



THE
POWER
TO KNOW.

SAS[®] Data Quality Server 9.2

Reference

Second Edition



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2010 *SAS® Data Quality Server 9.2: Reference, Second Edition*. Cary, NC: SAS Institute Inc.

SAS® Data Quality Server 9.2: Reference, Second Edition

Copyright © 2011, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-60764-450-7

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, May 2010

2nd electronic book, May 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contents

<i>What's New</i>	<i>v</i>
Overview	v
SAS Data Quality Server Enhancements	v
SAS System Option	vi
Support for the DataFlux Integration Server	vi
Support for the z/OS Operating Environment	vii
Chapter 1 \triangle Overview SAS Data Quality Server	1
Overview	1
Chapter 2 \triangle Concepts	3
SAS Data Quality Server Concepts	3
DataFlux Jobs and Services	5
Load and Unload Locales	7
Schemes	8
Create Match Codes	11
Clusters	12
Sensitivity	13
Chapter 3 \triangle Locale Definitions	15
Locale Definitions	15
Chapter 4 \triangle The DQMATCH Procedure	19
Overview: DQMATCH Procedure	19
Syntax: DQMATCH Procedure	20
DQMATCH Examples	24
Chapter 5 \triangle The DQSCHEME Procedure	33
Overview: DQSCHEME Procedure	33
Syntax: DQSCHEME Procedure	34
PROC DQSCHEME Examples	40
Chapter 6 \triangle The DQSRVADM Procedure	45
Overview: DQSRVADM Procedure	45
Syntax: DQSRVADM Procedure	45
The Job Status Data Set	46
Security	46
PROC DQSRVADM Examples	47
Chapter 7 \triangle The DQSRVSVC Procedure	49
Overview: DQSRVSVC Procedure	49
Syntax: DQSRVSVC Procedure	49
The Input and Output Data Sets	52
Examples	52

Chapter 8	△	AUTOCALL Macros	53
AUTOCALL Macros for SAS Data Quality Server			53
Chapter 9	△	Functions and CALL Routines	57
Overview			58
Functions Listed Alphabetically			58
Functions Listed by Category			60
Chapter 10	△	SAS Data Quality Server System Options	107
SAS Data Quality Server System Options			107
Appendix 1	△	Recommended Reading	111
Recommended Reading			111
Glossary			113
Index			117

What's New

Overview

The SAS Data Quality Server provides procedures and functions that enable you to administer and run jobs and services on DataFlux Integration Servers from DataFlux (a SAS company). The following enhancements were made in SAS 9.2:

- The SAS Data Quality Server has added support for grouping of clustering criteria, as well as a new CALL routine and new functions for parsing and version verification.
- A new SAS system option, DQOPTIONS, has options to increase data transfer rates and to control whether to terminate SAS execution if errors are encountered.
- New procedures, functions, and options have been added to support the DataFlux Integration Server.
- The DataFlux Integration Server now supports all the language elements that access DataFlux Integration Servers in the z/OS operating environment.

SAS Data Quality Server Enhancements

The DQMATCH procedure was enhanced to enable grouping of the clustering criteria into a series of conditions.

Starting in SAS 9.2 Phase 2, the DQPARSE function is complemented by the DQPARSE CALL routine. The new CALL routine returns a flag that indicates the status of the parse operation (success or failure).

In the third maintenance release for SAS 9.2, the SAS Data Quality Server was enhanced with several new functions:

- The DQOPTSURFACE function reveals or hides non-surfaced definitions.
- The DQPARSEINPUTLEN function specifies the input length for parsing functions.
- The DQPARSERESLIMIT function specifies the resource limit a parsing operation is allowed to consume.
- The DQPARSESCORDEPTH function specifies how deeply to search for the best parsing score.

- The DQVERQKB function returns the version of the currently loaded Quality Knowledge Base.
- The DQVERBF function returns the version of Blue Fusion.

SAS System Option

In SAS 9.2 Phase 2, the SAS system option DQOPTIONS was added. DQOPTIONS is available for use in the SAS start-up command and in the SAS configuration file. The value of the option is a series of option-value pairs.

- DQSRVPROTOCOL=WIRELINE increases data transfer performance to and from services.
- TRANSCODE=IGNORE|WARN tells SAS whether to terminate execution if errors are encountered during the translation of languages and character sets.

Support for the DataFlux Integration Server

The following procedures and functions support DataFlux Integration Servers:

- PROC DQSRVADM creates job status data sets after querying DataFlux Integration Servers.
- PROC DQSRVSVC runs real-time services on DataFlux Integration Servers. The services are created with DataFlux dfPower Architect software.
- The DQSRVARCHJOB function runs jobs on DataFlux Integration Servers. The jobs are created with DataFlux dfPower Architect software.
- The DQSRVCOPYLOG and DQSRVDELETELOG functions manage log entries on DataFlux Integration Servers.
- The DQSRVJOBSTATUS function reads log entries from DataFlux Integration Servers.
- The DQSRVKILLJOB function terminates jobs that are running on DataFlux Integration Servers.
- The DQSRVPROFJOBFILE function runs profile jobs on individual files on DataFlux Integration Servers. The jobs are created with the DataFlux dfPower Profile software.
- The DQSRVPROFJOBREP function runs profile jobs on repositories.
- The DQSRVUSER function authenticates users on DataFlux Integration Servers.

In SAS 9.2 Phase 2, the following options were added for the DQSRVSVC procedure:

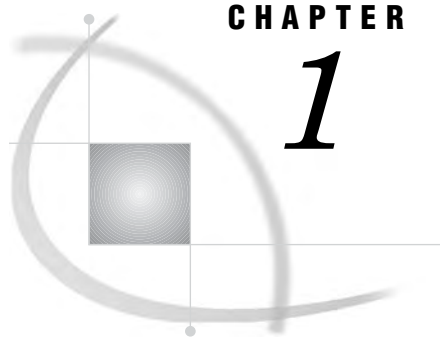
- MISSINGVARSOK enables the continuation of processing when an input data set is missing one or more variables.
- TRIM removes blank spaces from the ends of input data sets that are processed by real-time services executed on DataFlux Integration Servers.

In the third maintenance release for SAS 9.2, the DQSRVVER function enables you to determine the version of the DataFlux Integration Server. The following options were added to the DQSRVSVC procedure.

- MACROS enable a series of name-value pairs to be passed to a service as macros.
- NOPRINT suppresses writing the SERVICEINFO information to an output data set.
- SERVICEINFO lists the input and output columns used by a given service.

Support for the z/OS Operating Environment

In SAS 9.2 Phase 2 all of the language elements that access DataFlux Integration Servers are enabled in the z/OS operating environment.



