SAS® Data Quality 3.3 and SAS® 9.4 Data Quality Server: Language Reference

SAS® Documentation
January 12, 2018

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS® Data Quality 3.3 and SAS® 9.4 Data Quality Server: Language Reference*. Cary, NC: SAS Institute Inc.

SAS® Data Quality 3.3 and SAS® 9.4 Data Quality Server: Language Reference

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

January 2018

9.4_01-P2:dqclref

Contents

What's New in SAS Data Quality

Overview

SAS 9.4M5 Data Quality Server and SAS Data Quality 3.3 provide the following enhancements:

- Combined documentation for SAS Data Quality 3.3 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

Combined Documentation for SAS Data Quality 3.3 and SAS 9.4M5 Data Quality Server

In SAS 9.4M5 and SAS Viya 3.3, this document addresses all aspects of SAS 9.4M5 Data Quality Server and SAS Data Quality 3.3.

SAS Data Quality 3.3 provides full language element support in SAS, and adds data quality support for SAS Cloud Analytic Services (CAS). CAS runs in SAS Viya 3.3.

Perform Data Quality Operations on Data in CAS

Most of the data quality functions can be used in DATA step programs that 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.

Chapter 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.

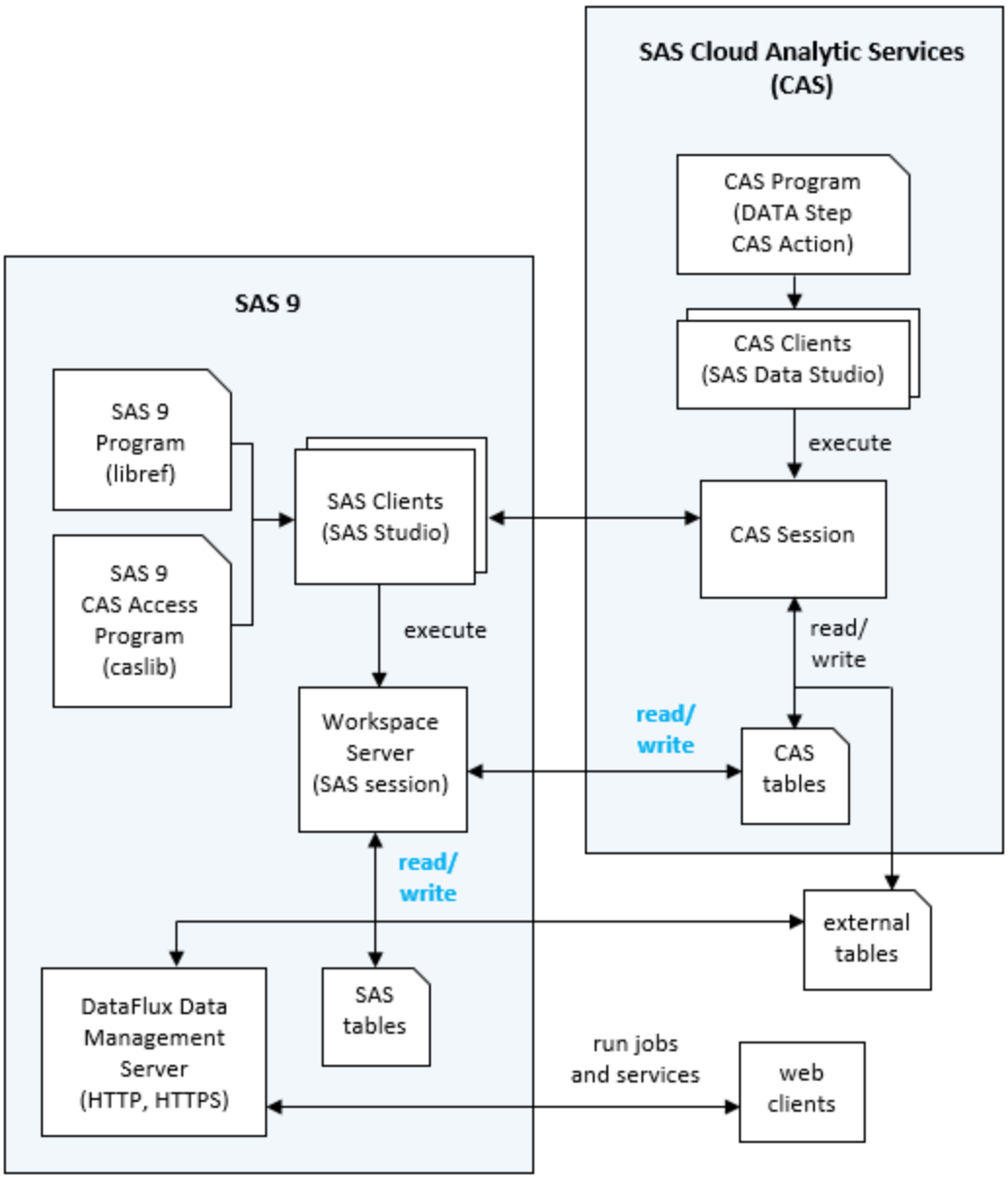
About SAS Data Quality in CAS

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.



Chapter 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.

Many of the following prerequisites are demonstrated in Appendix 1 - Examples.

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, the QKB must first be imported into CAS. To import a QKB into CAS, administrators use SAS Environment Manager. QKB import normally takes place during CAS deployment.

Specify a QKB and 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:

- "DQPARSE CALL Routine" on page 99
- "DQPARSEINFOGET Function" on page 100
- "DQTOKEN Function" on page 116
- "DQPARSETOKENGET Function" on page 104

- "DQPARSETOKENPUT Function" on page 105

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 83
- "DQEXTINFOGET Function" on page 82
- "DQEXTTOKENGET Function" on page 84
- "DQEXTTOKENPUT Function" on page 85

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 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 81. Standardization definitions are referenced by the "DQSTANDARDIZE Function" on page 115.

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 Definitions

Gender analysis, locale guess, locale score, and identification 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 source string
- the confidence score of a locale for a given source string
- the category of a source string, which is chosen from a set of available categories

Gender analysis definitions determine the gender of an individual based on that individual's name. The gender is determined to be unknown if the 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 86.

Locale guess definitions allow 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 DQLOCALEGUESS on page 90 function and the DQLOCALESCORE function.

Identification definitions are used to categorize source strings. For example, using the Entity identification definition in the ENUSA locale, a name value can apply to an individual or an organization. Identification definitions are referenced by the DQIDENTIFY function “DQIDENTIFY Function” on page 89.

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 customization in CAS, see QKB Management. To learn more about customization in SAS 9, see Managing and Customizing QKBs.

