

SHORT=1 shortens the length of the entry in the SAS log. Descriptions of how the definitions are used are not added to the log.

Default 0

Details

The %DQPUTLOC AUTOCALL macro displays the contents of the specified locale in the SAS log. Locale contents include all definitions, parse tokens, related functions, and the names of the parse definitions that are related to each match definition and gender analysis definition. Knowing the related parse definitions enables the creation of parsed character values. See “DQPARSETOKENPUT Function” on page 135 for additional information.

It also enables the creation of matchcodes for parsed character values. See “DQMATCHPARSED Function” on page 126 for additional information.

Load the specified locale into memory with %DQLOAD before you submit %DQPUTLOC.

Example

This example displays in the SAS log definitions, related parse definitions, and related SAS Data Quality Server functions for the ENUSA locale.

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

See Also

- "DQLOCALEINFOGET Function" on page 122
- "DQLOCALEINFOLIST Function" on page 122

%DQUNLOAD AUTOCALL Autocall Macro

Unloads all locales to increase the amount of free memory.

Restriction: Not valid in CAS

Requirement: After unloading locales from memory, load locales with the %DQLOAD AUTOCALL macro before running data quality operations.

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 data cleansing programs.

Macro Resources Autocall Macro

Macro Resources

See: *SAS Macro Language: Reference*
SAS Language Reference: Concepts

12

Functions and CALL Routines

Overview

The functions and CALL routines in the SAS Data Quality Server software enable you to cleanse data and access DataFlux Data Management Servers.

The functions and CALL routines are listed alphabetically and by category. Each function and CALL routine has a link to a detailed description and syntax.

Note: The SAS Data Quality Server functions and CALL routines are available in the Expression Builder of SAS Data Integration Studio software and SAS Enterprise Guide software.

Functions Supported in CAS

The following functions are supported by programs that run in SAS Cloud Analytic Services (CAS).

DQCASE	DQEXTINFOGET
DQEXTRACT	DQEXTTOKENGET
DQEXTTOKENPUT	DQGENDER
DQGENDERINFOGET	DQGENDERPARSED

DQIDENTIFY	DQIDENTIFYIDGET
DQIDENTIFYINFOGET	DQIDENTIFYMULTI
DQLOCALEGUESS	DQLOCALEINFOGET
DQLOCALEINFOLIST	DQLOCALESCORE
DQMATCH	DQMATCHINFOGET
DQMATCHPARSED	DQPARSE
DQPARSE CALL routine	DQPARSEINFOGET
DQPARSETOKENGET	DQPARSETOKENPUT
DQPATTERN	DQSTANDARDIZE
DQTOKEN	DQVER
DQVERQKB	

Functions Listed Alphabetically

- The DMSRVBATCHJOB function runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.
- The DMSRVCOPYLOG function copies a job's log file from a DataFlux Data Management Server.
- The DMSRVDELETELOG function deletes a job's log file from a DataFlux Data Management Server.
- The DMSRVJOBSTATUS function returns the status of a job that was submitted to a DataFlux Data Management Server.
- The “DMSRVKILLJOB Function” function. terminates a job that is running on a DataFlux Data Management Server.
- The “DMSRVPROFILEJOB Function” function generates a profile from a Data Management Server repository.
- The “DMSRVUSER Function” function registers credentials (user name and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.
- The “DMSRVVER Function” function returns the version of the DataFlux Data Management Server.
- The DQCASE function returns a character value with standardized capitalization.

- The DQEXTINFOGET function returns the token names that are supported by an extraction definition.
- The DQEXTRACT function returns a delimited string of extraction token values.
- The DQEXTTOKENGET function returns an extraction token value from a delimited string of extraction token values.
- The DQEXTTOKENPUT function inserts an extraction token value into a delimited string of extraction token values and returns the updated delimited string.
- The DQGENDER function returns a gender determination from the name of an individual.
- The DQGENDERINFOGET function returns the name of the parse definition that is associated with a specified gender analysis definition.
- The DQGENDERPARSED function returns the gender of an individual from a delimited string of parse token values.
- The DQIDENTIFY function returns the highest-scoring identity name for a character value.
- The DQIDENTIFYIDGET function returns an identification analysis score for a given identity from a delimited string of identification analysis scores.
- The DQIDENTIFYINFOGET function returns the names of all of the identities that are supported by a given identification analysis definition.
- The DQIDENTIFYMULTI function returns a delimited list of identification analysis scores from a character value.
- The DQLOCALEGUESS function returns the ISO code of the highest-scoring locale of a character value.
- The DQLOCALEINFOGET function returns a list of the locales that are loaded into memory.
- The DQLOCALEINFOLIST function returns a count of definitions and displays the names of definitions for a type of definition in a locale.
- The DQLOCALESCORE function returns a locale confidence score for an input character value.
- The DQMATCH function returns a matchcode from a character value.
- The DQMATCHINFOGET function returns the name of the parse definition that is associated with a match definition.
- The DQMATCHPARSED function returns a matchcode from a parsed character value.
- The DQOPTSURFACE function reveals or hides non-surfaced definitions.
- The DQPARSE CALL routine returns a delimited string of parse token values and a status flag.
- The DQPARSE function returns a delimited string of parse token values..
- The DQPARSEINFOGET function returns the names of the tokens that are supported by a parse definition.
- The DQPARSEINPUTLEN function sets the default length of parsed input, and returns a string indicating its previous value.
- The DQPARSESLIMIT function sets a limit on resources consumed during parsing.

- The DQPARSESCOREDEPTH function specifies how deeply to search for the best parsing score.
- The DQPARSETOKENGET function returns a parse token value from a delimited string of parse token values.
- The DQPARSETOKENPUT function inserts a parse token value into a delimited string of parse token values and returns the updated delimited string.
- The DQPATTERN function returns a pattern analysis from an input character value.
- The DQSCHEMEAPPLY CALL routine applies a scheme and returns a transformed value and a transformation flag.
- The DQSCHEMEAPPLY function applies a scheme and returns a transformed value after applying a scheme.
- The DQSTANDARDIZE function standardizes the casing, spacing, and format of certain words and abbreviations and returns an updated character value.
- The DQTOKEN function returns the value of a token from an input character value.
- The “DQVER Function” function returns the version of the SAS Data Quality engine.
- The DQVERQKB function returns the version of the currently loaded Quality Knowledge Base.

Functions Listed by Category

DataFlux Data Management Server Functions

- The DMSRVBATCHJOB function runs a DataFlux Data Management Studio process or data job on a DataFlux Data Management Server and returns a job identifier.
- The DMSRVCOPYLOG function copies a job's log file from a DataFlux Data Management Server.
- The DMSRVDELETELOG function deletes a job's log file from a DataFlux Data Management Server.
- The DMSRVJOBSTATUS function returns the status of a job that was submitted to a DataFlux Data Management Server.
- The DMSRVKILLJOB function terminates a job that is running on a DataFlux Data Management Server.
- The DMSRVPROFILEJOB function generates a profile from a repository on a DataFlux Data Management Server.
- The DMSRVUSER function registers credentials (user name and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.

- The DMSRVVER function returns the version of the DataFlux Data Management Server.

Case Functions

- The DQCASE function returns a character value with standardized capitalization.

Gender Analysis, Locale Guessing, and Identification Analysis Functions

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

- The DQGENDER function returns a gender determination from the name of an individual.
- The DQGENDERINFOGET function returns the name of the parse definition that is associated with a specified gender analysis definition.
- The DQGENDERPARSED function returns the gender of an individual from a delimited string of parse token values.
- The DQIDENTIFY function returns the highest-scoring identity for a character value.
- The DQIDENTIFYIDGET function returns the identification analysis score for a specified identity from a delimited string of identification analysis scores.
- The DQIDENTIFYINFOGET function returns the names of all of the identities that are supported by a given identification analysis definition.
- The DQIDENTIFYMULTI function returns all of the identification analysis scores of a character value.
- The DQLOCALEGUESS function returns the ISO code of the highest-scoring locale for a character value.
- The DQLOCALEINFOGET function returns a list of the locales that are loaded into memory.
- The DQLOCALEINFOLIST function returns a count of definitions and displays the names of definitions for a type of definition in a locale.
- The DQLOCALESCORE function returns a locale confidence score for an input character value.
- The “DQMATCHPARSED Function” function returns the names of the definitions in a locale and returns a count of those definitions.

Matching Functions

- The DQMATCH function returns a matchcode from a character value.

- The DQMATCHINFOGET function returns the name of the parse definition that is associated with a match definition.
- The DQMATCHPARSED returns a matchcode from a parsed character variable.