CHAPTER

1

Overview SAS Data Quality Server

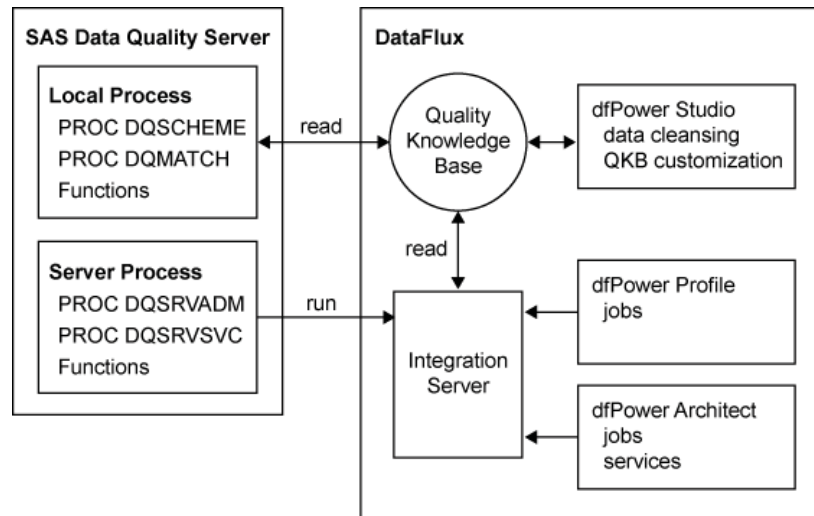
Overview 1
SAS Data Quality Server 1

Overview

SAS Data Quality Server

The SAS Data Quality Server consists of a Quality Knowledge Base (QKB) and SAS language elements. The language elements in the diagram are divided into two logical groups. The Local Process group and the Server Process group.

Figure 1.1 Server Interactions

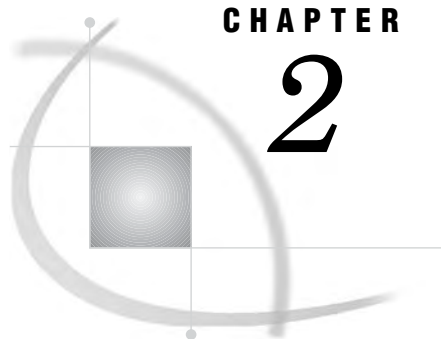


The DataFlux Integration Server and the related DataFlux dfPower Profile and DataFlux dfPower Architect applications are combined with the SAS Data Quality Server software in various software offerings.

You can also choose to add DataFlux software after purchasing the SAS Data Quality Server software.

A DataFlux Integration Server with a DataFlux Integration Server for SAS license accepts requests only from a SAS client for execution of a Web service. However, if you

have this license, DataFlux jobs can be executed by 3rd party products using the command-line interface that is available for the DataFlux Integration Server.



CHAPTER

2

Concepts

<i>SAS Data Quality Server Concepts</i>	3
<i>SAS Data Quality Setup File</i>	3
<i>Edit the SAS Data Quality Setup File</i>	4
<i>Configure Your SAS Session for Data Quality</i>	4
<i>Specify Definitions In SAS Data Cleansing Programs</i>	5
<i>Considerations for Installing and Updating the Software</i>	5
<i>DataFlux Jobs and Services</i>	5
<i>Running Jobs and Services on a DataFlux Integration Server</i>	6
<i>DataFlux Integration Server Passwords</i>	6
<i>Load and Unload Locales</i>	7
<i>Schemes</i>	8
<i>Create the Schemes</i>	8
<i>Analysis Data Sets</i>	8
<i>Applying Schemes</i>	9
<i>Meta Options</i>	9
<i>Create Match Codes</i>	11
<i>How Match Codes Are Created</i>	11
<i>Match Code Length</i>	12
<i>Clusters</i>	12
<i>Householding with the DQMATCH Procedure</i>	12
<i>Clustering with Exact Criteria</i>	13
<i>Sensitivity</i>	13

SAS Data Quality Server Concepts

SAS Data Quality Setup File

To access a Quality Knowledge Base, SAS programs reference path specifications in the setup file **dqsetup.txt**. The location of the setup file is specified with the **DQSETUPLOC=** system option. The value of the system option can be the path to **dqsetup.txt**, or a path to the root directory of a Quality Knowledge Base.

In the z/OS operating environment, the setup file **DQSETUP** is a member of a SAS Data Quality Configuration PDS.

Note: SAS programs running jobs and services on a DataFlux Integration Server do not reference the setup file. These programs do not directly access a Quality Knowledge Base. △

If you move your Quality Knowledge Base, update the path specifications with DQSETUPLOC= accordingly.

If your site uses multiple Quality Knowledge Bases, reference the intended setup file with the DQSETUPLOC= option.

Edit the SAS Data Quality Setup File

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

You can access the file in the following ways:

- specifying paths separated by commas to each individual directory

Example:

```
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\casetab;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\chopinfo;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\grammar;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\locale;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\phonetx;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\regexlib;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\scheme;
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A\vocab;
```

- specifying a path to the QKB-root directory

```
DISK C:\Program Files\DataFlux\QltyKB\CI\2009A;
```

If you edit the setup file, complete these tasks:

- Include a semicolon at the end of each fully qualified path.
- Specify the DISK access method.
- In the Windows and UNIX operating environments, do not change the last directory or filename in any path.
- For the z/OS operating environment, do not change the PDS names. Do not change the names of the PDS contents. For example, in the following entry in the setup file, you would retain the GRAMMAR PDS name and not change any member names inside that PDS:

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

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

You can insert blank lines into the setup file.

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

Configure Your SAS Session for Data Quality

Use the DQOPTIONS= system option to configure your SAS session for data quality. Specify the option in your SAS start-up command or in your SAS configuration file, SASV9.CFG. DQOPTIONS= enables you to specify one or more option-value pairs that pertain to the operation of the SAS Data Quality Server software.

The DQOPTIONS= system option currently enables two option-value pairs. The DQSRVPROTOCOL=WIRELINE pair improves the performance of the DQSRVSVC procedure by streamlining data transfer to and from the DataFlux Integration Server.

It is the default on z/OS, and improves performance on all other platforms. In other operating environments, the default SOAP protocol is recommended.

TRANSCODE=IGNORE | WARN specifies that transcoding errors between character sets are to be ignored, and SAS processing is allowed to continue. By default, transcoding errors terminate SAS processing.

See: “DQOPTIONS” on page 108.

Specify Definitions In SAS Data Cleansing Programs

To specify definitions in your SAS data cleansing programs, you need to know the names of the definitions that are available in a particular locale. Use the AUTOCALL macro %DQPUTLOC.

To display information about a locale that is currently loaded into memory, use the “DQLOCALEINFOGET Function” on page 69.

To display a list of definitions in a specified locale, use the “DQLOCALEINFOLIST Function” on page 69.

Use the “DQLOCALEGUESS Function” on page 68 to return the name of the locale that best fits your data.

Considerations for Installing and Updating the Software

The SAS Data Quality Server software is delivered with a sample Quality Knowledge Base. After you install the SAS Data Quality Server software, download the latest Quality Knowledge Base, including your choice of locales, from the DataFlux Web site www.dataflux.com.