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.

Chapter 3

DMSRVADM Procedure

Overview: DMSRVADM Procedure

What Does the DMSRVADM Procedure Do?

The DMSRVADM procedure creates a job status data set that provides the name, type, and description of all DataFlux Data Management jobs. The output data set includes jobs that ran or that are running on a on a DataFlux Data Management Server. Status information is provided for 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.

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 65 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.

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 79.

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;
```

Chapter 4

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.

The Input and Output Data Sets

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;
```

Chapter 5

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.

The Input and Output Data Sets

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

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;
```

Chapter 6

DQLOCLST Procedure

Overview: DQLOCLST

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

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.

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

Chapter 7

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 match code (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 match code 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 *option(s)*;

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*>
<CLUSTER_BLANKS | NO_CLUSTER_BLANKS>
<CLUSTERS_ONLY>
<DELIMITER | NODELIMITER>
<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*>
<DELIMSTR=*input-variable-name* | VAR=*input-variable-name*>
<EXACT | MATCHDEF='*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 95.

DELIMSTR= | VAR=
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 98 for additional information.

"DQPARSETOKENPUT Function" on page 105 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 7.1 *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 match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

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

"Example 2: Matching Values Using Mixed Sensitivity Levels" on page 32 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 31 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 7.2 *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 match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that six of the character values are grouped in a single cluster and that the other two are not part of any cluster. The clustering is based on the values of the MATCH_CD variable.

This result is different than in "Example 1: Generate Composite Matchcodes" on page 31, 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 7.3 *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 match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

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

```
Mr. Raul 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 7.4 *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 7.5 *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 match code with a single DQMATCH procedure step. The first example, created a composite match code 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 7.6 *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 match code values were combined to form a composite match code 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 31 to compare the examples.

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

Chapter 8

DQSCHEME Procedure

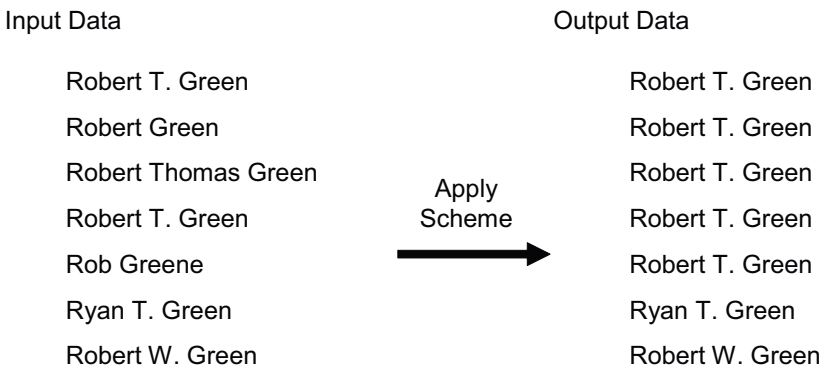
Overview: DQSCHEME Procedure

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 “DQSCHEMEAPPLY CALL Routine” on page 111.