Parsing Functions

- The DQPARSE CALL routine returns a parsed character value and a status flag.
- The DQPARSE function returns a parsed character value.
- The DQPARSEINFOGET function returns the token names for the specified parse definition.
- The DQPARSEINPUTLEN function sets the default length of parsed input. DQPARSEINPUTLEN also returns a string indicating its previous value.
- The DQPARSERESLIMIT function sets a limit on resources consumed during parsing.
- The DQPARSESCORDEPTH function specifies how deeply to search for the best parsing score.
- The DQPARSETOKENGET function returns a token from a parsed character value.
- The DQPARSETOKENPUT function inserts a token into a parsed character value and returns the updated parsed character value.

Extraction Functions

- The DQEXTINFOGET function returns the token names in an extraction definition.
- The DQEXTRACT function returns a delimited string of extraction token values.
- The DQEXTTOKENGET function returns an extraction token value from a delimited string of extraction token values.
- The DQEXTTOKENPUT function inserts a token into an extraction character value and returns the updated extraction character value.

Pattern Analysis Functions

The DQPATTERN function returns a pattern analysis from an input character value.

Reporting Functions

- The DQGENDER function returns a gender determination from the name of an individual.
- The DQGENDERINFOGET function returns the name of the parse definition that is associated with a specified gender analysis definition.
- The DQGENDERPARSED function returns the gender of an individual from a delimited string of parse token values.
- The DQIDENTIFY function returns the highest-scoring identity for a character value.
- The DQIDENTIFYIDGET function returns the identification analysis score for a specified identity from a delimited string of identification analysis scores.
- The DQIDENTIFYGET function returns the names of all of the identities that are supported by an identification analysis definition.
- The DQIDENTIFYMULTI function returns a delimited list of identification analysis scores from a character value.
- The DQLOCALEGUESS function returns the ISO code of the highest-scoring locale of a character value.
- The DQLOCLAEINFOGET function returns a list of the locales that are loaded into memory.
- The DQLOCALEINFOLIST function returns a count of definitions and displays the names of definitions for a type of definition in a locale.
- The DQLOCALESCORE function returns a locale confidence score for an input character value.
- The DQMATCH function returns a matchcode from a character value.
- The DQMATCHINFOGET function returns the name of the parse definition that is associated with a match definition.
- The DQMATCHPARSED function returns a matchcode from a parsed character value.
- The DQPARSE CALL routine returns a delimited string of parse token values and a status flag.
- The DQPARSE on page 128 function returns a delimited string of parse token values.
- The DQPARSEINFOGET function returns the names of the tokens that are supported by a parse definition.
- The DQPARSETOKENGET function returns a parse token value from a delimited string of parse token values.
- The DQPARSETOKENPUT function inserts a parse token value into a delimited string of parse token values and returns the updated delimited string.
- The DQPATTERN function returns a pattern analysis from an input character value.
- The DQSCHEMEAPPLY CALL routine applies a scheme and returns a transformed value and a transformation flag.
- The DQSCHEMEAPPLY function applies a scheme and returns a transformed value after applying a scheme.
- The DQSTANDARIZE function standardizes the casing, spacing, and format of certain words and abbreviations and returns an updated character value.

- The DQTOKEN function returns the value of a token from an input character value.
- The DQVER function returns the version of the SAS Data Quality engine.
- The DQVERQKB function returns the version of the currently loaded Quality Knowledge Base.

Scheme Functions and CALL Routines

- The DQSCHEMEAPPLY function applies a scheme and returns a transformed value.
- The DQSCHEMEAPPLY CALL routine applies a scheme and returns a transformed value and a transformation flag.

Standardization Functions

- The DQSTANDARDIZE function standardizes the casing, spacing, and format of certain words and abbreviations and returns an updated character value.

Dictionary

DMSRVBATCHJOB Function

Runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.

Valid in:	SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.
Restriction:	Not valid in SAS Viya CAS.
Requirements:	If specified, the locale must be loaded into memory as part of the locale list. The character variable that receives the return value must have a minimum length of 52.

Syntax

DMSRVBATCHJOB(*job-name*, *host*, *port* <,*parameter-list*>)

Required Arguments

job-name
the DataFlux Data Management Studio job or process as it exists on the specified DataFlux Data Management Studio Server.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvBatchJob('jobname');

/* Localhost is used for the host */
   dmsrvBatchJob(jobname, '', 21036);

/* Correct */
   dmsrvBatchJob(jobname, 'http://myhost.unx.com', 21036);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvBatchJob('jobname');

/* Port 21036 or 21037 is used */
   dmsrvBatchJob('jobname', 'http://myhost.unx.com', 0);

/* Correct */
   dmsrvBatchJob('jobname', 'http://myhost.unx.com', 21036);
```

Optional Argument

parameter-list

the variable list of name and value pairs, where each name and value pair in the list must be defined as an input to the job.

Details

The DMSRVBATCHJOB function returns a job-identifier. The return value is either a job identifier of up to 52 characters or the value MISSING. Use the job identifier in subsequent function calls to manage the job, using DMSRVJOBSTATUS, DMSRVCOPYLOG, DMSRVDELETELOG, and DMSRVKILLJOB.

You can specify any number of name and value pairs.

Example: DMSRVBATCHJOB Function

The following example runs a job on a DataFlux Data Management Server.

```
data _null_;
```

```
     jobid = dmsrvBatchJob  ('myjob.djf', 'http://myhost.unx.com',
21036);
run;
```

See Also

- "DMSRVCOPYLOG Function" on page 97
- "DMSRVDELETELOG Function" on page 98
- "DMSRVJOBSTATUS Function" on page 100

DMSRVCOPYLOG Function

Copies a job's log file from a DataFlux Data Management Server to a local host.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DMSRVCOPYLOG(*job-ID*, *host*, *port*, *filename*)

Required Arguments

job–ID

identifies the job that is submitted to a DataFlux Data Management Server. The identifier is previously returned by a function such as DMSRVBATCHJOB.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvCopyLog('jobid');

/* Localhost is used for the host */
   dmsrvCopyLog('jobid', '', 21036, 'filename');

/* Correct */
   dmsrvCopyLog('jobid', 'http://myhost.unx.com',
    21036, 'filename');
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvCopyLog('jobid');

/* Port 21036 or 21037 is used */
   dmsrvCopyLog('jobid', 'http://myhost.unx.com',
  0, 'filename');

/* Correct */
   dmsrvCopyLog('jobid', 'http://myhost.unx.com',
  21036, 'filename');
```

filename
identifies where the log file is copied on the local host.

Details

To capture log information for a particular job, use the DMSRVJOBSTATUS function to ensure that the job is finished before you copy the log.

Return values are 0 (log copied successfully) or 1 (log failed to copy).

Example: DMSRVCOPYLOG Function

The following example copies a log file from a DataFlux Data Management Server. The log file is generated when the server runs a job. The job identifier is returned in the function that runs the job.

```
copyrc= dmsrvCopyLog(jobid,'http://myhost.unx.com',
5001,'dmServer1.log');
```

See Also

- “DMSRVBATCHJOB Function” on page 95
- “DMSRVDELETELOG Function” on page 98
- “DMSRVJOBSTATUS Function” on page 100

DMSRVDELETELOG Function

Deletes a job's log file from a DataFlux Data Management Server.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DMSRVDELETELOG(*job-ID*, *host*, *port*)

Required Arguments

job-ID

identifies the job submitted to a DataFlux Data Management Server. The identifier is set by a function such as DMSRVBATCHJOB.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvDeleteLog('jobid');

/* Localhost is used for the host */
   dmsrvDeleteLog('jobid', '', 21036);

/* Correct */
   dmsrvDeleteLog('jobid', 'http://myhost.unx.com', 21036);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvDeleteLog('jobid');

/* Port 21036 or 21037 is used */
   dmsrvDeleteLog('jobid', 'http://myhost.unx.com', 0);

/* Correct */
   dmsrvDeleteLog('jobid', 'http://myhost.unx.com', 21036);
```

Details

The log file is created after the job terminates. Use DMSRVJOBSTATUS to ensure that the log file is available for deletion.

DMSRVDELETELOG does not delete local copies of the job's log file.

Return values are 0 (log deleted successfully) or 1 (log failed to delete).

Example: DMSRVDELETELOG FUNCTION

The following example deletes a log file from a DataFlux Data Management Server. The log file is created when the server runs a job. The job identifier is returned in the function that runs the job.