To maximize performance, download new Quality Knowledge Bases as they are made available by DataFlux. On the DataFlux Web site, check the release notes of the latest release. Determine whether your locales have been updated. Decide whether you need any of the new locales that might have been added.

When you update your Quality Knowledge Base, you might want to install it in a new location rather than overwriting your existing Quality Knowledge Base. This decision is particularly important if you have customized your Quality Knowledge Base in the previous release. Customizations are made with the DataFlux dfPower Customize software. If you install your updated Quality Knowledge Base in a new location, either change your setup file or reference a new setup file in your SAS programs.

If you customized your previous Quality Knowledge Base, evaluate those changes and carry them over to your new Quality Knowledge Base as needed.

CAUTION:

When you upgrade your Quality Knowledge Base, be sure to regenerate your existing match codes so that they are consistent with the newly created match codes. Δ

DataFlux Jobs and Services

Jobs and services that are run on a DataFlux Integration Server fulfill separate needs. Use jobs to access larger data sets in batch mode, when a client application is not waiting for a response. Use services and small data sets in real time, when clients await a response from the server.

To create jobs and services for your DataFlux Integration Server, use the DataFlux dfPower Profile and DataFlux dfPower Architect applications. Create jobs to analyze

the quality of your data with DataFlux dfPower Profile software. Profile jobs and their output can be stored either in a file format or in the DataFlux unified repository. The DQSRVPROFJOBFILE function executes a profile job that has been stored as a file format job. The DQSRVPROFJOBREP function executes a profile job that has been stored in the repository.

Use DataFlux dfPower Architect software to create jobs and services using a drag-and-drop interface. You can trigger the DataFlux dfPower Architect jobs with the function DQSRVARCHJOB. You can run DataFlux dfPower Architect services with PROC DQSRVSVC.

Jobs and services that run on the DataFlux Integration Servers generate information that is written to the log. You can read the server logs using the DQSRVADM procedure. Based on the information returned by the DQSRVSTATUS function, you can terminate jobs using the DQSRVKILLJOB function.

Running Jobs and Services on a DataFlux Integration Server

Follow these steps to run jobs and services on a DataFlux Integration Server.

- 1 In the z/OS operating environment, for DataFlux Integration Servers version 8.1.1 or newer, configure your DataFlux Integration Server to use the Wireline protocol. This is described in the *DataFlux Integration Server: User's Guide*. The Wireline protocol improves data transfer performance.
- 2 In the z/OS operating environment, reconfigure your SAS session to use the Wireline protocol, as described in *Configure Your SAS Session for Data Quality* section, earlier in this chapter.
- 3 Create jobs and services using the DataFlux dfPower Profile and DataFlux dfPower Architect software.
- 4 Upload the jobs to the DataFlux Integration Server using the DataFlux Integration Server Manager.
- 5 Create and run the SAS programs that execute or trigger the jobs and services on the DataFlux Integration Server.

To run jobs and services, you do not need to load a Quality Knowledge Base onto your local host. The DataFlux Integration Server handles all interactions with your Quality Knowledge Bases.

See: Chapter 6, “The DQSRVADM Procedure,” on page 45 and Chapter 7, “The DQSRVSVC Procedure,” on page 49.

DataFlux Integration Server Passwords

If security has been implemented on your DataFlux Integration Server, include user names and passwords in the procedures and function calls that access that server. Specify the passwords directly, in plain text, or as encoded passwords. SAS recognizes encoded passwords and decodes them before it sends the passwords to the DataFlux Integration Server.

The following example shows how to encode a password and use that password in a call to the DQSRVSVC procedure:

```
/* Encode password in file. */
filename pwfile 'c:\dataEntry01Pwfile';
proc pwencode in='0e3s2m5' out=pwfile;
run;

/* Load encoded password into macro variable. */
```

```

data _null_;
  infile pwfile obs=1 length=1;
  input @;
  input @1 line $varying1024. 1;
  call symput ('dbpass', substr(line,1,1));
run;

/* Run service on secure DataFlux Integration Server */
proc dqsrsvvc
  service='cleanseCorpName' host='entryServer1'
  userid='DataEntry1'          password="&dbpass"
  data=corpName                out=corpNameClean;
run;

```

PROC PWENCODE concepts, syntax, and examples are documented in the *Base SAS Procedures Guide*.

Load and Unload Locales

You need to load and unload locales in order to run data-cleansing programs in SAS. Conversely, you do not need to load locales if your SAS programs run jobs and services on a DataFlux Integration Server.

Before you run data-cleansing programs in SAS, load locales into memory using the AUTOCALL macro %DQLOAD. The macro sets the value of the system options DQSETUPLOC and DQLOCALE. The macro also loads the specified locales into local memory. The DQSETUPLOC= option specifies the location of the setup file. The DQLOAD= option specifies an ordered list of locales.

See: “%DQLOAD AUTOCALL Macro” on page 53.

The order of locales in the locale list is pertinent only when one of the following conditions is true:

- A locale is not specified by name.
- The specified locale is not loaded into memory.
- Input data is insufficient for the DQLOCALEGUESS function.

If a locale cannot be established, SAS searches the list of locales. SAS references the first definition it finds that has the specified name. Use the “DQLOCALEGUESS Function” on page 68, to determine the best locale for that data.

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

If you change locale files in the Quality Knowledge Base using the DataFlux dfPower Customize software, you must reload macros into memory with the %DQLOAD macro before cleansing data.

After you submit your data-cleansing programs, you can unload the locale from memory by using the “%DQUNLOAD AUTOCALL Macro” on page 55.

New locales, and updates to existing locales, are provided periodically by DataFlux in the form of a new Quality Knowledge Base, which you can download from the following Web address:

www.dataflux.com/QKB

Schemes

A scheme is file that you create to transform the values of a character variable. Applying the scheme to the data, transforms similar representations of a data value into a standard representation.

To create and apply multiple schemes in a single procedure, see Chapter 5, “The DQSCHEME Procedure,” on page 33. Use “DQSCHEMEAPPLY Function” on page 88 and “DQSCHEMEAPPLY CALL Routine” on page 84 to apply schemes as well.

This only pertains to schemes that are BFD (DataFlux) type of schemes. The Scheme Builder application does not recognize schemes that are stored as SAS data sets.

Create the Schemes

Schemes are created with the CREATE statement in the DQSCHEME procedure. The CREATE statement uses the matching technology, behind the scenes, to effectively group like data values together. A survivor is selected out of each group to be the standard value for that group of data values. The survivor is selected based on highest frequency of occurrence of the data values.

Note: During scheme creation, the DQSCHEME procedure evaluates the definition of each input variable in each CREATE statement. An error message is generated if the defined length of an input variable exceeds 1024 bytes. \triangle

Scheme data sets are created in SAS format or in Blue Fusion Data format. Blue Fusion Data (BFD) format is recognized by SAS and by DataFlux dfPower Studio software.