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 8.1 *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 <option(s)>;
CONVERT <option(s)>;
CREATE <option(s)>;
```

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>
   <QKB | NOQKB>
     <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 50.

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=ELEMENT | PHRASE>
<SCHEME=*scheme-name*>
<SCHEME_LOOKUP =EXACT | IGNORE_CASE | USE_MATCHDEF>
<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=ELEMENT | PHRASE
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 match code 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 =EXACT | IGNORE_CASE | USE_MATCHDEF
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 match code of an input value is identical to the match code of a DATA value in the scheme.

See "Meta Options" on page 50 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 50 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.

Note:

CONVERT Statement

Converts schemes between SAS and QKB scheme file formats.

Requirement: All options are required.

Syntax

CONVERT <QKBTOSAS | SASTOQKB>
<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=PHRASE | ELEMENT>
<SCHEME=*scheme-name*>
<SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF>
<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 50 for additional information.

MODE=ELEMENT | PHRASE
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 match code 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 42 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=EXACT | IGNORE_CASE | USE_MATCHDEF
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 match code of an input value is identical to the match code 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 50 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 50 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.

Using PROC DQSCHEME

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 49 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 match code 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 match code
- 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 8.1 *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 8.2 *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.

Chapter 9

AUTOCALL Macros

Dictionary

%DQLOAD 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 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 105 for additional information.

It also enables the creation of matchcodes for parsed character values. See "DQMATCHPARSED Function" on page 96 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 91
- "DQLOCALEINFOLIST Function" on page 92

%DQUNLOAD 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

Macro Resources

See: *SAS Macro Language: Reference*

SAS Language Reference: Concepts

Chapter 10

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 in programs that run in SAS Cloud Analytic Services (CAS).

"DQCASE Function"	"DQEXTINFOGET Function"
"DQEXTRACT Function"	"DQEXTTOKENGET Function"
"DQEXTTOKENPUT Function"	"DQGENDER Function"
"DQGENDERINFOGET Function"	"DQGENDERPARSED Function"
"DQIDENTIFY Function"	"DQLOCALEGUESS Function"
"DQLOCALEINFOGET Function"	"DQLOCALEINFOLIST Function"
"DQLOCALESCORE Function"	"DQMATCH Function"
"DQMATCHINFOGET Function"	"DQMATCHPARSED Function"

Functions Listed Alphabetically

- The "DMSRVBATCHJOB Function" on page 70 runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.
- The "DMSRVCOPYLOG Function" on page 72 copies a job's log file from a DataFlux Data Management Server.
- The "DMSRVDELETELOG Function" on page 73 deletes a job's log file from a DataFlux Data Management Server.
- The "DMSRVJOBSTATUS Function" on page 76 returns the status of a job that was submitted to a DataFlux Data Management Server.
- The "DMSRVKILLJOB Function" on page 76 terminates a job that is running on a DataFlux Data Management Server.
- The "DMSRVPROFILEJOB Function" on page 78 generates a profile from a Data Management Server repository.
- The "DMSRVUSER Function" on page 79 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" on page 80 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 in an extraction definition.
- The "DQEXTRACT Function" returns an extracted character value.
- The "DQEXTTOKENGET Function" returns a token from an extraction character value.
- The "DQEXTTOKENPUT Function" inserts a token into an extraction character value and returns the updated extraction 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 a gender determination from the parsed name of an individual.

- The "DQIDENTIFY Function" returns a category name from a character value.
- The "DQLOCALEGUESS Function" returns the name of the locale that is most likely represented by a character value.
- The "DQLOCALEINFOGET Function" returns information about locales.
- The "DQLOCALEINFOLIST Function" returns the names of the definitions in a locale and returns a count of those definitions.
- The "DQLOCALESCORE Function" returns an integer confidence score for a character value and a locale.
- The "DQMATCH Function" returns a match code from a character value.
- The "DQMATCHINFOGET Function" on page 95 returns the name of the parse definition that is associated with a match definition.
- The "DQMATCHPARSED Function" on page 96 returns a match code from a parsed character value.
- The "DQOPTSURFACE Function" reveals or hides non-surfaced definitions.
- 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, and 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.
- 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" returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.
- The "DQTOKEN Function" returns a token from a character value.
- The "DQVER Function" on page 117 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" on page 70 runs a DataFlux Data Management Studio process or data job on a DataFlux Data Management Server and returns a job identifier.
- The "DMSRVCOPYLOG Function" on page 72 copies a job's log file from a DataFlux Data Management Server.
- The "DMSRVDELETELOG Function" on page 73 deletes a job's log file from a DataFlux Data Management Server.
- The "DMSRVJOBSTATUS Function" on page 74 returns the status of a job that was submitted to a DataFlux Data Management Server.
- The "DMSRVKILLJOB Function" on page 76 terminates a job that is running on a DataFlux Data Management Server.
- The "DMSRVPROFILEJOB Function" on page 78 generates a profile from a repository on a DataFlux Data Management Server.
- The "DMSRVUSER Function" on page 79 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" on page 80 returns the version of the DataFlux Data Management Server.

Case Functions

- The "DQCASE Function" on page 81 returns a character value with standardized capitalization.

Gender Analysis, Locale Guessing, and Identification Functions

The gender analysis, locale guessing, and identification functions return information that is determined from the 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 a gender determination from the parsed name of an individual.
- The "DQIDENTIFY Function" returns a category name from a character value.
- The "DQLOCALEGUESS Function" returns the name of the locale that is most likely represented by a character value.
- The "DQLOCALEINFOGET Function" returns information about locales.
- The "DQLOCALEINFOLIST Function" returns information about locales.

- The "DQLOCALESCORE Function" returns information about locales.
- The "DQMATCHPARSED Function" on page 96 returns the names of the definitions in a locale and returns a count of those definitions.

Matching Functions

- The "DQMATCH Function" returns a match code from a character value.
- The "DQMATCHINFOGET Function" returns the name of the parse definition that is associated with a match definition.
- The "DQMATCHPARSED Function" on page 96 returns a match code from a parsed character variable.

Parsing Functions

- The "DQPARSE CALL Routine" on page 99 returns a parsed character value and a status flag.
- The "DQPARSE Function" on page 98 returns a parsed character value.
- The "DQPARSEINFOGET Function" on page 100 returns the token names for the specified parse definition.
- The "DQPARSEINPUTLEN Function" on page 101 sets the default length of parsed input. DQPARSEINPUTLEN also returns a string indicating its previous value.
- The "DQPARSERESLIMIT Function" on page 102 sets a limit on resources consumed during parsing.
- The "DQPARSESCORDEPTH Function" on page 103 specifies how deeply to search for the best parsing score.
- The "DQPARSETOKENGET Function" on page 104 returns a token from a parsed character value.
- The "DQPARSETOKENPUT Function" on page 105 inserts a token into a parsed character value and returns the updated parsed character value.

Extraction Functions

- The "DQEXTINFOGET Function" on page 82 returns the token names in an extraction definition.
- The "DQEXTRACT Function" on page 83 returns an extracted character value.
- The "DQEXTTOKENGET Function" on page 84 returns a token from an extraction character value.
- The "DQEXTTOKENPUT Function" on page 85 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 a gender determination from the parsed name of an individual.
- The "DQIDENTIFY Function" returns a category name from a character value.
- The "DQLOCALEGUESS Function" returns the name of the locale that is most likely represented by a character value.
- The "DQLOCALEINFOGET Function" returns information about locales.
- The "DQLOCALEINFOLIST Function" returns the names of the definitions in a locale and returns a count of those definitions.
- The "DQLOCALESCORE Function" returns the names of the definitions in a locale and returns a count of those definitions.
- The "DQMATCH Function" returns a match code from a character value.
- The "DQMATCHINFOGET Function" returns the name of the parse definition that is associated with a match definition.
- The "DQMATCHPARSED Function" on page 96 returns a match code from a parsed character value.
- The "DQPARSE CALL Routine" returns a parsed character value and a status flag.
- The "DQPARSE Function" on page 98 returns a parsed character value.
- The "DQPARSEINFOGET Function" returns the token names for the specified parse definition.
- 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.
- 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" on page 107 applies a scheme and returns a transformed value after applying a scheme.
- The "DQSTANDARDIZE Function" returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.
- The "DQTOKEN Function" returns a token from a 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" on page 107 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" returns a character value after standardizing the casing, spacing, and format, and after applying a common representation to certain words and abbreviations.

Dictionary

DMSRVBATCHJOB Function

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

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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 72
- “DMSRVDELETELOG Function” on page 73
- “DMSRVJOBSTATUS Function” on page 74

DMSRVCOPYLOG Function

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

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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 70
- "DMSRVDELETELOG Function" on page 73
- "DMSRVJOBSTATUS Function" on page 74

DMSRVDELETELOG Function

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

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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 70
- "DMSRVCOPYLOG Function" on page 72
- "DMSRVJOBSTATUS Function" on page 74

DMSRVJOBSTATUS Function

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

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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), or 2 (job running). This value invalidates the *interval* argument.
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), or 2 (job running).

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 70
- "DMSRVDELETELOG Function" on page 73
- "DMSRVKILLJOB Function" on page 76

DMSRVKILLJOB Function

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

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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 70
- "DMSRVJOBSTATUS Function" on page 74

DMSRVPROFILEJOB Function

Generates a profile from a Data Management server repository.

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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

Functions

- "DMSRVJOBSTATUS Function" on page 76
- "DMSRVKILLJOB Function" on page 77

DMSRVUSER Function

Registers a user on a DataFlux Data Management Server.

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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: DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in 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 70

DQCASE Function

Returns a character value with standardized capitalization.

Valid in: CAS DATA step, and SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

DQCASE(*source-string*, '*case-definition*' <, '*locale*'>)

Required Arguments

source-string
specifies a text constant, text variable, or expression that resolves to a text constant.

case-definition
specifies the case definition that is referenced to transform the *source-string*. The definition must be available in the specified *locale*. If the value of *source-string* is represented by a specific case definition, then the use of that definition is recommended over the generic case definition. For example, if the value of *source-string* 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.

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

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

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 token names in an extraction definition.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Argument

extraction-definition
specifies the name of the extraction definition. The definition must exist in the specified locale.

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

The DQEXTINFOGET function returns the names of the tokens that can be inserted into character values using the DQEXTTOKENPUT function.

Example: DQEXTINFOGET Function

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

```
tokenNames=dqExtInfoGet('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 extraction definition.