```
delrc= dmsrvDeleteLog(jobid,'http://myhost.unx.com', 5001);
```

See Also

- "DMSRVBATCHJOB Function" on page 95
- "DMSRVCOPYLOG Function" on page 97
- "DMSRVJOBSTATUS Function" on page 100

DMSRVJOBSTATUS Function

Returns the status of a job that was submitted to a DataFlux Data Management Server.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DMSRVJOBSTATUS(*job-ID*, *host*, *port*, *time-out*, *interval*)

Required Arguments

job-ID

identifies the job that was submitted to a DataFlux Data Management Server. The identifier is previously set by a function such as DMSRVBATCHJOB.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvJobStatus('jobid');

/* Localhost is used for the host */
   dmsrvJobStatus('jobid', '', 21036, 20, 5);

/* Correct */
   dmsrvJobStatus('jobid', 'http://myhost.unx.com',
  21036, 20, 5);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvJobStatus('jobid');

/* Port 21036 or 21037 is used */
   dmsrvJobStatus('jobid', 'http://myhost.unx.com',
  0, 20, 5);

/* Correct */
   dmsrvJobStatus('jobid', 'http://myhost.unx.com',
  21036, 20, 5);
```

time-out
a time in seconds that determines when status information is returned from the host. Valid values are defined as follows:

-1	returns status information about when the job is finished. Return values are 0 (job completed successfully) or 1 (job failed). This value invalidates the *interval* argument.
0	returns status information immediately. Return values are 0 (job completed successfully), 1 (job failed), 2 (job running), or 3 (job in queue awaiting execution). When the *time-out* argument is 0, the value of the *interval* argument is not valid.
greater-than-zero	specifies a time limit for the *interval* argument. If the job is still running after the *time-out* value, another value is returned only when the job is finished.

interval
the repeat period for the return of status information, within the limit that is imposed by the *time-out* argument.

Details

Use the DMSRVJOBSTATUS function to return job status information instantly, periodically, or at the completion of the job. With an *interval* of 20 and a *time-out* of 60, DMSRVJOBSTATUS returns status information up to four times. After 60 seconds, the last return value is provided at the completion of the job.

Return values are 0 (job completed successfully), 1 (job failed), 2 (job running), or 3 (job in queue awaiting execution).

Example: DMSRVJOBSTATUS Function

The following example returns a status number for a job that ran or is running on a DataFlux Data Management Server. The job identifier was returned by the function that ran the job. Status information is returned in 20 seconds or less, depending on the termination of the job. Job status is checked every 5 seconds.

```
status= dmsrvJobStatus(jobid,'http://myhost.unx.com', 5001, 20, 5);
```

See Also

- "DMSRVBATCHJOB Function" on page 95
- "DMSRVDELETELOG Function" on page 98
- "DMSRVKILLJOB Function" on page 102

DMSRVKILLJOB Function

Terminates a job that is running on a DataFlux Data Management Server.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DMSRVKILLJOB(*job-ID*, *host*, *port*)

Required Arguments

job-ID

identifies the job submitted to a DataFlux Data Management Server. The identifier is set by a function such as DMSRVBATCHJOB.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvKillJob('jobid');

/* Localhost is used for the host */
   dmsrvKillJob('jobid', '', 21036);

/* Correct */
   dmsrvKillJob('jobid', 'http://myhost.unx.com', 21036);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvKillJob('jobid');
```

```
/* Port 21036 or 21037 is used */
   dmsrvKillJob('jobid', 'http://myhost.unx.com', 0);

/* Correct */
   dmsrvKillJob('jobid', 'http://myhost.unx.com', 21036);
```

Details

The DMSRVKILLJOB function terminates a job. Use the DMSRVJOBSTATUS function to determine whether a job is still running. Return values are 0 (job terminated) or 1 (job failed to terminate).

Example: DMSRVKILLJOB Function

The following example terminates a job that is running on a DataFlux Data Management Server. The job identifier is returned by the function that ran the job. Status information is returned in 20 seconds or less, depending on the termination of the job. Job status is checked every 5 seconds.

```
killrc= dmsrvKillJob(jobid,'http://myhost.unx.com',5001);
```

See Also

- "DMSRVBATCHJOB Function" on page 95
- "DMSRVJOBSTATUS Function" on page 100

DMSRVPROFILEJOB Function

Generates a profile from a Data Management server repository.

Valid in:	SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.
Restriction:	Not valid in SAS Viya CAS.
Requirement:	The character variable that receives the return value must have a minimum length of 52 characters.

Syntax

DMSRVPROFILEJOB(*job-name*, *host*, *port*, *append-flag* <,*description-character*>)

Required Arguments

job-name
identifies the DataFlux Data Management Profile job as it exists on the specified DataFlux Data Management Server.

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Incorrect use of function arguments */
   dmsrvProfileJob('jobname');

/* Localhost is used for the host */
   dmsrvProfileJob(jobname, '', 21036, 0);

/* Correct */
   dmsrvProfileJob(jobname, 'http://myhost.unx.com',
  21036, 0);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Incorrect use of function arguments */
   dmsrvProfileJob('jobname');

/* Port 21036 or 21037 is used */
   dmsrvProfileJob('jobname', 'http://myhost.unx.com',
  0, 1);

/* Correct */
   dmsrvProfileJob('jobname', 'http://myhost.unx.com',
  21036, 1);
```

append-flag

appends or overwrites job results.

0 appends job results below any existing content in the results file.

1 overwrites any existing content in the results file.

Optional Argument

description-character

identifies a character variable whose value describes the current run of the job. The descriptive text is added either to the top of the results file or above the results that are appended to the bottom of the results file.

Details

The DMSRVPROFILEJOB function generates a profile from a DataFlux Data Management Server repository.

Example: DMSRVPROFILEJOB Function

The following example generates a profile from the specified repository.

```
data _null_;
    jobid = dmsrvProfileJob('myfolder/prof_job',
       'http://myhost.unx.com', 21036, 1);
run;
```

See Also

- "DMSRVJOBSTATUS Function" on page 100
- "DMSRVKILLJOB Function" on page 102

DMSRVUSER Function

Registers a user on a DataFlux Data Management Server.

Valid in:	SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.
Restriction:	Not valid in SAS Viya CAS.

Syntax

DMSRVUSER(*user-ID*, *password*)

Required Arguments

user-ID
identifies a user-ID according to the registry in a DataFlux Data Management Server.

password
identifies the associated user-ID user according to the registry in the DataFlux Data Management Server. The password can be plain text or encoded in SAS.

Details

The DMSRVUSER function registers a user on a secure DataFlux Data Management Server. A return value of zero indicates storage of credentials was successful. A return value of 1 indicates a failure to store the credentials.

Call this function as needed in a single DATA step to access different Data Management Servers or to change the registered user credentials within a single Data Management Server.

If security has not been configured on a DataFlux Data Management Server, then the DMSRVUSER function has no effect.

Return values are 0 (successful registration of credentials) or 1 (failed to register credentials).

Example: DMSRVUSER Function

The following example supplies a user identifier and a password to a secure DataFlux Data Management Server:

```
rc= dmsrvUser('dfUser3','pwdUser3');
```

DMSRVVER Function

Returns the version of the DataFlux Data Management Server.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DMSRVVER(*host*, *port*)

Required Arguments

host

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **`https`** instead of **`http`** (for example, https://myhost.unx.com).

Interaction If a zero-length string is entered for the *host* argument, then the value *localhost* is used.

Example

```
/* Localhost is used for the host */
   dmsrvVer('', 21036);
```

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

```
/* Port 21036 or 21037 is used */
   dmsrvVer('http://myhost.unx.com', 0);
```

Details

The DMSRVVER function takes two arguments, a host name and a port number. If *host* is not specified, the local host is used. If *port* is not specified, or if the value is zero or a negative number, the default port number 21036 is used.

DMSRVVER returns a string listing the version number of the integration server, designated by the host and port values.

Example: DMSRVVER Function

The following example sets the value of the version to the character string of the DataFlux Data Management Server, running on machine 'myhost' and communicating with port 19525.

```
version=dmsrvVer ('http://myhost.unx.com', 19525);
```

See Also

"DMSRVBATCHJOB Function" on page 95

DQCASE Function

Returns a character value with standardized capitalization.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQCASE(*character-value*, '*case-definition*' <, '*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that resolves to a constant or variable.

case-definition
specifies the case definition that is referenced to generate the return value. The definition must be available in the specified *locale*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The DQCASE function operates on any character content, such as names, organizations, and addresses.

If the value of *character-value* is represented by a specific case definition, then the use of that definition is recommended, rather than using a generic definition. For example, if the value of *character-value* is a street address, and if and you are using the ENUSA locale, then the recommended case definition is PROPER–ADDRESS. This definition is recommended instead of the generic case definition PROPER.

Example: DQCASE Function

The following example standardizes the 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 of ORGNAME is **`Bill's Plumbing & Heating`**.

DQEXTINFOGET Function

Returns the names of the tokens that are supported by an extraction definition.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQEXTINFOGET('*extraction-definition*' <,'*locale*'>)

Required Argument

extraction-definition
specifies the name of the extraction definition, which is referenced to generate the return value. The definition must be supported by the *locale*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The DQEXTINFOGET function returns the names of the tokens that are supported by the *extraction-definition*. The names of the tokens are used as arguments in other extraction functions, such as DQEXTRACT.