- The SAS Data Quality Server software can create and apply schemes. You can also view the schemes with the SAS table viewer.
- DataFlux dfPower Studio software can create, apply, and edit schemes in BFD format only.
- In the z/OS operating environment, the SAS Data Quality Server software can create, apply, and display schemes in SAS format. Schemes in Blue Fusion format can be applied.

Note: There is a CONVERT statement that is used to convert schemes between the two formats. \triangle

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 best suits your data and your goal. You can create a series of analysis data sets using different sensitivity values to compare the results. Changing the sensitivity value changes the clustering of input values, as described in “Sensitivity” on page 13.

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 “Create the Schemes” on page 8.

Applying Schemes

After you create a scheme data set, apply it to an input variable to transform its values. You can apply a scheme with the APPLY statement in the DQSCHEME procedure (see “APPLY Statement” on page 35), or with the DQSCHEMEAPPLY function or CALL routine. Use the DQSCHEMEAPPLY CALL routine if you want to return the number of transformations that occurred during the application of the scheme. See “DQSCHEMEAPPLY Function” on page 88.

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

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

The lookup method determines how the input value is matched to the DATA values in the scheme. The SCHEME_LOOKUP option or argument specifies that the match must be exact, although case is insensitive. Alternatively the match can consist of a match between the match codes of the input value, and the match codes of the DATA values. When a match occurs, any adjacent blank spaces in the transformation value are replaced with single blank spaces. Then the value is written into the output data set.

If no match is found for an input value, that exact value is written into the output data set.

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

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

The file format of a scheme is important when that scheme is applied. In the z/OS operating environment, schemes must be created and applied in SAS format. Schemes that are stored in a PDS in BFD format can be applied. Schemes in BFD format can be converted to SAS format using the CONVERT statement in the DQSCHEME procedure. Before you apply a scheme, see “Applying Schemes” on page 9.

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

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.

The meta options for the match definition and sensitivity value are valid only when the scheme is applied with match-code lookup, when the value of the SCHEME_LOOKUP option is USE_MATCHDEF.

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

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

The meta options are stored using the following syntax:

'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. For an input value to be transformed, that value must exactly match a DATA value in the scheme.

IC specifies SCHEME_LOOKUP=IGNORE_CASE.

UM specifies that SCHEME_LOOKUP=USE_MATCHDEF. Match codes 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

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

Valid values range from 50 to 95.

match-definition

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

For example, the following meta options string 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

'UM' 'P' '80' 'NAME'

Create Match Codes

Match codes are encoded representations of character values that are used for analysis, transformation, and standardization of data. Match codes are created by the following procedures and functions:

The DQMATCH procedure creates match codes for one or more variables or parsed tokens that have been extracted from a variable. The procedure can also assign cluster numbers to values with identical match codes.

See “Syntax: DQMATCH Procedure ” on page 20

The “DQMATCH Function” on page 71 generates match codes for a variable.

The “DQMATCHPARSED Function” on page 73 generates match codes for tokens that have been parsed from a variable.

Match codes are created by the DQMATCH procedure and by the DQMATCH and DQMATCHPARSED functions. The functions DQMATCH and DQMATCHPARSED return one match code for one input character variable. With these tools you can create match codes for an entire character value or a parsed token extracted from a character value.

During processing, match codes are generated according to the specified locale, match definition, and sensitivity-level.

The locale identifies the language and geographical region of the source data. For example, the locale ENUSA specifies that the source data uses the English language as it is used in the United States of America.

The match definition in the Quality Knowledge Base identifies the category of the data and determines the content of the match codes. Examples of match definitions are named ADDRESS, ORGANIZATION, and DATE(YMD).

To determine the match definitions that are available in a Quality Knowledge Base, consult the QKB documentation from DataFlux (a SAS company). Alternatively, use the DQLOCALEINFOLIST function to return the names of the locale’s match definitions. Use the DQLOCALEINFOLIST function if your site modifies the default Quality Knowledge Base using DataFlux dfPower Customize software.

The sensitivity level is a value between 0 and 99 that determines the amount of information that is captured in the match code, as described in “Sensitivity” on page 13.

If two or more match codes are identical, a cluster number can be assigned to a specified variable, as described in “Clusters” on page 12.

The content of the output data set is determined by option values. You can include values that generate unique match codes and you can include and add a cluster number to blank or missing values. You can also concatenate multiple match codes.

Match codes are also generated internally when you create a scheme with the DQSCHEME procedure, as described in “Schemes” on page 8. Match codes are also created internally by the DQSCHEMEAPPLY function, and the DQSCHEMEAPPLY CALL routine. The match codes are used in the process of creating or applying a scheme.

How Match Codes Are Created

You can create two types of match codes:

- *Simple match codes* from a single input character variable.

- *Composite match codes* a concatenation of match codes from two or more input character variables. The separate match codes are concatenated into a composite match code.

Use the DELIMITER= option to specify that a delimiter exclamation point (!), is to be inserted between the simple match codes in the combined match code.

To create simple match codes, specify one CRITERIA statement, one input variable identified in the VAR= option, and one output variable identified with the MATCHCODE= option.

Composite match codes are similar, except that you specify multiple CRITERIA statements for multiple variables. All the CRITERIA statements specify the same output variable in their respective MATCHCODE= options.

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

- 1 Parse the input character value to identify tokens.
- 2 Remove insignificant words.
- 3 Remove some of the vowels. Remove fewer vowels when a scheme-build match definition has been specified. See “Scheme Build Match Definitions” on page 16.
- 4 Standardize the format and capitalization of words.
- 5 Create the match code by extracting the appropriate amount of information from one or more tokens, based on the specified match definition and level of sensitivity.

Certain match definitions skip some of these steps.

Note: To analyze or join two or more data sets using match codes, create the match codes in each data set with identical sensitivity levels and match definitions. Δ

Match Code Length

Match codes can vary in length between 1 and 1024 bytes. The length is determined by the specified match definition. If you receive a message in the SAS log that states that match codes have been truncated, extend the length of the match code variable. Truncated match codes do not produce accurate results.

Clusters

Clusters are numbered groups of values that generate identical match codes or that have an exact match of characters. Clusters are used in the creation of schemes using the DQSCHHEME procedure. The cluster with the greatest number of members becomes the transformation value for the scheme.

Householding with the DQMATCH Procedure

You can use the DQMATCH procedure to generate cluster numbers as it generates match codes. An important application for clustering in is commonly referred to as householding. Members of a family or household are identified in clusters that are based on multiple criteria and conditions.

To establish the criteria and conditions for householding, use multiple CRITERIA statements and CONDITION= options within those statements.

- The integer values of the CONDITION= options are reused across multiple CRITERIA statements to establish groups of criteria.
- Within each group, match codes are created for each criteria.

- If a source row is to receive a cluster number, all of the match codes in the group must match all of the codes in another source row.
- The match codes within a group are therefore evaluated with a logical AND.

If more than one condition number is specified across multiple CRITERIA statements, there are multiple groups, and multiple groups of match codes. In this case, source rows receive cluster numbers when any of its groups matches any other group in another source row. The groups are therefore evaluated with a logical OR.