See Also

- "DQEXTTOKENGET Function" on page 84
- "DQEXTTOKENPUT Function" on page 85
- "DQEXTRACT Function" on page 83

DQEXTRACT Function

Returns an extracted character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Always use the DQEXTTOKENGET function to retrieve tokens from extracted values.

Syntax

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

Required Arguments

extraction-string
specifies a constant, text variable, or expression that evaluates to a constant.

extraction-definition
specifies the name of the extraction definition. The definition must exist in the specified locale.

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

The DQEXTRACT function returns an extracted text value. The return value is a delimited text string. The string consists of a series of label and value pairs. The labels correspond to the tokens that can be extracted from the delimited string. To extract a name and value pair from the delimited string, use the use DQEXTTOKENGETfunction.

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', 'NAME', 'ENUSA');
prefix=dqExtTokenGet(extValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqExtTokenGet(extValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of PREFIX is **`Mr.`** and the value of GIVEN is **`James`**.

See Also

- "DQEXTTOKENGET Function" on page 84
- "DQTOKEN Function" on page 116

DQEXTTOKENGET Function

Returns a token from an extraction character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Do not attempt to retrieve tokens from extraction values using any means other than the DQEXTTOKENGET function.

Syntax

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

Required Arguments

extraction-string
specifies a constant, a text variable, or an expression that contains delimited elements that were previously extracted.

token
the name of the token in the *extraction-string*, the value of which will be returned. The token must be enabled by the specified extraction definition.

extraction-definition
the name of the extraction definition. The definition must exist in the locale that is used. The extraction definition must be the same as the extraction definition that was used to create the *extraction-string*.

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

The DQEXTTOKENGET function returns the value of the specified token from a previously extracted character value.

Example

The following example extracts a character value with the DQEXTRACT function and extracts two of the tokens with the DQEXTTOKENGET function.

```
extValue=dqExtract('Mr. James Joseph Westly', 'NAME', 'ENUSA');
prefix=dqExtTokenGet(extValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqExtTokenGet(extValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of **`prefix`** is **`Mr.`** and the value of **`given`** is **`James`**.

See Also

- "DQEXTRACT Function" on page 83
- "DQEXTINFOGET Function" on page 82
- "DQTOKEN Function" on page 116

DQEXTTOKENPUT Function

Inserts a token into an extraction character value and returns the updated extraction character value.

Valid in: CAS DATA step, and in SAS 9 DATA step and SAS Component Language

Syntax

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

Required Arguments

extraction-string

specifies a constant, a text variable, or an expression that resolves to a text variable or constant. The value is a delimited string of name and value pairs that was generated by an extraction definition.

token-value

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

token-name
the name of the token. The specified token must be enabled by the extraction definition.

extraction-definition
the name of the extraction definition. The definition must exist in the specified locale. The extraction definition must be the same definition that was used to extract the *extraction-string*.

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

The DQEXTTOKENPUT function enables you to insert a new value that is associated with a specified token into an extracted value. If a value exists for that token in the input value, the new value is inserted before the existing value. The existing value is retained.

You can specify a variable name for the value of *extraction-string*, and then assign the return value from DQEXTTOKENPUT to the same variable.

See Also

- "DQGENDERINFOGET Function" on page 87
- "DQGENDERPARSED Function" on page 88
- "DQMATCHPARSED Function" on page 96
- "DQEXTTOKENGET Function" on page 84

DQGENDER Function

Returns a gender determination from the name of an individual.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

source-string
specifies a character constant, variable, or expression that contains the value that is evaluated to determine the gender.

gender-analysis-definition
specifies the gender analysis definition, which must exist in the specified locale. The value must be the name of a text variable in quotation marks, an expression that evaluates to a variable name, or a quoted value is also valid.

Optional Argument

locale
specifies a character constant, variable, or expression that contains the locale name.

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', 'Gender', 'ENUSA');
```

The *gender-analysis-definition* must exist in the specified locale. Because recent versions of the locales use the variable NAME instead of GENDER, this example could also be coded as follows:

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

See Also

"DQGENDERPARSED Function" on page 88

DQGENDERINFOGET Function

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

Valid in: SAS DATA step, 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 value can be a constant, a character variable, or an expression that evaluates to a constant or variable.

Optional Argument

locale
specifies a character 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 enabled for that parse definition. The tokens are then used to construct a parsed value and 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 86
- "DQGENDERPARSED Function" on page 88
- "DQPARSE Function" on page 98
- "DQPARSETOKENPUT Function" on page 105

DQGENDERPARSED Function

Returns the gender of an individual.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

parsed-string
the parsed value that is analyzed to determine the gender of an individual. The value can be a constant, a character variable, or an expression that evaluates to a constant or variable.

gender-analysis-definition
specifies the name of the gender analysis definition that will be referenced to determine gender. The analysis definition must exist in the locale that is used.

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

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

See Also

- "DQGENDER Function" on page 86
- "DQGENDERINFOGET Function" on page 87

DQIDENTIFY Function

Returns a category name from a character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

source-string
specifies a constant, character variable, or expression that evaluates to the value will be analyzed. .

identification-definition
the name of the identification definition. The definition must be in the specified locale.

Optional Argument

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

Example: DQIDENTIFY Function

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

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

After this function call, the value of DQID is Organization.

DQLOCALEGUESS Function

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

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

DQLOCALEGUESS(*source-string*, '*locale-guess-definition*')

Required Arguments

source-string
specifies a constant, character variable, or expression that evaluates to the value that will be analyzed.

locale-guess-definition
specifies a constant, character variable, or expression that evaluates to the name of the locale guess definition that will be referenced during analysis.

Details

The DQLOCALEGUESS function applies the source string to the specified locale guess definition in each of the locales that are listed in the DQLOCALE= system option or the 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 exist in each of the locales that are listed in the DQLOCALE= system option. If a locale does not contain a locale guess definition, then the DQLOCALEGUESS function fails to execute.

If two or more locales receive the highest confidence score, then the DQLOCALEGUESS function selects a return value as follows. The return value is the ISO code of the highest scoring locale that appears last in the value of the DQLOCALE= system option.