Example: DQEXTINFOGET Function

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

```
tokenNames=dqExtInfoGet('Contact Info','ENUSA');
put tokenNames;
```

After this function call, the value of TOKENNAMES is **`Name, Organization, Address, E-mail, Phone, Additional Info`**.

See Also

- "DQEXTTOKENGET Function" on page 110
- "DQEXTTOKENPUT Function" on page 111

DQEXTRACT Function

Returns a delimited string of extraction token values from an input character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQEXTRACT('*character-value*', '*extraction-definition*' <,'*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

extraction-definition
specifies the name of the extraction definition, which is referenced to generate the return value. The definition must be supported in the *locale*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The return value is a delimited string of extraction token values. The tokens are named segments of the input value that are detected by the *extraction-definition*.

To learn the names of tokens that are supported by an extraction definition, use the DQEXTINFOGET function.

To return an extraction token value from the delimited string, use the DQEXTTOKENGET function.

Example

The following example extracts the name of an individual. The DQEXTTOKENGET function then returns the values of two of the tokens.

```
extValue=dqExtract('mr. james joseph westly james.westly@sas.com',
    'Contact Info', 'ENUSA');
name=dqExtTokenGet(extValue, 'Name', 'Contact Info, 'ENUSA');
email =dqExtTokenGet(extValue, 'E-mail', 'Contact Info', 'ENUSA');
```

After these function calls, the value of **name** is **Mr. James Joseph Westly** and the value of **email** is **james.westly@sas.com**.

See Also

"DQEXTTOKENPUT Function" on page 111

DQEXTTOKENGET Function

Returns an extraction token value from a delimited string of extraction token values.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQEXTTOKENGET(*delimited-string*, '*token*', '*extraction-definition*' <,'*locale*'>)

Required Arguments

delimited-string
specifies the input value, which must be a delimited string of extraction token values. The input value is required to have been generated with the *extraction-definition*. The argument can be specified as a variable or an expression that evaluates to a variable.

token
specifies the name of the extraction token, the value of which is returned by the function. The token must be supported by the *extraction-definition*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

extraction-definition
specifies the extraction definition that is referenced to generate the return value. The definition must be supported by the *locale*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale

specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The *token* is a named component of a character value that is recognized by the *extraction-definition*. The value of the token is the text that is recognized and extracted from a character value. The delimited string of extraction token values is a list of all of the token values that are extracted from a character value. The delimited string is generated by the DQEXTRACT function. The DQEXTTOKENGET function returns one extraction token value from a delimited string.

To determine the names of the tokens that are supported by a given extraction definition, use the DQEXTINFOGET function.

Example

The following example extracts the name of an individual. The DQEXTTOKENGET function then returns the values of two of the tokens.

```
extValue=dqExtract('mr. james joseph westly james.westly@sas.com',
    'Contact Info', 'ENUSA');
name=dqExtTokenGet(extValue, 'Name', 'Contact Info, 'ENUSA');
email =dqExtTokenGet(extValue, 'E-mail', 'Contact Info', 'ENUSA');
```

After these function calls, the value of **`name`** is **`Mr. James Joseph Westly`** and the value of **`email`** is **`james.westly@sas.com`**.

See Also

"DQEXTTOKENPUT Function" on page 111

DQEXTTOKENPUT Function

Inserts an extraction token value into a delimited string of extraction token values and returns the updated delimited string.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQEXTTOKENPUT('*delimited-string*', '*token-value*', '*token-name*', '*extraction-definition*' <,'*locale*'>)

Required Arguments

delimited-string
specifies the input value, which must be a delimited string of extraction token values. The input value is required to have been generated with the *extraction-definition*. The argument can be specified as a variable or an expression that evaluates to a variable.

token-value
specifies the value of the token that is to be inserted into the *delimited-string*.

token-name
specifies the name of the token that receives the inserted *token-value*. The token must be supported by the *extraction-definition*.

extraction-definition
specifies the name of the extraction definition that is referenced to generate the return value. The specified definition must be the same definition that was used to generate the *delimited-string*. The argument can be specified as a constant, variable, or an expression that evaluates to a constant or variable. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

If a value exists for the *token* in the input delimited string, then the returned delimited string includes the new value before the existing value. The existing value is retained.

You can assign the return value from DQEXTTOKENPUT to the same variable that is specified as the input delimited string.

Example

The following example updates a delimited string of extraction token values. The updated delimited string adds a new value to the existing value of the **Name** token.

```
extValue=dqExtract('james joseph westly james.westly@sas.com',
    'Contact Info', 'ENUSA');
nameAdd=dqExtTokenPut(extValue, 'jim', 'Name', 'Contact Info, 'ENUSA');
newName=dqExtTokenGet(nameAdd, 'Name', 'Contact Info', 'ENUSA');
```

After these function calls, the value of the **newName** variable is **Jim James Joseph Westly**.

See Also

- “DQEXTRACT Function” on page 109
- “DQEXTINFOGET Function” on page 108

DQGENDER Function

Returns a gender determination from the name of an individual.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Arguments

character-value
specifies the input value as a character constant, variable, or expression that evaluates to a constant or variable.

gender-analysis-definition
specifies the gender analysis definition, which is referenced to generate the return value. The definition must be supported in the *locale*. The argument can be specified as a constant, variable, or expression that evaluates to a constant or an expression.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

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.

Example: DQGENDER Function

The following example returns the value M for the variable **gender**.

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

See Also

"DQGENDERPARSED Function" on page 115

DQGENDERINFOGET Function

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

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Argument

gender-analysis-definition
specifies the gender analysis definition of interest. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Example: DQGENDERINFOGET Function

The following example writes the parse definition that is associated with GENDER to the SAS log. The parse definition that is returned is then used to display the names of the tokens that are supported by that parse definition. The tokens are then used to construct a parsed value and to write the results of the gender to the log.

```
/* 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

- "DQGENDER Function" on page 113
- "DQGENDERPARSED Function" on page 115
- "DQPARSE Function" on page 128
- "DQPARSETOKENPUT Function" on page 135

DQGENDERPARSED Function

Returns the gender of an individual from a delimited string of parse token values.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQGENDERPARSED(*delimited-string*, '*gender-analysis-definition*' <,'*locale*'>)

Required Arguments

delimited-string
specifies the input value, which must be a delimited string of parse token values.The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

gender-analysis-definition
specifies the name of the gender analysis definition that is referenced to generate the return value. The definition must be supported by the *locale*. The argument can be specified as a constant, a variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, character variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

If the gender 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 114

- "DQPARSE Function" on page 128
- "DQPARSETOKENPUT Function" on page 135

DQIDENTIFY Function

Returns the highest-scoring identity for a character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQIDENTIFY(*character-value*, *'identification-analysis-definition'* <,'*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or a variable.

identification-analysis-definition
specifies the name of the identification analysis definition that is referenced to determine the return value. The definition must be supported by the *locale*. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, then the default locale is used.

Details

Use the DQIDENTIFY function to return the name of the highest-scoring identity that is supported by the given identification analysis definition.

To list the names of the identities that are supported by a given identification definition, use the DQIDENTIFYINFOGET function.

Example: DQIDENTIFY Function

The following example determines the type of information that is conveyed by a character value.

```
dqid=dqIdentify('Puerto Rico','Field Content','ENUSA');
```

After this function call, the value of **dqid** is **STATE/PROVINCE**.

DQIDENTIFYIDGET Function

Returns an identification analysis score for a given identity from a delimited string of identification analysis scores.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, and SAS Viya DS2 Compute Server.

Restriction: Not valid in SAS 9.

Syntax

DQIDENTIFYIDGET(*delimited-string*, '*identity-name*', '*identification-analysis-definition*' <,'*locale*'>)

Required Arguments

delimited-string
specifies the input value, which must be a delimited string of identification analysis scores. The delimited string is required to have been generated by the *identification-analysis-definition*. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

identity-name
specifies the name of the identity, the value of which is returned from the *delimited-string*. The value of the identity is an identification analysis score. The identity must be supported by the *identity-analysis-definition*.

identification-analysis-definition
specifies the name of the identification analysis definition that is referenced to generate the return value. The definition must be supported by the *locale*. The definition must be the same definition that was used to generate the *delimited-string*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The DQIDENTIFYIDGET function returns a numeric confidence score for a specified identity. The available identities are determined by the specified *identification-analysis-definition*. The numeric confidence score is returned from a *delimited-string* of scores that were generated by the DQIDENTIFYMULTI function.

To learn the names of the identities that are supported by a given identification analysis definition, use the DQIDENTIFYINFOGET function.

Example: DQIDENTIFYIDGET Function

The following example generates a delimited string of identification analysis scores and returns several scores from the delimited string:

```
idscores=dqIdentifyMulti('Puerto Rico', 'Field Content');
spscore=dqIdentifyIdGet(idScores, 'STATE/PROVINCE', 'Field Content');
cscore=dqIdentifyIdGet(idScores, 'COUNTRY', 'Field Content');
iscore=dqIdentifyIdGet(idScores, 'INDIVIDUAL', 'Field Content');
```

The resulting identification analysis scores are as follows:

spscore = 76 iscore = 57
cscore = 67

DQIDENTIFYINFOGET Function

Returns the names of the identities that are supported by a given identification analysis definition.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, and SAS Viya DS2 Compute Server.

Restriction: Not valid in SAS 9.

Syntax

DQIDENTIFYINFOGET('*identification-analysis-definition*' <,'*locale*'>)

Required Argument

identification-analysis-definition
specifies the name of the identification analysis definition that is referenced to generate the return value. The definition must be supported by the *locale*. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The DQIDENTIFYINFOGET function returns a list of identity names in the following format:

```
identity1,identity2,...identityN
```

Example: DQIDENTIFYINFOGET Function

The following example returns a list of identity names for the identification analysis definition **Field Content**.