For an example of householding, assume that a data set that contains customer information. To assign cluster numbers you use two groups of two CRITERIA statements. One group (condition 1) uses two CRITERIA statements to generate match codes based on the names of individuals and an address. The other group (condition 2) generates match codes based on organization name and address. A cluster number is assigned to a source row when either pair of match codes matches at least one group matches the match codes from another source row. The code and output for this example is provided in Example 5 on page 29.

Clustering with Exact Criteria

Use the EXACT= option of the DQMATCH procedure's CRITERIA statement to use exact character matches as part of your clustering criteria. Exact character matches are helpful in situations where you want to assign cluster numbers using a logical AND of an exact number and the match codes of a character variable.

For example, you could assign cluster numbers using two criteria, one using an exact match on a customer ID values. The other using a match code generated from customer names. The syntax of the EXACT= option is provided in "CRITERIA Statement" on page 22.

Sensitivity

The amount of information contained in match codes is determined by a specified sensitivity level. Changing the sensitivity level enables you to change what is considered a match. Match codes created at lower levels of sensitivity capture little information about the input values. The result is more matches, fewer clusters, and more values in each cluster. See "Clusters" on page 12.

Higher sensitivity levels require that input values are more similar to receive the same match code. Clusters are more numerous, and each cluster contains fewer entries. For example, when collecting customer data based on account numbers, cluster on account numbers, with a high sensitivity value.

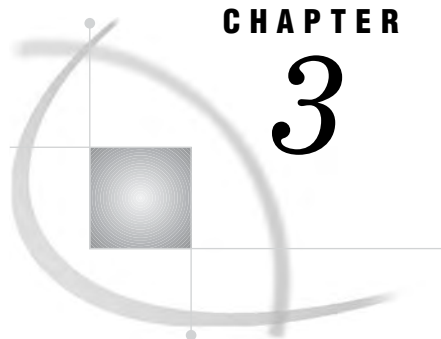
In some data cleansing jobs, a lower sensitivity value is needed. To transform the following names to one consistent value using a scheme, specify a lower sensitivity level.

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

All four values are assigned to the same cluster. The clusters are transformed to the most common value, **Patty Fielding**.

Sensitivity values range from 50 to 95. The default value is 85.

To arrive at the sensitivity level that fits your data and your application, test with the DQMATCH procedure. Alternatively create analysis data sets with the DQSCHEME procedure.



CHAPTER

3

Locale Definitions

<i>Locale Definitions</i>	15
<i>Parse Definitions</i>	15
<i>Global Parse Definitions</i>	16
<i>Match Definitions</i>	16
<i>Scheme Build Match Definitions</i>	16
<i>Case and Standardization Definitions</i>	17
<i>Standardization of Dates in the EN Locale</i>	17
<i>Gender Analysis, Locale Guess, and Identification Definitions</i>	18
<i>Pattern Analysis Definitions</i>	18

Locale Definitions

Parse Definitions

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

Parse definitions and tokens are referenced by the following functions:

- “DQPARSE Function” on page 76
- “DQPARSEINFOGET Function” on page 77
- “DQTOKEN Function” on page 103
- “DQPARSETOKENGET Function” on page 81
- “DQPARSETOKENPUT Function” on page 82

For a brief example of how tokens are assigned and used, see “Specify Definitions In SAS Data Cleansing Programs” on page 5.

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

```
NAME PREFIX    empty
GIVEN NAME    Ian
MIDDLE NAME   M.
```

FAMILY NAME	Banks
NAME SUFFIX	empty
NAME	empty
APPENDAGE	

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. Δ

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.

Match Definitions

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

When you create match codes, you determine the number of clusters (values with the same match code) and the number of members in each cluster by specifying a sensitivity level. The default sensitivity level is specified by the procedure or function, rather than the match definition. For information about sensitivity levels, see “Sensitivity” on page 13.

Match definitions are referenced by the following procedures and functions:

- Chapter 4, “The DQMATCH Procedure,” on page 19
- Chapter 5, “The DQSCHEME Procedure,” on page 33
- “DQMATCH Function” on page 71
- “DQMATCHINFOGET Function” on page 72
- “DQMATCHPARSED Function” on page 73

When you create match codes for parsed character values, your choice of match definition depends on the parse definition that was used to parse the input character value. To determine the parse definition that is associated with a given match definition, use the “DQMATCHINFOGET Function” on page 72.

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

Scheme Build Match Definitions

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

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

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

Case and Standardization Definitions

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

Case definitions are referenced by the “Functions: DQCASE Function” on page 63. Standardization definitions are referenced by the “DQSTANDARDIZE Function” on page 102.

Case definitions transform the capitalization of character values. For example, the case definition Proper in the ENUSA locale takes as input any general text. It 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 (FULL NAME) in the locale ENUSA converts abbreviated state names to full names in uppercase.

Standardization of Dates in the EN Locale

In the EN locale, dates are standardized to two-digit days (00–31), two-digit months (01–12), and four-digit years. 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 3.1 Examples of Date Standardizations

Input Date	Standardization Definition	Standardized Date
July04, 03	Date (MDY)	07 04 2003
July 04 04	Date (MDY)	07 04 1904
July0401	Date (MDY)	07 04 2001
04.07.02	Date (DMY)	04 07 2002
04-07-2004	Date (DMY)	04 07 2004
03/07/04	Date (YMD)	2003 07 04

Two-digit year values are standardized as follows:

- If an input year is greater than 00 and less than or equal to 03, the standardized year 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=.

Gender Analysis, Locale Guess, and Identification Definitions

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

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

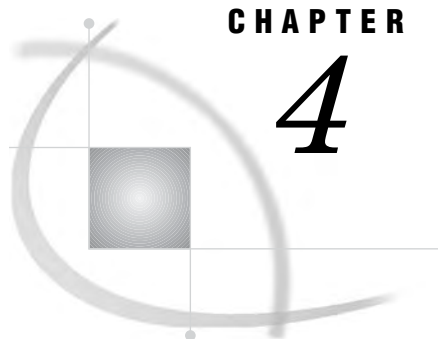
Gender analysis definitions determine the gender of an individual based on that individual's name. The gender is determined to be unknown, if the first name is used by both males and females. If no other clues are provided in the name, or if conflicting clues are found, gender analysis definitions are referenced by the "DQGENDER Function" on page 64.

Locale guess definitions allow the software to determine the locale that is most likely represented by a character value. All locales that are loaded into memory as part of the locale list are considered, but only if they contain the specified guess definition. If a definite locale determination cannot be made, the chosen locale is the first locale in the locale list. Locale guess definitions are referenced by the "DQLOCALEGUESS Function" on page 68.

Identification definitions are used to categorize character values. 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" on page 67.