If all locales receive a confidence score of zero, then the DQLOCALEGUESS function returns a value that depends on a setting in the locale guess definitions. 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 91

DQLOCALEINFOGET Function

Returns information about locales.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SCL

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 codes. 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 92
- "%DQPUTLOC AUTOCALL Macro" on page 60

DQLOCALEINFOLIST Function

Returns the names of the definitions in a locale and a count of those definitions.

Valid in: CAS DATA step, and SAS 9 DATA step, PROC SQL, and SAS Component Language.

Syntax

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

Required Arguments

definition-type

specifies the value that is analyzed to determine the names and count of the definition type. The definition type must exist in the specified locale.

Definition types are as follows:

ALL
CASE
EXTRACTION
GENDER
GUESS
IDENTIFICATION
MATCH

PARSE
PATTERN
STANDARDIZATION

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

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

Interaction The DQLOCALEINFOLIST function writes the names of the type-definitions to the SAS log. The return value of the function is the total number of type-definitions.

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

- "%DQPUTLOC AUTOCALL Macro" on page 60
- "DQLOCALEINFOGET Function" on page 91

DQLOCALESCORE Function

Returns an integer confidence score for a character value and a locale.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SCL

Syntax

DQLOCALESCORE('*source-string*', '*locale-guess-definition*', '*locale*')

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the string that is analyzed to determine the confidence score.

locale-guess-definition
specifies a constant, character variable, or expression that evaluates to the name of the locale guess definition that is used to evaluate the source string.

Optional Argument

locale
specifies a character 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 DQLOCALESCORE function applies a source string to a specified locale guess definition in a specified locale. 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 90
- " %DQLOAD AUTOCALL Macro" on page 59

DQMATCH Function

Returns a match code from a character value.

Valid in: CAS DATA step, and SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

source-string
specifies a constant, character variable, or expression that evaluates to the value for which a match code is created.

match-definition
specifies the name of the match definition. The definition must exist in the specified 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 match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster.

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

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

Example: DQMATCH Function

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

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

See Also

Chapter 7, “DQMATCH Procedure,” on page 25

DQMATCHINFOGET Function

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

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

DQMATCHINFOGET('*match-definition*' <,'*locale*'>)

Required Argument

match-definition
the name of the match definition. The definition must exist in the specified locale.

Optional Argument

locale
specifies a constant, a character 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

The DQMATCHINFOGET function returns the name of the parse definition that is associated with the specified match definition. Obtaining the name of that parse definition enables you to create parsed character values with the DQPARSE or DQPARSETOKENPUT functions.

If the specified match definition does not have an associated parse definition, the DQMATCHINFOGET function will return a zero-length character variable. 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. That parse definition is then used to display the tokens that are enabled for that parse definition. The tokens are then used to construct a parsed value, create and return a match code, and display the match code.

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

DQMATCHPARSED Function

Returns a match code from a parsed character value.

Valid in: CAS, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Requirement: If specified, the locale must be loaded into memory as part of the locale list.

Syntax

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

Required Arguments

parsed-string

specifies the parsed value that is analyzed to determine the matchcode. The value can be a constant, a character variable, or an expression that evaluates to a constant or variable.

match-definition
specifies the name of the match definition. The definition must be supported in the specified 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 match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster.

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

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

Example: DQMATCHPARSED Function

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

Functions

- Chapter 7, “DQMATCH Procedure,” on page 25
- “DQMATCHINFOGET Function” on page 95
- “DQPARSEINFOGET Function” on page 100
- “DQTOKEN Function” on page 116

DQOPTSURFACE Function

Reveals or hides non-surfaced definitions.

Valid in: SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in CAS

Syntax

DQOPTSURFACE('*surface-definition*')

Required Argument

surface-definition
specifies the policy for the surface definitions.

Details

The DQOPTSURFACE function specifies whether the non-surfaced definitions are revealed or hidden. By default, non-surfaced definitions are hidden. Valid input values are as follows:

YES
reveals the non-surfaced definitions.

NO
hides the non-surfaced definitions.

The DQOPTSURFACE function returns the previous value of the surface definition policy.

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 parsed character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Always use the DQPARSETOKENGET function to extract tokens from parsed values. To extract tokens from values that do not contain delimiters, use the DQTOKEN function.

Syntax

DQPARSE('*parse-string*', '*parse-definition*' <,'*locale*'>)

Required Arguments

parse-string
the value that is parsed according to the specified parse definition. The value can be a constant, a character variable, or an expression that evaluates to a parse string.

parse-definition
the name of the parse definition that is referenced during the analysis of the input string. The definition must be supported in the specified locale.

Optional Argument

locale
specifies a character 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 DQPARSE function returns a parsed character value. The parsed character value contains delimited pairs of names and values. The names are the tokens that apply to the parse definition. The values are the components of the input string that apply to the tokens. To return the values of tokens from the output string, use DQPARSETOKENGET.

Example: DQPARSE Function

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

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

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

See Also

- "DQPARSEINFOGET Function" on page 100
- "DQTOKEN Function" on page 116

DQPARSE CALL Routine

Returns a parsed character value and a status flag.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Always use the DQPARSETOKENGET function to extract tokens from parsed values. To extract tokens from values that do not contain delimiters, use the DQTOKEN function.

Syntax

CALL DQPARSE(*parse-string*, *parse-definition*, '*parse-result*, *parse-return-code* <, '*locale* '>);

Required Arguments

parse-string
the input value that is parsed according to the specified parse definition. The value can be a constant, a character variable, or an expression that evaluates to the value that will be analyzed.

parse-definition
the name of the parse definition that is referenced during parsing. The definition must be supported in the specified locale.

parse-result
an output character variable that receives the result of the parse operation.

parse-return-code
an output numeric variable that returns 1 when the parse operation is successful. Otherwise, this variable receives a 0.

Optional Argument

locale
specifies a character 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 DQPARSE CALL routine returns a parsed character value and a return code into separate variables. The parsed character value contains delimited pairs of names and values. The names are the tokens that apply to the parse definition. The values are the components of the input string that apply to the tokens. To return the values of tokens from the output string, use DQPARSETOKENGET.

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

- "DQPARSEINFOGET Function" on page 100
- "DQTOKEN Function" on page 116

DQPARSEINFOGET Function

Returns the token names in a parse definition.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

DQPARSEINFOGET('*parse-definition*' <,'*locale*'>)

Required Argument

parse-definition
specifies the name of the parse definition. The definition must be supported by the specified locale.

Optional Argument

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

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

Details

The DQPARSEINFOGET function returns the names of the tokens that can be inserted into character values using the DQPARSETOKENPUT function.