```
dqidnames=dqidentifyInfoGet('Field Content','ENUSA');
```

After this function call, the value of **dqidnames** is as follows:

```
IBAN,PAYMENT CARD NUMBER,NETWORK ADDRESS,E-MAIL,
URL,DATE,DATE/TIME,CURRENCY,COUNTRY,GEOGRAPHICAL POINT,
INDIVIDUAL,ORGANIZATION,FULL ADDRESS,DELIVERY ADDRESS,
CITY,CITY-STATE/PROVINCE-POSTAL CODE,STATE/PROVINCE,
POSTAL CODE,PHONE,SOCIAL SECURITY NUMBER,UNKNOWN,BLANK
```

See Also

- "DQIDENTIFY Function" on page 116
- "DQIDENTIFYIDGET Function" on page 117
- "DQIDENTIFYMULTI Function" on page 119

DQIDENTIFYMULTI Function

Returns all of the identification analysis scores of a character value.

Valid in:	SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, and SAS Viya DS2 Compute Server.
Restriction:	Not valid in SAS 9.

Syntax

DQIDENTIFYMULTI(*character-value*, '*identification-analysis-definition*', <'*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

identification-analysis-definition
specifies the identification analysis definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The returned value is a delimited list of scores for each identity that is supported by the *identification-analysis-definition*.

Example: DQIDENTIFYMULTI Function

The following example generates a delimited string of identification analysis scores and returns several scores from the delimited string:

```
idscores=dqIdentifyMulti('Puerto Rico', 'Field Content');
spscore=dqIdentifyIdGet(idScores, 'STATE/PROVINCE', 'Field Content');
cscore=dqIdentifyIdGet(idScores, 'COUNTRY', 'Field Content');
iscore=dqIdentifyIdGet(idScores, 'INDIVIDUAL', 'Field Content');
```

The resulting identity scores are as follows:

spscore = 76 iscore = 57
cscore = 67

See Also

"DQIDENTIFYIDGET Function" on page 117

DQLOCALEGUESS Function

Returns the ISO code of the highest-scoring locale for a character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQLOCALEGUESS(*character-value*, '*locale-guess-definition*')

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

locale-guess-definition
specifies the definition that is referenced to generate the return value. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

Details

The DQLOCALEGUESS function applies the input character value to the *locale-guess-definition* in each of the locales that are listed in the DQLOCALE= system option or session option. Each locale receives a confidence score. The DQLOCALEGUESS function returns the ISO code of the locale that receives the highest confidence score.

Note: At least one locale guess definition must be supported by each of the locales that are listed in the DQLOCALE= system option. If a locale does not support a locale guess definition, then the DQLOCALEGUESS function fails to execute.

Two or more locales can receive the highest confidence score. When this occurs, the return value is the highest-scoring locale that appears last in the value of DQLOCALE=.

If all of the locales receive a confidence score of zero, then the DQLOCALEGUESS function returns a value that depends on a setting in the locale guess definition. The setting is **`Select the last locale given if no score can be computed`**. The DQLOCALEGUESS function looks for the setting in each locale, starting with the last locale in DQLOCALE= and working forward through the list. The first locale that contains a locale guess definition that enables the setting becomes the name that is returned by the function. If none of the locale guess definitions enable the setting, then the return value of the function is indeterminate.

Examples:

Example 1: DQLOCALEGUESS Function

The following example returns the name of a locale as the value of the LOCALE variable.

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

Example 2: Multiple Highest Confidence Scores

Assume that the confidence scores in the following example are ENUSA=750, ENGBR=750, and ENHGK=250. The value of LOCALE is ENGBR.

```
options dqlocale=(ENUSA ENGBR ENHKG);
data _null_;
locale = dqLocaleGuess(input, 'Country');
run;
```

Example 3: All Locales Have Zero Confidence Scores

In the following example, assume that the setting **`Select the last locale given if no score can be computed`** is applied to the locale guess definition in the ENUSA and ENGBR locales. Also, assume that the value is not set in the locale guess definition in the ENHGK locale. The value of LOCALE is ENGBR.

```
options dqlocale=(ENUSA ENGBR ENHKG);
data _null_;
locale = dqLocaleGuess(input, 'Country');
run;
```

See Also

“DQLOCALEINFOGET Function” on page 122

DQLOCALEINFOGET Function

Returns a list of the locales that are loaded into memory.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Optional Argument

info-type
the value that is analyzed to determine the locales that are currently loaded into memory. If no parameter is specified, the default LOADED is used. The only valid value is LOADED.

Details

The DQLOCALEINFOGET function returns a comma-delimited list of ISO code. These are the locales that are currently loaded into memory. The first locale is the default locale that is used by data quality operations.

Example: DQLOCALEINFOGET Function

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

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

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

See Also

- “DQLOCALEINFOLIST Function” on page 122
- “%DQPUTLOC AUTOCALL Autocall Macro” on page 84

DQLOCALEINFOLIST Function

Returns a count of definitions and displays the names of definitions for a type of definition in a locale.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Arguments

definition-type
specifies the type of definition that is displayed and counted. The type must exist in the specified locale. Valid definition types include the following:

ALL	IDENTIFICATION
CASE	MATCH
EXTRACTION	PARSE
GENDER	PATTERN
GUESS	STANDARDIZATION

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

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

Example: DQLOCALEINFOLIST Function

The following example writes a list of the definition names and count in the first locale in the locale list to the SAS log.

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

The following example writes a list of parse definitions in the DEDEU locale to the SAS log.

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

See Also

"DQLOCALEINFOGET Function" on page 122

DQLOCALESCORE Function

Returns a locale confidence score for an input character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQLOCALESCORE('*character-value*', '*locale-guess-definition*', '*locale*')

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

locale-guess-definition
specifies the locale guess definition that is referenced to generate the return value. The argument can be specified as a constant, variable, or expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default The default locale is the first locale in the locale list.

Details

The return value is an integer that represents a confidence score. A higher confidence score indicates a greater likelihood that the source string originates in the specified locale.

Return values range from 0 to 1000.

Example: DQLOCALESCORE Function

The following example returns a confidence score for an address and a locale. The locale applies to the English language, as that language is used in the United States of America.

```
data test;
length score 8.;
score = dqLocaleScore('123 Weatherly Lane', 'Address', 'ENUSA');
put 'Confidence score for ENUSA: ' score;
run;
```

See Also

"DQLOCALEGUESS Function" on page 120

DQMATCH Function

Returns a matchcode from a character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

match-definition
specifies the name of the match definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Arguments

sensitivity
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 matchcode. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster.

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

For higher levels of *sensitivity*, two values must be very similar to produce the same matchcodes. At lower sensitivities, two values can produce the same matchcodes when they are less similar.

Example: DQMATCH Function

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

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

See Also

Chapter 8, “DQMATCH Procedure,” on page 47

DQMATCHINFOGET Function

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

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQMATCHINFOGET('*match-definition*' <,'*locale*'>)

Required Argument

match-definition
specifies the name of the match definition. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

Obtaining the name of a parse definition enables you to create delimited strings of parse token values with the DQPARSE or DQPARSETOKENPUT functions.

If the specified match definition does not have an associated parse definition, then the DQMATCHINFOGET function returns a zero-length value. The returned missing value indicates that the function has run successfully but has not found an associated parse definition.

Example: DQMATCHINFOGET Function

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

```
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 matchcode from a delimited string of parse token values.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQMATCHPARSED(*delimited-string*, '*match-definition*' <,*sensitivity*> <,'*locale*'>)

Required Arguments

delimited-string
specifies the input variable, which must have a value that is a delimited string of parse token values. The delimited string must have been generated with a parse definition that is associated with the *match-definition* in the *locale*.

match-definition
specifies the name of the match definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Arguments

sensitivity
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 matchcode. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster.