Pattern Analysis Definitions

Pattern analysis definitions enable you to determine whether an input character value contains characters that are alphabetic, numeric, non-alphanumeric (punctuation marks or symbols), or a mixture of alphanumeric and non-alphanumeric. The ENUSA locale contains two pattern analysis definitions: The pattern analysis definition WORD is referenced by the DQPATTERN function. To generate one character of analytical information for each word in the input character value. See "DQPATTERN Function" on page 83. The CHARACTER definition generates one character of analytical information for each character in the input character value.



CHAPTER

4

The DQMATCH Procedure

<i>Overview: DQMATCH Procedure</i>	19
<i>What Does the DQMATCH Procedure Do?</i>	19
<i>Syntax: DQMATCH Procedure</i>	20
<i>PROC DQMATCH Statement</i>	20
<i>CRITERIA Statement</i>	22
<i>DQMATCH Examples</i>	24
<i>Example 1: Generate Composite Match Codes</i>	24
<i>Example 2: Matching Values Using Mixed Sensitivity Levels</i>	25
<i>Example 3: Matching Values Using Minimal Sensitivity</i>	27
<i>Example 4: Creating Match Codes for Parsed Values</i>	28
<i>Example 5: Clustering with Multiple CRITERIA Statements</i>	29
<i>Example 6: Generating Multiple Simple Match Codes</i>	30

Overview: DQMATCH Procedure

What Does the DQMATCH Procedure Do?

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

The DQMATCH procedure can generate cluster numbers for input values that generate identical *match-codes*. Cluster numbers are not assigned to input values that generate unique *match-codes*. 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. Blank values can receive a cluster number.

A specified sensitivity level determines the amount of information in the *match-codes*. 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 *sensitivity-levels* 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

Requirements: At least one CRITERIA statement is required.

Note: Match-codes are an encoded version of a character value that is created as a basis for data analysis and data cleansing. Match-codes are used to cluster and compare character values. The DQMATCH procedure generates match-codes based on a definition and sensitivity value.

```
PROC DQMATCH DATA=<input-data-set>
  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>;

CRITERIA <option(s)>;
```

PROC DQMATCH Statement

```
PROC DQMATCH <option(s)>;
```

Options

CLUSTER=*variable-name*

specifies the name of the numeric variable in the output data set that contains the cluster number.

Note: 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.

Default: CLUSTER_BLANKS

CLUSTER_BLANKS

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

NO_CLUSTER_BLANKS

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

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=*data-set-name*

specifies then 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.

SAS Default: uses a delimiter.

DataFlux dfPower Studio Default: does not use a delimiter.

Note: Be sure to used delimiters consistently if you plan to analyze, compare, or combine match codes created in SAS and in DataFlux dfPower Studio.

DELIMITER

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

NODELIMITER

specifies that multiple *match-codes* are concatenated without exclamation point delimiters.

LOCALE=*locale-name*

specifies the name of the locale that is used to create *match-codes*. 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.

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=*character-variable*

specifies the name of the output character variable that stores the match codes. The DQMATCH procedure defines a sufficient length for this variable, even if a variable with the same name 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.

OUT=*output-data-set*

specifies the name of the output data set for *match-codes* created with the DQMATCH procedure. The DQMATCH procedure creates *match-codes* 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 match codes and optional cluster numbers for an input variable.

Requirement: At least one CRITERIA statement is required in DQMATCH procedures.

```
CRITERIA CONDITION=<integer>
  DELIMSTR=<variable-name> | VAR=<variable-name>
  EXACT | MATCHDEF
  MATCHCODE=<output-character-variable>
  SENSITIVITY=<sensitivity-level>;
```

Options

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, SAS generates an error. To prevent the error, specify the MATCHCODE= option in CRITERIA statements only.

Note: If you have not assigned a value to the CLUSTER= option in the DQMATCH procedure, cluster numbers are assigned to a variable named **CLUSTER** by default.

See: The “DQMATCHINFOGET Function” on page 72

DELIMSTR | VAR

specifies the name of a variable.

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

See: The “DQPARSE Function” on page 76 and the “DQPARSETOKENPUT Function” on page 82.

DELIMSTR=*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=*variable-name*

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

Restriction: The values of this variable cannot contain delimiters added with the DQPARSE function or the DQPARSETOKENPUT function.

EXACT | MATCHDEF

assigns a cluster number.

Default: If the CLUSTER= option has not been assigned a variable in the DQMATCH procedure, then cluster numbers are assigned to the variable named CLUSTER.

Restriction: If you specify the MATCHCODE= option in the DQMATCH procedure, the *match-code* is a composite of the exact *character-value* and the *match-code* that is generated by the *match-definition*.

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 *match-code* for the specified variable.

Restriction: The *match-definition* must exist in the locale that is specified in the LOCALE= option of the DQMATCH procedure.

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

MATCHCODE=*character-variable*

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

Restriction: The MATCHCODE= option is not valid if you also specify the MATCHCODE= option in the DQMATCH procedure.

Restriction: If you are using multiple CRITERIA statements in a single procedure step, either:

- specify the MATCHCODE=*character-variable* in each CRITERIA statement
- or generate composite *match-codes* by specifying the MATCHCODE= option only in the DQMATCH procedure.

SENSITIVITY=*sensitivity-level*

determines the amount of information in the resulting match codes. Higher sensitivity values create match codes that contain more information about the input values. Higher sensitivity levels result in a greater number of clusters, with fewer values in each cluster.

Default: The default value is 85.

Valid values: Valid values range from 50 to 95.

Details

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

Simple *match-codes* are created when the CRITERIA statements specify different values for their respective MATCHCODE= options. Composite match codes are created

when two or more CRITERIA statements specify the same value for their respective MATCHCODE= options.

To create match codes for a parsed character variable, specify the DELIMSTR= option instead of the VAR= option. 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.

DQMATCH Examples

Example 1: Generate Composite Match Codes

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

```

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

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

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

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

```


Display 4.1 PROC Print Output

	customer	address	MATCH_CD	CLUSTERGRP
1	Paul Becker	392 N. Main St. PO Box 7720	M3Y\$\$\$\$NW\$\$\$\$\$!K-H\$\$BP\$\$IH0\$\$	
2	Mr. Raul Becker	392 North Main St.	M3Y\$\$\$\$YW\$\$\$\$\$!K-H\$\$BP\$\$\$\$\$\$\$\$	
3	Bobby Becket	392 Main St.	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$\$\$\$\$\$\$	
4	Mr. Robert E Beckett	392 South Main Street #2270	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$HHI0\$\$	1
5	Rob Beckett	392 S. Main St. PO Box 2270	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$HHI0\$\$	1
6	Mr. Robert J. Beckett	P. O. Box 2270 392 S. Main St.	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$HHI0\$\$	1
7	Bob Beckett	392 S. Main St. PO Box 2270	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$HHI0\$\$	1
8	Robert E. Beckett	392 S. Main St. PO Box 2270	M3~\$\$\$\$M@M\$\$\$\$\$!K-H\$\$BP\$\$HHI0\$\$	1

The output data set, OUT_DB1, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the 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 match codes, the address difference caused it to be excluded in Cluster 1.