Example: DQPARSEINFOGET Function

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

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

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

See Also

- "DQPARSETOKENGET Function" on page 104
- "DQPARSETOKENPUT Function" on page 105
- "DQTOKEN Function" on page 116

DQPARSEINPUTLEN Function

Overrides the default expected length of parsed input, and returns a string indicating its previous value.

Valid in: SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in CAS

Syntax

DQPARSEINPUTLEN('*input-length*')

Required Argument

input-length
specifies one of four available expected input lengths.

Details

The DQPARSEINPUTLEN function overrides the default expected input length that is specified in the parse definition. If REMOVE is specified, the expected input length is set to the default value in the parse defintion. Valid values for the input length are as follows:

- SHORT
- LONG
- AUTO
- REMOVE

The DQPARSEINPUTLEN function returns a value that indicates the previous value of the input length. If the value NOTSET is returned, the previous input length was not set. Possible values for the previous input length are as follows:

- SHORT
- LONG
- AUTO
- NOTSET

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 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in CAS

Syntax

DQPARSERESLIMIT('*resource-limit*')

Required Argument

resource-limit
specifies one of seven available resource limits for parsing operations.

Details

The DQPARSERESLIMIT function changes the default expected resource limit that is specified in the parse definition. If REMOVE is specified, the resource limit is set to the default value that is specified in the parse definition. Valid values are as follows:

- VERYLOW
- LOW

- MEDIUM
- HIGH
- VERYHIGH
- INTENSIVE
- REMOVE

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. t. Available return values are as follows:

- VERYLOW
- LOW
- MEDIUM
- HIGH
- VERYHIGH
- INTENSIVE
- NOTSET

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 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in CAS

Syntax

DQPARSESCORDEPTH(*level*)

Required Argument

level

an integer that specified the maximum depth permitted during scoring.

Range 5-10, and 0

Details

The DQPARSESCORDEPTH function changes the default level of search that will be used to find the best parsing score. The default parse score depth is specified in the parse definition. If 0 is specified, 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 token from a parsed character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

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

Syntax

DQPARSETOKENGET(*parsed-string*, '*token*', '*parse-definition'* <,'*locale*'>)

Required Arguments

parsed-string
the value from which a specified token and value are to be retrieved. The value can be a constant, a character variable, or an expression that evaluates to a parsed string.

To determine the tokens that are supported by a given parse definition, use the DQPARSEINFOGET function.

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

parse-definition
the name of the parse definition. The definition must be supported by the specified locale. The specified parse definition must be the same definition that was used to generate the *parsed-string*.

Optional Argument

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

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

Details

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

Example: DQPARSETOKENGET Function

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

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

After these function calls, the value of **prefix** is **Mrs.** and the value of **given** is **Sallie**.

See Also

Functions

- "DQPARSE Function" on page 98
- "DQPARSEINFOGET Function" on page 100
- "DQTOKEN Function" on page 116

DQPARSETOKENPUT Function

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

Valid in: CAS, and in SAS 9 DATA step and SAS Component Language

Syntax

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

Required Arguments

parsed-string
specifies a constant, a character variable, or an expression that evaluates to the parsed character value that will be updated.

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

token-name
the name of the token, the value of which will be updated. The specified token must be enabled by the specified parse definition.

parse-definition
the name of the parse definition that will be referenced to update the parsed string. The parse definition must be supported in the specified locale. The parse definition must be the same definition that was used to parse the *parsed-string*.

Optional Argument

locale
specifies a constant, a character variable, or an expression that evaluates to the ISO codeof the locale.

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

Details

The DQPARSETOKENPUT function enables you to insert a new value for a specified token. If a value exists for that token in the input parsed string, then the new value is inserted before the existing value. The existing value is retained.

You can specify a variable name for the value of the *parsed-string*, and then assign the return value from DQPARSETOKENPUT to the same variable.

See Also

- "DQGENDERPARSED Function" on page 88
- "DQMATCHPARSED Function" on page 96
- "DQPARSETOKENGET Function" on page 104

DQPATTERN Function

Returns a pattern analysis from an input character value.

Valid in: CAS DATA step, and in SAS 9 DATA step and SAS Component Language

Syntax

DQPATTERN('*source-string*' , '*pattern-analysis-definition*' <,'*locale*'>)

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the source string that will be analyzed.

pattern-analysis-definition
the name of the pattern analysis definition. The definition must be supported in the specified locale.

Optional Argument

locale
specifies a constant, a character 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

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

Valid in: SAS 9 DATA step, PROC SQL, and SAS Component Language

Restriction: Not valid in CAS

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

Syntax

DQSCHEMEAPPLY(*source-string*, '*scheme*', '*scheme-format*' <,'*mode*'> <,'*scheme-lookup-method*'>
<,'*match-definition*'> <,*sensitivity*> <,'*locale*'>)

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the string that is to be transformed by the scheme.

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 49

Optional Arguments

mode
specifies how the scheme is to be applied to the source string..

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 match code 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 *source-string* to each of the DATA values in the scheme. When the value of the *scheme-lookup-method* is USE_MATCHDEF, the match code of the entire input 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 input 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 one of three mutually exclusive methods of applying the scheme.

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

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

USE_MATCHDEF
specifies that the matchcode of the input 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 source string. The match definition must be supported in the specified 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 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 50

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 match code. At lower sensitivity values, values can receive the same match code 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.

Note To return a count of the number of transformations that take place during a scheme application, use the DQSCHEMEAPPLY CALL routine.

See "Meta Options" on page 50

locale

specifies a constant, a character variable, or an 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 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 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 8, "DQSCHEME Procedure," on page 41

DQSCHEMEAPPLY CALL Routine

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

Valid in: SAS 9 DATA step and SAS Component Language

Restriction: Not valid in CAS

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

Syntax

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

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the input value to which the scheme will be applied.

output-variable
the character 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 the 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 49

Optional Arguments

mode

specifies how the scheme is to be applied to the values of the input character variable. The default value of *mode* is the mode that is stored in the scheme. If a mode is not stored in the scheme, 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*, the default value of *mode* (PHRASE) is used.

Valid values for *mode* are defined 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 input character value to each of the DATA values in the scheme. When the value of the *scheme-lookup-method* is USE_MATCHDEF, the match code of the entire input 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 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 input variable 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 value of *mode* is PHRASE and the input value is not transformed, then the value of the *transform-count-variable* is 0.

If the input variable is transformed, the value of *transform-count-variable* is 1.

If the value of the *mode* is ELEMENT and the input value is not transformed, then the value of the *transform-count-variable* is 0.