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Example: DQMATCHPARSED Function

The following example returns a matchcode for the parsed name of an individual. The amount of information in the matchcode is 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

- “DQMATCHINFOGET Function” on page 125
- “DQPARSE Function” on page 128
- “DQPARSETOKENPUT Function” on page 135

DQOPTSURFACE Function

Reveals or hides non-surfaced definitions.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DQOPTSURFACE('*surface-flag*')

Required Argument

surface-flag

specifies the policy for non-surfaced definitions. Valid values are as follows:

YES reveals all non-surfaced definitions.

NO hides all non-surfaced definitions.

Default NO

Details

Definitions that are not surfaced cannot be displayed or referenced by data quality language elements. By default, non-surfaced definitions are hidden.

The DQOPTSURFACE function returns the previous value of the surface definition policy.

To learn more about the surfaced and non-surfaced designations for QKB definitions, see the configuration option SURFACEALL in the *DataFlux Data Management Studio: Installation and Configuration Guide.*

Example: DQOPTSURFACE Function

The following example specifies that non-surfaced definitions are revealed. The character value `oldDEFAULT` contains the value of the previous setting.

```
oldDefault=DQOPTSURFACE('YES');
```

DQPARSE Function

Returns a delimited string of parse token values.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQPARSE('*character-value*', '*parse-definition*' <,'*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

parse-definition
specifies the name of the parse definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The return value is a delimited string of parse token values. The parse tokens are the segments of the input value that are detected by the *parse-definition*. To see the names of the tokens that are detected by the *parse-definition*, use the DQPARSEINFOGET function.

To return a parse token value from a delimited string, use the DQPARSETOKENGET function. To return a parse token value from a character value directly, without generating a delimited string, use the DQTOKEN function.

Example: DQPARSE Function

The following example parses the name of an individual and returns the values of two 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 PREFIX is Mrs. and the value of GIVEN is Sallie.

See Also

- “DQPARSETOKENGET Function” on page 134
- “DQPARSETOKENPUT Function” on page 135
- “CALL DQPARSE Routine” on page 130

CALL DQPARSE Routine

Returns a delimited string of parse token values and a status flag.

Valid in: SAS Viya CAS, SAS Viya Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

CALL DQPARSE(*character-value*, *parse-definition*, '*parse-result*, *parse-return-code* <,'*locale*'>);

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

parse-definition
specifies the name of the parse definition that is referenced to create the return value. The definition must be supported by the *locale*.

parse-result
specifies a variable or an expression that evaluates to a variable. The variable receives the delimited string of parse token values.

parse-return-code
specifies a variable or an expression that evaluates to a variable. The value of the variable is a status flag. The value of the variable is 1 when the parse operation is successful. Otherwise, the value is 0.

Optional Argument

locale
specifies a constant, variable, or expression that resolves to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The return values are a delimited string of parse token values and a status flag. The parse tokens are the named segments of the input value that are detected by the *parse-definition*. To see the names of the tokens that are detected by the *parse-definition*, use the DQPARSEINFOGET function.

To return the value of a parse token from a delimited string, use the DQPARSETOKENGET function. To return a parse token value from a character value directly, without generating a delimited string, use the DQTOKEN function.

Example: DQPARSE CALL Routine

The following example parses the name of an individual.

```
data a;
    length parsename $ 40;
    call dqparse (name, 'Name', parsename, solution);
    if solution= 1 then
      put 'found solution';
    else
      put 'no solution';
run;
```

See Also

- "DQPARSETOKENGET Function" on page 134
- "DQPARSETOKENPUT Function" on page 135

DQPARSEINFOGET Function

Returns the names of the tokens that are supported by a parse definition.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQPARSEINFOGET('*parse-definition*' <,'*locale*'>)

Required Argument

parse-definition
specifies the name of the parse definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that resolves to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Example: DQPARSEINFOGET Function

The following example returns the token names for the parse definition email in the locale ENUSA, and it displays the token names in the SAS log.

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

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

See Also

- "DQPARSETOKENGET Function" on page 134
- "DQPARSETOKENPUT Function" on page 135

DQPARSEINPUTLEN Function

Overrides the default expected length of parsed input and returns a string indicating its previous value.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DQPARSEINPUTLEN('*input-length*')

Required Argument

input-length
specifies an expected input length. Valid values are as follows:

SHORT	AUTO
LONG	REMOVE

Details

If REMOVE is specified, the expected input length is set to the default value in the *parse-definition*.

The DQPARSEINPUTLEN function returns a value that indicates the previous value of the input length. If the value NOTSET is returned, then the previous input length was not set.

Example: DQPARSEINPUTLEN Function

The following example sets the default input length to SHORT. The previous value of the parse input length is returned as the value of `oldDEFAULT`.

```
oldDefault= dqParseInfPutLen('short');
```

DQPARSERESLIMIT Function

Overrides the default expected limit on resources consumed during parsing.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DQPARSERESLIMIT('*resource-limit*')

Required Argument

resource-limit
specifies a resource limit for parsing operations. Valid values include the following:

VERYLOW	VERYHIGH
LOW	INTENSIVE
MEDIUM	REMOVE
HIGH	

Details

If REMOVE is specified, the resource limit is set to the default value that is specified in the *parse-definition*.

The DQPARSERESLIMIT function returns a value that indicates the previous value of the resource limit. If the value NOTSET is returned, then the resource limit was not previously set.

Example: DQPARSERESLIMIT Function

The following example sets the default resource limit to INTENSIVE. The value of `oldDEFAULT` is the previous value of the resource limit.

```
oldDefault=DQPARSERESLIMIT('intensive');
```

DQPARSESCORDEPTH Function

Overrides the default value that determines how deeply to search for the best parsing score.

Valid in: SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.

Restriction: Not valid in SAS Viya CAS.

Syntax

DQPARSESCORDEPTH(*level*)

Required Argument

level
an integer that specifies the maximum depth permitted during scoring.

Range 5-10, and 0

Details

The DQPARSESCORDEPTH function changes the default search level is used to find the best parsing score. The default parse score depth is specified in the parse definition. If 0 is specified, then the search level is set to the default value that is specified in the parse definition.

Example: DQPARSESCORDEPTH Function

The following example sets DQPARSESCORDEPTH to eight. The numeric variable `oldDEFAULT` contains the scoring depth previously in force.

```
oldDefault=DQPARSESCOREDEPTH(8);
```

DQPARSETOKENGET Function

Returns a parse token value from a delimited string of parse token values.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQPARSETOKENGET(*delimited-string*, '*token*', '*parse-definition*', <,'*locale*'>)

Required Arguments

delimited-string
specifies the input string as a variable or an expression that evaluates to a variable. The value of the variable must be a delimited string of parse token values. The delimited string is required to have been generated by the *parse-definition*.

token
specifies the name of a token, the value of which is returned. The token must be supported by the *parse-definition*.

parse-definition
specifies the parse definition that was referenced to generate the *delimited-string*. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

To determine the names of the tokens that are supported by a given parse definition, use the DQPARSEINFOGET function.

Example: DQPARSETOKENGET Function

The following example generates a delimited string of parsed token values and 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 `prefix` is **`Mrs.`** and the value of `given` is **`Sallie`**.

See Also

- "DQPARSEINFOGET Function" on page 131
- "DQPARSE Function" on page 128
- "CALL DQPARSE Routine" on page 130

DQPARSETOKENPUT Function

Inserts a parse token value into a delimited string of parse token values and returns the updated delimited string.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQPARSETOKENPUT('*delimited-string*', '*token-value*', '*token-name*', '*parse-definition*'
<,'*locale*'>)

Required Arguments

delimited-string
specifies the input string as a variable or an expression that evaluates to a variable. The value of the variable must be a delimited string of parse token values. The delimited string is required to have been generated by the *parse-definition*.

token-value
specifies the value of the token that is to be inserted into the *delimited-string*.

token-name
specifies the name of the token that receives the new value. The token must be supported by the *parse-definition.*

parse-definition
specifies the name of the parse definition that is referenced to generate the return value. The parse definition must be supported by the *locale*. The parse definition must be the same definition that was used to generate the *delimited-string*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

In the input *delimited-string*, if a value exists for the *token-name*, then the new value is inserted before the existing value. The existing value is retained.

> **TIP** If you specify a variable name for the value of the *delimited-string*, you can assign that same variable to receive the return value from the DQPARSETOKENPUT function.

See Also

"DQPARSEINFOGET Function" on page 131

DQPATTERN Function

Returns a pattern analysis from a character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQPATTERN('*character-value*' , '*pattern-analysis-definition*' <,'*locale*'>)

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or a variable.

pattern-analysis-definition
specifies he name of the definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Argument

locale
specifies a constant, variable, or an expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

DQPATTERN 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, as follows:

*
: non-alphanumeric, such as punctuation marks or symbols

A
: alphabetic

M
: mixture of alphabetic, numeric, and non-alphanumeric

N
: numeric

Example: DQPATTERN Function

The following example analyzes the words in the input character value. The results are written to the SAS log using the PUT statement.