Example 2 on page 25 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. Δ

Example 2: Matching Values Using Mixed Sensitivity Levels

The following example is similar to Example 1 on page 24, in that it displays match codes and clusters for a simple data set. This example differs in that the CRITERIA statement for the ADDRESS variable uses a sensitivity of 50. The CRITERIA statement for the NAME variable uses the same default sensitivity of 85.

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

This example primarily shows how to identify possible duplicate customers based on their 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. Becket  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;

```

Display 4.2 PROC Print Output

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

The output data set, OUT_DB2, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code 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 on page 24, 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. Δ

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;

```

Display 4.3 PROC Print Output

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

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

A higher sensitivity level would not cluster these observations.

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

Example 4: Creating Match Codes for Parsed Values

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

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

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

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

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

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

```

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

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

```

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

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;

```

The output is as follows:

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

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. Δ

Example 6: Generating Multiple Simple Match Codes

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 match codes by specifying the MATCHCODE= option on each CRITERIA statement. In addition, unlike the first example, which creates a cluster number, you cannot create a cluster number when generating multiple simple match codes.

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;

```

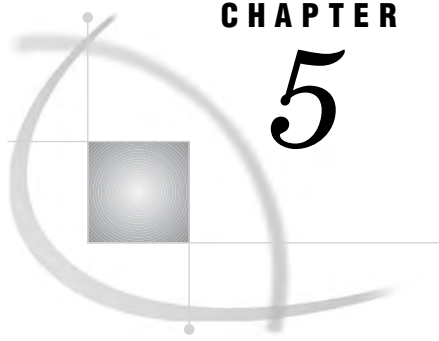
```
/* Run the DQMATCH procedure. */  
proc dqmatch data=cust_db out=out_db5 locale='ENUSA';  
  criteria matchdef='Name' var=customer matchcode=mc_name;  
  criteria matchdef='Address' var=address matchcode=mc_addr;  
run;  
  
/* Print the results. */  
proc print data=out_db5;  
run;
```

The output data set, OUT_DB5, includes the new variables MC_NAME and MC_ADDR. Compare this to the result of 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 match codes depends on the type of comparison that you need. If you want to compare names and addresses separately, generate separate match codes 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 on page 24 to compare the examples.

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



CHAPTER

5

The DQSCHHEME Procedure

<i>Overview: DQSCHHEME Procedure</i>	33
<i>What Does the DQSCHHEME Procedure Do?</i>	33
<i>Syntax: DQSCHHEME Procedure</i>	34
<i>PROC DQSCHHEME Statement</i>	34
<i>APPLY Statement</i>	35
<i>CONVERT Statement</i>	36
<i>CREATE Statement</i>	37
<i>PROC DQSCHHEME Examples</i>	40
<i>Example 1: Creating an Analysis Data Set</i>	40
<i>Example 2: Creating Schemes</i>	41
<i>Example 3: Creating BFD Schemes</i>	42
<i>Example 4: Applying Schemes</i>	42

Overview: DQSCHHEME Procedure


What Does the DQSCHHEME Procedure Do?

PROC DQSCHHEME creates scheme data sets and analysis data sets and applies schemes to input data sets. You can also apply schemes with the DQSCHHEMEAPPLY function or CALL routine.

See “DQSCHHEMEAPPLY CALL Routine” on page 84

The DQSCHHEME 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 5.1 Transform Similar Data Values

Input Data		Output Data
Robert T. Green		Robert T. Green
Robert Green		Robert T. Green
Robert Thomas Green		Robert T. Green
Robert T. Green	Apply Scheme	Robert T. Green
Rob Greene		Robert T. Green
Ryan T. Green		Ryan T. Green
Robert W. Green		Robert W. Green

The DQSCHHEME procedure also analyzes and reports on the quality of your data. See Chapter 5, “The DQSCHHEME Procedure,” on page 33.

Syntax: DQSCHHEME Procedure

```
PROC DQSCHHEME DATA=<input-data-set>
  BFD | NOBFD
  OUT=<output-data-set> ;
APPLY<option(s)>;
CONVERT<option(s)>;
CREATE<option(s)>;
```

PROC DQSCHHEME Statement

```
PROC DQSCHHEME <option(s)>
```

Options

BFD | NOBFD

specifies the format of the scheme.

BFD

specifying BFD indicates that all schemes are in Blue Fusion Data format.

NOBFD

specifying NOBFD indicates that all schemes are in SAS format.

The DQSCHHEME procedure can create and apply schemes in either format. Schemes in BFD format can be edited using the feature-rich graphical user interface of the DataFlux dfPower Studio software.

Default: BFD

Restriction: 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.

Restriction: Always specify NOBFD when creating schemes in the z/OS operating environment.

See: “Meta Options” on page 9

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* against which the schemes are applied.

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

OUT=*output-data-set*

specifies the *output-data-set*. 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. Results are written to the output data set after all schemes have been applied. If you specify OUT= without any APPLY statements an empty output data set is created.

APPLY Statement

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

See also: “Applying Schemes” on page 9

```

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

```

Options

LOCALE=*locale-name*

specifies the name of the match definition, in the specified locales, that is used to create match codes running the application of the scheme.

MATCH-DEFINITION=*match-definition*

specifies the name of the match definition, in the specified locales, that is used to create match codes running the application of the scheme.

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. See “Applying Schemes” on page 9.

ELEMENT

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

SCHEME=*scheme-name*

identifies the scheme to apply to the input data set. In all the operating environments other than z/OS, schemes using BFD format are identified by specifying a fileref or a fully qualified name that ends in `.sch.bfd`

SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF

specifies the method of applying the scheme to the data values of the input variable.

SENSITIVITY=*sensitivity-level*

defines the amount of information in the resulting match codes.

VAR=*variable-name*

specifies the name of the variable that is analyzed and transformed.

CONVERT Statement

Converts schemes between SAS and BFD formats.

Required: All options are required.

See also: “Applying Schemes” on page 9.

CONVERT BFDTOSAS | SASTOBFD

IN=*input data-set*

OUT=*output data-set*;

Arguments**BFDTOSAS | SASTOBFD**

specify BFDTOSAS to convert a scheme in Blue Fusion Data format to SAS format. Specify SASTOBFD to convert a scheme in SAS format to Blue Fusion Data format. Schemes with SAS format are created with the CREATE statement using the NOBFD option in the DQSCHHEME procedure.

CAUTION:

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

IN=*scheme-data-set*

identifies the existing scheme data set that is to be converted.

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

If SASTOBFD 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.

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

If SASTOBFD is specified, the value must be the name of a fileref. This fileref references a fully qualified path in lowercase that ends in **.sch.bfd**.

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

CREATE Statement

Creates a scheme or an analysis data set.

See also: “Applying Schemes” on page 9

```

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=<variable-name>;

```

Options

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.