The transformation count might appear to be inaccurate if the transformation value in the scheme is the same as the input value (or any element in the input value).

scheme-lookup-method

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

EXACT

(default value) specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank

spaces in the input value are replaced with single blank spaces before comparison.

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

USE_MATCHDEF
specifies that the matchcode of the input 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 used to generate a matchcode for the source string. The definition must be supported by the specified locale.

Interactions If USE_MATCHDEF is specified and if *match-definition* is not specified, 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 source string. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, two values receive the same match code despite their dissimilarities.

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 s 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, a character 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 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 is 1 if the organization name is transformed.

In the following example, a transformation count of 1 is shown in instances, 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 8, "DQSCHEME Procedure," on page 41

DQSTANDARDIZE Function

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

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the input string that will be standardized.

standardization-definition
specifies the name of the standardization definition that is referenced to standardize the source string. The definition must be supported in the specified locale.

Optional Argument

locale
specifies a constant, a character 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

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

The return value is provided in the appropriate case, with insignificant blank spaces and punctuation removed. The standardization definition that was specified in the DQSTANDARDIZE function can standardize certain words and abbreviations. 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 after parsing a character value.

Valid in: CAS DATA step, and in SAS 9 DATA step, PROC SQL, and SAS Component Language

Syntax

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

Required Arguments

source-string
specifies a constant, a character variable, or an expression that evaluates to the string that is to be parsed.

token
identifies the token that provides the return value.

parse-definition
the name of the parse definition that is referenced to locate the token. The definition must be supported in the specified locale.

Optional Argument

locale
specifies a constant, a character 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

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

To return more than one token from a parsed value, use the functions DQPARSE and DQPARSETOKENGET.

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: CAS DATA step, and in SAS 9 DATA step, 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: CAS DATA step, and in SAS 9 DATA step, 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 ();
```

Chapter 11

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 Macro" on page 59.

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 120 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 121.
- DQSETUPLOC must be run before running data cleansing programs. See "DQSETUPLOC= System Option" on page 122 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 Macro" on page 59 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=WIRELINE | SOAP <TRANSCODE=IGNORE | WARN><IWA=NO | YES>)

Required Argument

DQSRVPROTOCOL=WIRELINE | SOAP
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=IGNORE | WARN
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=NO | YES
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 location of the root directory of the Quality Knowledge Base.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: Input Control: Data Quality

PROC OPTIONS GROUP= DATAQUALITY

Syntax

DQSETUPLOC=('*quality-knowledge-base-root-directory*')

Required Argument

quality-knowledge-base-root-directory
identifies the directory that is the root of the Quality Knowledge Base.

Chapter 12

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.

Appendix 1

Examples

Example 1 - Data Quality Program That Runs in CAS

The following example runs in SAS Cloud Analytic Services (CAS). The program cleanses the data in a CAS table.

```
/* Connect to CAS. Use session options to specify a QKB
   and load a locale into CAS memory. */
cas mysess cashost="myhost.sas.com"
    casport=12345 sessopts=(dqSetupLoc="qkbci" dqLocale="(ENUSA)");

/* Specify the active caslib */
cas mysess cassessopts=(caslib="CASUser");

proc cas;

/* Load the data to be processed. */
action table.loadTable /
    casOut={name="CUSTOMERS"}
    path="CUSTOMERS.sashdat";

/* Standardize two columns and determine the gender. */
action dataStep.runCode /
code="
data customers_out;
    length STD_NAME STD_ADDR1 $ 200;
    length GNDR_NAME $ 1;

    set CUSTOMERS;

    STD_NAME=dqStandardize(NAME, 'Name', 'ENUSA');
    STD_ADDR1=dqStandardize(ADDR1, 'Address', 'ENUSA');
    GNDR_NAME=dqGender(NAME, 'Name', 'ENUSA');
";
```

```
action table.fetch /
    table={name="customers_out"}
    to=1000.0;;
run;
```

Example 2 - SAS 9 Data Quality Program That Accesses Data in CAS

The following example runs in SAS 9. The program uses a caslib to access CAS data.

```
/*  Use a system option to specify a QKB. */
options dqsetuploc="/path/to/qkbci";

/* Use an autocall macro to specify and load into SAS 0 memory a locale. */
%dqload(dqlocale=(ENUSA));

/* Connect to CAS with a new session. */
cas mysess cashost="myhost.sas.com" casport=12345;

/* Specify a caslib */
cas mysess cassessopts=(caslib="CASUser");

proc cas;

/* Load the source table. */
action table.loadTable /
      casOut={name="CUSTOMERS"}
      path="CUSTOMERS.sashdat";
run;

/* Setup access to a CAS table. */
libname mycas cas sessref=mysess;

/* Standardize and determine gender. */
data customers_out;
    length STD_NAME STD_ADDR1 $ 200;
    length GNDR_NAME $ 1;

    set mycas.CUSTOMERS;

    STD_NAME=dqStandardize(NAME, 'Name', 'ENUSA');
    STD_ADDR1=dqStandardize(ADDR1, 'Address', 'ENUSA');
    GNDR_NAME=dqGender(NAME, 'Name', 'ENUSA');
run;