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

The DQPATTERN function returns A N* M. Using the CHARACTER pattern analysis definition returns AAAAAAA N* NNAA.

DQSCHEMEAPPLY Function

Applies a scheme and returns a transformed character value.

Valid in:	SAS Viya Compute Server, SAS 9, PROC SQL, and SAS Component Language.
Restriction:	Not valid in SAS Viya CAS.
Requirement:	Schemes using SAS format are required in the z/OS operating environment.

Syntax

DQSCHEMEAPPLY('*character-value*', '*scheme*', '*scheme-format*' <,'*mode*'> <,'*scheme-lookup-method*'> <,'*match-definition*'> <,'*sensitivity*'> <,'*locale*'>)

Required Arguments

character-value
: specifies a constant, variable, or expression that evaluates to the input value.

scheme
: identifies the scheme that is applied to the input value. To apply a SAS format scheme, the *scheme* argument includes both the path and the filename of the SAS data set, in quotation marks. To apply a QKB-format scheme, 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 file specification that includes both the path and the filename that ends in **.sch.qkb**.

Requirement Lowercase characters are required.

Note In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

scheme-format

identifies the format of the scheme. The valid values for this argument are defined as follows:

QKB

indicates that the scheme is stored in QKB scheme file format. This is the default value.

NOQKB

indicates that the scheme is stored in SAS format.

See "Creating Schemes" on page 71

Optional Arguments

mode

specifies how the scheme is to be applied to the input character value. If the value of *scheme-lookup-method* is USE_MATCHDEF, and a value is not specified for *mode*, then the default value of mode (PHRASE) is used. Valid values for *mode* are as follows:

PHRASE

compares the entire input character value to the entire length of each of the DATA values in the scheme. When the value of the scheme-lookup-method is USE_MATCHDEF, the matchcode values of the entire input value are compared to the matchcodes of DATA values in the scheme. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

ELEMENT

compares each element in the *character-value* to each of the DATA values in the scheme. When the value of the *scheme-lookup-method* is USE_MATCHDEF, the matchcode of the entire *character-value* is compared to the matchcodes of the scheme's DATA values. A transformation occurs when a match is found between an element in the *character-value* and a DATA value in the scheme.

Default The mode that is stored in the scheme. If a mode is not stored in the scheme, then the default value of PHRASE is used.

scheme-lookup-method

specifies the method that is used to apply the scheme. Valid values include the following:

EXACT

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

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

USE_MATCHDEF
specifies that the matchcode of the *character-value* is to be compared to the matchcode of the DATA values in the scheme. A transformation occurs when the matchcodes are identical.

Specify USE_MATCHDEF to enable *locale*, *match-definition*, and *sensitivity*.

Default EXACT

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

match-definition
the name of the match definition that is referenced to create the matchcode for the *character-value*. The match definition must be supported by the *locale*. If USE_MATCHDEF is specified, and if a match definition is not stored in the scheme, then a value is required for the *match-definition* argument.

Default If USE_MATCHDEF is specified, and if the *match-definition* argument is not specified, then the default match definition is the one that is stored in the scheme.

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

See "Meta Options" on page 72

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

Default If USE_MATCHDEF is specified, and if the sensitivity argument is not specified, then the default sensitivity is the sensitivity value that is stored in the scheme. When USE_MATCHDEF is specified and a sensitivity value is not stored in the scheme, the default sensitivity value is 85.

Range 50 to 95

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

See "Meta Options" on page 72

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

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

Details

The DQSCHEMEAPPLY function transforms the *character-value* by applying a scheme. The scheme can be in SAS format or QKB format. To create schemes in SAS format, use the DQSCHEME procedure. To create schemes in QKB format, use the DQSCHEME procedure or use the DataFlux Data Management Studio software.

Example: DQSCHEMEAPPLY Function

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

```
/* Create the input data set. */P
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.qkb';

/* Create the scheme. */
proc dqscheme data=suppliers qkb;
  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','qkb','phrase','EXACT');
   put 'Before applying the scheme: ' company /
       'After applying the scheme:  ' outCompany;
run;
```

See Also

Chapter 9, “DQSCHEME Procedure,” on page 63

CALL DQSCHEMEAPPLY Routine

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

Valid in:	SAS Viya Compute Server, SAS 9, and SAS Component Language.
Restriction:	Not valid in SAS Viya CAS.
Requirement:	Schemes using SAS format are required in the z/OS operating environment.

Syntax

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

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

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

scheme
the scheme that is applied to the input value. To specify a SAS format scheme, provide a filename specification that includes a pathname and the SAS data set name enclosed in quotation marks. To specify a QKB-format scheme, provide the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a file specification that includes both the pathname and a filename that ends in .sch.qkb.

Requirement	Lowercase letters.
Note	In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

scheme-format

identifies the format of the scheme. The valid values for this argument are defined as follows:

QKB

indicates that the scheme is stored in QKB scheme file format. This is the default value.

NOQKB

indicates that the scheme is stored in SAS format.

Default QKB

See "Creating Schemes" on page 71

Optional Arguments

mode

specifies how the scheme is to be applied to the *character-value*. The default value is the mode that is stored in the scheme. If a mode is not stored in the scheme, then the default value of *mode* is PHRASE.

If the value of *scheme-lookup-method* is USE_MATCHDEF, and if a value is not specified for *mode*, then the default value of *mode* (PHRASE) is used.

Valid values for *mode* are defined as follows:

PHRASE

compares the entire *character-value* to the entire length of each of the DATA values in the scheme. When the value of the *scheme-lookup-method* is USE_MATCHDEF, the matchcode of the entire *character-value* is compared to the matchcodes of DATA values in the scheme. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

ELEMENT

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

transform-count-variable

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

If the *character-value* is transformed, then the value is a positive integer that represents the number of elements in the input value that are transformed.

Interactions If the *mode* is PHRASE or ELEMENT, and if the *character-value* is not transformed, then the value of the *transform-count-variable* is 0.

If the *character-value* is transformed, then the value of *transform-count-variable* is 1.

The transformation count might appear to be inaccurate if the transformation value in the scheme is the same as the *character-value*, or the same as any element in the *character-value*.

scheme-lookup-method

specifies the method that is used to apply the scheme. Valid values are as follows:

EXACT

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

IGNORE_CASE

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

USE_MATCHDEF

specifies that the matchcode of the *character-value* is to be compared to the matchcode of the DATA values in the scheme. A transformation occurs when the two matchcodes are identical.

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

match-definition

the name of the match definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Interactions

If USE_MATCHDEF is specified and if *match-definition* is not specified, then the default match definition is the definition that is stored in the scheme.

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

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

sensitivity

specifies the amount of information in the matchcode that is created for the *character-value*. With higher sensitivity values, two values must be increasingly similar to generate the same matchcode. At lower sensitivity values, two values can receive the same matchcode despite their dissimilarities.

Default

85

Range

50 to 95

Interactions

Sensitivity is valid only when the value of the scheme-lookup-method is USE_MATCHDEF.

If a sensitivity value is not provided in the scheme file, and if USE_MATCHDEF is specified, then the *sensitivity* value that is stored in the scheme is used.

locale

specifies a constant, variable, or an expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

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

Details

The DQSCHEMEAPPLY CALL routine transforms an input value by applying a scheme. The *scheme* can be in SAS format or QKB format. To create schemes in SAS format, use the DQSCHEME procedure. To create schemes in QKB format, use the DQSCHEME procedure or use the DataFlux Data Management Studio software.

Example: DQSCHEMEAPPLY CALL Routine

The following example generates a scheme using the QKB scheme file format with the DQSCHEME procedure and then applies that scheme to a data set with the DQSCHEMEAPPLY CALL routine. The example assumes that 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 noqkb;
  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','noqkb',
       '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. The value of NUMTRANS is 1 if the organization name is transformed.

In the following example, a transformation count of 1 is shown when no transformation appears to have been made. This is shown in the 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. In these cases, the transformation value is the same as the input value.

See Also

Chapter 9, “DQSCHEME Procedure,” on page 63

DQSTANDARDIZE Function

Standardizes the casing, spacing, and format of certain words and abbreviations and returns an updated character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or a variable.

standardization-definition
specifies the definition that is referenced to generate the return value. The definition must be supported by the *locale*.

Optional Argument

locale

specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

The returned character value is standardized in the appropriate case, spelling, and format, with insignificant blank spaces and punctuation removed. The order of the elements in the return value can differ from the order of the elements in the input character value.