proc print data=customers_out;
run;
```

Appendix 2

Deprecated Language Elements

Dictionary

PROC DQSRVADM

The DQSRVADM procedure creates a data set that provides the name, type, and description of all DataFlux dfPower Architect and DataFlux dfPower Profile jobs.

See Also

- PROC DMSRVADM
- *SAS Data Quality Server 9.2*

PROC DQSRVSVC

Deprecated. The DQSRVSVC procedure runs a DataFlux dfPower Architect real-time service on a DataFlux Integration Server.

See Also

- PROC DMSRVDATASVC
- *SAS Data Quality Server 9.2*

DQSRVARCHJOB Function

Deprecated. Runs a DataFlux dfPower Architect job on a DataFlux Integration Server and returns a job identifier.

See Also

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- *SAS Data Quality Server 9.2*

DQSRVCOPYLOG Function

Deprecated. Copies a job's log file from a DataFlux Integration Server.

See Also

- DMSRVCOPYLOG function
- *SAS Data Quality Server 9.2*

DQSRVDELETELOG Function

Deprecated. Deletes a job's log file from a DataFlux Integration Server.

See Also

- DMSRVDELETELOG function
- *SAS Data Quality Server 9.2*

DQSRVJOBSTATUS Function

Deprecated. Returns the status of a job that was submitted to a DataFlux Integration Server.

See Also

- DMSRVJOBSTATUS function
- *SAS Data Quality Server 9.2*

DQSRVKILLJOB Function

Deprecated. Terminates a job that is running on a DataFlux Integration Server.

See Also

- DMSRVKILLJOB function
- *SAS Data Quality Server 9.2*

DQSRVPROFJOBFILE Function

Deprecated. Runs a file-type DataFlux dfProfile job on a DataFlux Integration Server and returns a job identifier.

See Also

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- *SAS Data Quality Server 9.2*

DQSRVPROFJOBREP Function

Deprecated. Runs a repository–type DataFlux dfProfile job on a DataFlux Integration Server and returns a job identifier.

See Also

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- *SAS Data Quality Server 9.2*

DQSRVUSER Function

Deprecated. Registers a user on a DataFlux Integration Server.

See Also

- DMSRVUSER function
- *SAS Data Quality Server 9.2*

DQSRVVER Function

Deprecated. Returns the version of the DataFlux Integration Server.

See Also

- DMSRVVER function
- *SAS Data Quality Server 9.2*

DQVERBF Function

Deprecated. Returns the version of Blue Fusion.

See Also

- DQVERBF function
- *SAS 9.3 Data Quality Server: Reference*

Recommended Reading

Here is the recommended reading list for this title:

- *SAS Language Reference: Concepts*
- *Base SAS Procedures Guide*
- *SAS Statements: Reference*
- *SAS System Options: Reference*
- *SAS Macro Language: Reference*
- SAS DataFlux documentation is available under each product's name on the support site's SAS Documentation (http://support.sas.com/documentation). See also the following documents:
  - *DataFlux Data Management Studio: User's Guide*
  - *DataFlux Data Management Server: Administrator's Guide*
  - *DataFlux Expression Language Reference Guide*
  - Depending on your role, you might want to review the SAS Federation Server, DataFlux Authentication Server, and DataFlux Secure documentation for users or administrators.
- See the *SAS Data Integration Studio: User's Guide* to learn about working with DataFlux Data Management Platform and data quality transformations, especially the information about creating matchcodes and applying lookup standardization.

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Glossary

analysis data set
in SAS data quality, a SAS output data set that provides information about the degree of divergence in specified character values.

case definition
a part of a locale that is referenced during data cleansing to impose on character values a consistent usage of uppercase and lowercase letters.

cleanse
to improve the consistency and accuracy of data by standardizing it, reorganizing it, and eliminating redundancy.

cluster
in SAS data quality, a set of character values that have the same match code.

composite match code
a match code that consists of a concatenation of matchcodes from values from two or more input character variables in the same observation. A delimiter can be specified to separate the individual matchcodes in the concatenation.

compound match code
a match code that consists of a concatenation of matchcodes that are created for each token in a delimited or parsed string. Within a compound match code, individual matchcodes might be separated by a delimiter.

data analysis
in SAS data quality, the process of evaluating input data sets in order to determine whether data cleansing is needed.

data cleansing
the process of eliminating inaccuracies, irregularities, and discrepancies from data.

data quality
the relative value of data, which is based on the accuracy of the knowledge that can be generated using that data. High-quality data is consistent, accurate, and unambiguous, and it can be processed efficiently.

data transformation
in SAS data quality, a cleansing process that applies a scheme to a specified character variable. The scheme creates matchcodes internally to create clusters. All

values in each cluster are then transformed to the standardization value that is specified in the scheme for each cluster.

delimiter
a character that serves as a boundary that separates the elements of a text string.

gender definition
a part of a locale that is referenced during data cleansing to determine the gender of individuals based on the names of those individuals.

guess definition
a part of a locale that is referenced during the selection of the locale from the locale list that is the best choice for use in the analysis or cleansing of the specified character values.

identification definition
a part of a locale that is referenced during data analysis or data cleansing to determine categories for specified character values.

locale
a setting that reflects the language, local conventions, and culture for a geographic region. Local conventions can include specific formatting rules for paper sizes, dates, times, and numbers, and a currency symbol for the country or region. Some examples of locale values are French_Canada, Portuguese_Brazil, and Chinese_Singapore.

locale list
an ordered list of locales that is loaded into memory prior to data analysis or data cleansing. The first locale in the list is the default locale.

match
a set of values that produce identical matchcodes or identical match code components. Identical matchcodes are assigned to clusters.

match code
an encoded version of a character value that is created as a basis for data analysis and data cleansing. Matchcodes are used to cluster and compare character values.

match definition
a part of a locale that is referenced during the creation of matchcodes. Each match definition is specific to a category of data content. In the ENUSA locale. For example, match definitions are provided for names, e-mail addresses, and street addresses, among others.

name prefix
a title of respect or a professional title that precedes a first name or an initial. For example, Mr., Mrs., and Dr. are name prefixes.

name suffix
a part of a name that follows the last name. For example, Jr. and Sr. are name suffixes.

parse
to analyze text, such as a SAS statement, for the purpose of separating it into its constituent words, phrases, punctuation marks, values, or other types of information. The information can then be analyzed according to a definition or set of rules.

parse definition
a part of a locale that is referenced during the parsing of character values. The parse definition specifies the number and location of the delimiters that are inserted during parsing. The location of the delimiters depends on the content of the character values.

parse token
a named element that can be assigned a value during parsing. The specified parse definition provides the criteria that detect the value in the string. After the value is detected and assigned to the token, the character value can be manipulated using the name of the token.

parsed string
in SAS data quality, a text string into which has been inserted a delimiter and name at the beginning of each token in that string. The string is automatically parsed by referencing a parse definition.

Quality Knowledge Base
a collection of locales and other information that is referenced during data analysis and data cleansing. For example, to create matchcodes for a data set that contains street addresses in Great Britain, you would reference the ADDRESS match definition in the ENGBR locale in the Quality Knowledge Base.

scheme
a reusable collection of matchcodes and standardization values that is applied to input character values for the purposes of transformation or analysis.

scheme format
schemes are created and applied as files. The scheme files are created in the QKB format or the NOQKB format by the DQSCHEME procedure.

sensitivity
in SAS Data Quality, a value that specifies the amount of information in matchcodes. Greater sensitivity values result in matchcodes that contain greater amounts of information. As sensitivity values increase, character values must be increasingly similar to generate the same matchcodes.

standardization definition
a part of a locale that is referenced during data cleansing to impose a specified format on character values.

standardize
to eliminate unnecessary variation in data in order to maximize the consistency and accuracy of the data.

token
in SAS data quality, a named word or phrase in a parsed or delimited string that can be individually analyzed and cleansed.

transformation
in data integration, an operation that extracts data, transforms data, or loads data into data stores.

transformation value
in SAS data quality, the most frequently occurring value in a cluster. In data cleansing, this value is propagated to all of the values in the cluster.

Index

E

F

G

N

O

P

T

U

V