Example: DQSTANDARDIZE Function

The following example standardizes four names using the Name standardization definition in the ENUSA locale. The following example assumes that the ENUSA locale has been loaded into memory.

```
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 the value of a token from an input character value.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

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

Required Arguments

character-value
specifies the input value as a constant, variable, or expression that evaluates to a constant or variable.

token
specifies the token that is supported by the *parse-definition*.

parse-definition
specifies the parse definition that is referenced to generate the return value. The definition must be supported in the *locale*.

Optional Argument

locale
specifies a constant, variable, or expression that evaluates to the ISO code of the locale.

Default If no value is specified, the default locale is used.

Details

Use the DQTOKEN function to return the value of one token. To return the value of more than one token, use the functions DQPARSE and DQPARSETOKENGET.

If the DQTOKEN function does not find a value for the *token*, the return value for that token is blank.

Example: DQTOKEN Function

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

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

After the DQTOKEN call, the value for the PREFIX variable is `Mrs`.

DQVER Function

Returns the version of the SAS Data Quality engine.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQVER()

Details

The DQVER function takes no arguments and returns the version number of the SAS Data Quality engine.

Example: DQVER Function

The following example returns the SAS Data Quality engine version.

```
version=DQVER ();
```

DQVERQKB Function

Returns the version of the currently loaded QKB.

Valid in: SAS Viya CAS, SAS Viya Compute Server, SAS Viya DS2 CAS, SAS Viya DS2 Compute Server, and SAS 9. Also valid in PROC SQL and SAS Component Language.

Syntax

DQVERQKB ()

Details

The DQVERQKB function takes no arguments and returns a five-character string that contains the version of the currently loaded QKB. If the version cannot be determined (as with QKB versions before 2005A), the value UNKNW is returned.

Example: DQVERQKB Function

The following example returns the version of the currently loaded QKB.

```
version= DQVERQKB ();
```

13

SAS Data Quality Server System Options

SAS Data Quality Server System Options

The SAS Data Quality Server system options DQLOCALE= and DQSETUPLOC= must be asserted before you run data cleansing programs. The DQOPTIONS= system option is used at SAS invocation to set data quality parameters.

To specify values for the DQLOCALE= and DQSETUPLOC= system options, use the "%DQLOAD AUTOCALL Autocall Macro" on page 83.

CAUTION
It is not recommended that you specify these system options by any means other than invoking the %DQLOAD AUTOCALL macro. Failure to use %DQLOAD or misapplied use of default settings for these system options can result in data that is cleansed with inappropriate locales.

System Options:

- DQLOCALE must be run before running data cleansing programs. See "DQLOCALE= System Option" on page 150 for additional information.
- The DQOPTIONS= system option enables you to optimize your SAS session for data quality. The value of the system option is a set of option-value pairs that you specify on the SAS start-up command or in the SAS configuration file. The data quality system options can be referenced by the OPTIONS procedure by specifying GROUP=DATAQUALITY. See: "DQOPTIONS= System Option" on page 151.
- DQSETUPLOC must be run before running data cleansing programs. See "DQSETUPLOC= System Option" on page 152 for additional information.

Dictionary

DQLOCALE= System Option

Specifies a locale, or an ordered list of locales

Valid in:	Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Input Control: Data Quality
PROC OPTIONS GROUP=	DATAQUALITY
Requirement:	You must specify at least one locale.

Syntax

DQLOCALE=(*locale-1* <, *locale-2*, …>)

Action

***locale-1* <, *locale-2*, ...>**
specifies the QKB ISO codes for the default locale and an optional ordered list of locales. The ordered list is used only for locale guessing with the DQLOCALEGUESS function. If a list is specified, the default locale is the first locale in the list. All of the locales in the list must exist in the Quality Knowledge Base.

Details

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. Because 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.

CAUTION
It is recommended that you invoke the AUTOCALL macro %DQLOAD at the beginning of each data cleansing program or session. Failure to do so might 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 enable 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. See "%DQLOAD AUTOCALL Autocall Macro" on page 83 for additional information.

DQOPTIONS= System Option

Specifies SAS session parameters for data quality programs.

Valid in:	Configuration file, SAS invocation
Category:	Environment Control: Initialization and Operation
PROC OPTIONS GROUP=	EXECMODES

Syntax

DQOPTIONS=(DQSRVPROTOCOL= | <TRANSCODE= | ><IWA= | >)

Required Argument

DQSRVPROTOCOL= |
specifies the SAS Data Quality Server protocol. In operating environments, other than z/OS, the default SOAP protocol is recommended.

SOAP
specifies to use the Simple Object Access Protocol (SOAP).

WIRELINE
specifies the Wireline protocol in the z/OS operating environment. The Wireline protocol improves data transfer performance in z/OS. In the SAS Data Quality Server software, z/OS support encompasses the DMSRVDATASVC procedure and all functions.

Requirement The Wireline protocol must be specified in the z/OS operating environment.

Optional Arguments

TRANSCODE= |
specifies whether transcoding errors end SAS processing.

- Errors can also occur when transcoding the locale's character set into the character set that is used in the SAS session.
- Transcoding errors can occur if characters in the source data cannot be converted into the character set that is used by the selected locale.

IGNORE
prevents writing of transcoding warning messages to the SAS log. SAS processing continues and ignores any transcoding errors.

WARN
writes transcoding error messages to the SAS log, and SAS stops processing.

Default A value is not supplied for the TRANSCODE= option.

IWA= |

specifies whether Integrated Windows Authentication is enabled. When IWA is enabled, the SAS administrator adds IWA=YES to the configuration file of the SAS Workspace Server. When a SAS Data Quality Server procedure or function attempts to connect to a DataFlux Data Management Server, it checks DQOPTIONS. If IWA=YES, then the procedure or function loads the code that supports IWA.

NO

specifies that connections from SAS Workspace Server to DataFlux Data Management Server use the function DMSVRUSER.

YES

specifies that connections to DataFlux Data Management Server generate passwords to support IWA, without using the function DMSVRUSER. Passwords are generated once for each server connection.

Restriction The SAS Workspace Server and DataFlux Data Management Server must be registered on the same instance of SAS Metadata Server.

DQSETUPLOC= System Option

Specifies the name of the Quality Knowledge Base.

Valid in:	SAS Viya 4.0.1 and later with Compute Server, Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Input Control: Data Quality
PROC OPTIONS GROUP=	DATAQUALITY

Syntax

DQSETUPLOC=('*qkb-reference*')

Required Argument

qkb-reference

- in SAS 9 and SAS Viya 3.5 and previous versions, identifies the directory that is the root of the Quality Knowledge Base.
- in SAS Viya 4.0.1 and later, identifies the name of the QKB.

Details

QKBs are located in the QKB Repository. For more information, see "Overview" in *SAS Viya Administration: QKB Management*.

To get a list of all QKBs that are contained in the QKB Repository, use the QKB procedure. For more information, see Overview: QKB Procedure on page 81.

14

SAS Data Quality Session Options

CAS Session Options

SAS Data Quality supports the following session options for SAS Cloud Analytic Services (CAS):

Set the following session options before running data cleansing programs in CAS:

- The "DQLOCALE= Session Option" specifies one or more locales for the current CAS session.
- The "DQSETUPLOC= Session Option" specifies the name of the Quality Knowledge Base that will be referenced in the current CAS session.

Dictionary

DQLOCALE= Session Option

Specifies a locale, or an ordered list of locales, in a SAS Cloud Analytic Services (CAS) session

Valid in:	CAS
Category:	Input Control: Data Quality
Requirement:	You must specify at least one locale.

Syntax

DQLOCALE=(*locale-1* <, *locale-2*, …>)

Action

locale-1 <, locale-2, ...>
specifies the QKB ISO codes for the default locale and an optional ordered list of locales. The ordered list is used only for locale guessing with the DQLOCALEGUESS function. If a list is specified, the default locale is the first locale in the list. All of the locales in the list must exist in the Quality Knowledge Base.

Details

If the locale or locales are not found in the QKB, then you may need to select another QKB with the DQSETUPLOC= session option.

To learn more about setting session options, see Setting Session Options.

DQSETUPLOC= Session Option

Specifies the name of the Quality Knowledge Base that will be referenced in the current CAS session.

Valid in:	CAS
Category:	Input Control: Data Quality

Syntax

DQSETUPLOC=('*qkb-name*')

Required Argument

qkb-name
specifies the name of the QKB. The name that is required is the name that was entered when the QKB was imported into CAS. The name of all of the QKBs in CAS are displayed in SAS Environment Manager. To learn more, see List QKBs.

Details

The DQSETUPLOC= session option specifies the name of the QKB that will be referenced by data quality language elements that run in the current CAS session.

Note: The QKB must be imported into CAS before the server can access the QKB. QKB import is managed by administrators using SAS Environment Manager. To learn more, see Import a QKB.

To learn more about setting session options, see Setting Session Options.