See: “Create the Schemes” on page 8

INCLUDE_ALL

specifies that the scheme is to contain all of the values of the input variable. This includes input variables:

- with unique match codes
- 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 locale that contains the specified match definition. The value can be a locale name in quotation marks. It can be the name of a variable whose value is a locale name, or is an expression that evaluates to a locale name.

The specified locale must be loaded into memory as part of the locale list.

Default: The first locale in the locale list.

Restriction: If no value is specified, the default locale is used.

See: “Load and Unload Locales” on page 7

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 is superseded by match definitions specified in the APPLY statement or the DQSCHEMEAPPLY function or CALL routine.

Best Practice: 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 9

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. See “Applying Schemes” on page 9

ELEMENT

specifies that each element in each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP=USE_MATCHDEF, the match code for each element is compared to match codes 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 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 match codes 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.bfd**. Lowercase letters are required. To create a scheme data set in Blue Fusion Data format, specify the BFD option in the DQSCHEME procedure.

CAUTION:

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

To create a scheme in SAS format, specify the NOBFD 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: DQSCHEME Procedure” on page 34

SCHEME_LOOKUP= EXACT | IGNORE_CASE | USE_MATCHDEF

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

EXACT

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

IGNORE_CASE

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

USE_MATCHDEF

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

Specifying USE_MATCHDEF enables the options LOCALE=, MATCHDEF=, and SENSITIVITY=, which can be used to override the default values that might be stored 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 is 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 9

SENSITIVITY=*sensitivity-level*

determines the amount of information that is included in the match codes that are generated during the creation and perhaps the application of the scheme.

Higher sensitivity values generate match codes that contain more information. These match codes generally result in:

- fewer matches
- greater number of clusters
- fewer values in each cluster

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. This meta option is used at apply time only when SCHEME_LOOKUP=MATCHDEF. The default value supplied by this meta option is superseded by a sensitivity value specified in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine.

Default: 85

See: “Meta Options” on page 9

Valid values: 50 to 95

VAR=*input-character-variable*

specifies the input character variable that is analyzed and transformed. The maximum length of input values is 1024 bytes.

PROC DQSCHEME Examples

Example 1: Creating an Analysis Data Set

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. Δ

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

```

/* 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;

```


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. Δ

Example 2: Creating Schemes

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

```

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

/* See Example 1: 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 nobfd;
  create matchdef='City (Scheme Build)' var=city
    scheme=city_scheme locale='ENUSA';
  create matchdef='State (Scheme Build)' var=state
    scheme=state_scheme locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
    var=company scheme=org_scheme locale='ENUSA';
run;

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

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

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

```

Notice that 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. Δ

Example 3: Creating BFD Schemes

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

```

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

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

/* See Example 1: 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 bfd;
  create matchdef='City (Scheme Build)'
    var=city scheme=city locale='ENUSA';
  create matchdef='State (Scheme Build)'
    var=state scheme=state locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
    var=company scheme=org locale='ENUSA';
run;

```

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

Example 4: Applying Schemes

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

```

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

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

/* See Example 1: 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 out=vendors_out bfd;
  create matchdef='City (Scheme Build)'
    var=city scheme=city_scheme locale='ENUSA';
  create matchdef='State (Scheme Build)'
    var=state scheme=state_scheme locale='ENUSA';
  create matchdef='Organization (Scheme Build)'
    var=company scheme=org_scheme locale='ENUSA';
  apply var=city scheme=city_scheme;
  apply var=state scheme=state_scheme;
  apply var=company scheme=org_scheme;
run;

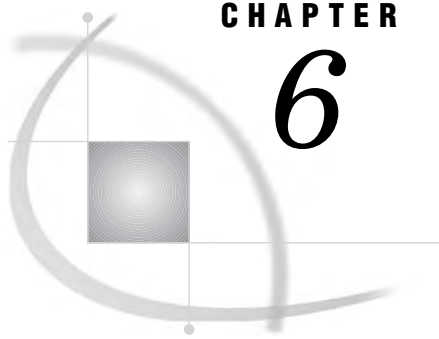
title 'Result after Applying all Three SAS Format Schemes';
proc print data=work.vendors_out;
run;

```

Note that the APPLY statements do not specify a locale. Nor do they specify the scheme lookup method using the SCHEME_LOOKUP= option. Because neither the locale nor the lookup method is specified, the schemes are applied with the ENUSA locale. The ENUSA locale is stored in the schemes.

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. Using the default scheme apply mode MODE=PHRASE is used, each input value is compared to the DATA values in the scheme.

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



CHAPTER

6

The DQSRVADM Procedure

<i>Overview: DQSRVADM Procedure</i>	45
<i>What Does the DQSRVADM Procedure Do?</i>	45
<i>Syntax: DQSRVADM Procedure</i>	45
<i>PROC DQSRVADM Statement</i>	45
<i>The Job Status Data Set</i>	46
<i>Security</i>	46
<i>PROC DQSRVADM Examples</i>	47
<i>Example 1: Generate a Job Status Data Set</i>	47
<i>Example 2: Clean Up Jobs and Logs</i>	47

Overview: DQSRVADM Procedure

What Does the DQSRVADM Procedure Do?

The DQSRVADM procedure creates a data set that provides the name, type, and description of all DataFlux dfPower Architect and DataFlux dfPower Profile jobs. This includes jobs that ran or that are running on a specified port on a DataFlux Integration Server. Status information is provided for all jobs that have a log file on the server.

Syntax: DQSRVADM Procedure

```
PROC DQSRVADM option(s);  
    OUT=output-data-set  
    PASSWORD=password  
    PORT=job-port-number  
    USERID=userid  
;
```

PROC DQSRVADM Statement

```
PROC DQSRVADM <option(s)>;
```

Options

HOST=host-name

identifies the host of the DataFlux Integration Server.

Default: *localhost*

OUT=output-data-set

specifies the storage location of the job status data set.

PASSWORD=password

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

PORT=port-number

identifies the port through which the local host communicates with the DataFlux Integration Server. If the value is not specified, or the value is 0 or a negative number, the default port number is used.

Default 21036

The Job Status Data Set

The job status data set contains the following variables:

JOBID

identifies the job on the DataFlux Integration Server. These values can be applied to the functions DQSRVJOBSTATUS, DQSRVKILLJOB, DQSRVCOPYLOG, and DQSRVDELETELOG, as described in “DataFlux Integration Server Functions” on page 60.

STATUS

provides the following status codes:

0	Job completed successfully.
1	Job failed.
2	Job still running.

JOBDESC

provides the following descriptive text:

Architect job
 Profile job
 Service
 Unknown type

JOBTYP

provides the following job type codes:

0	Architect service
1	Architect job
2	Profile job

Security

If security is implemented on a DataFlux Integration Server, then you need to authenticate with the function DQSRVUSER before you run PROC DQSRVADM.

See “DQSRVUSER Function” on page 100

PROC DQSRVADM Examples

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 Integration Server.

```
proc dqsrvadm
  out=work.jobReport
  host='myhost'
  port=50001;
run;
```

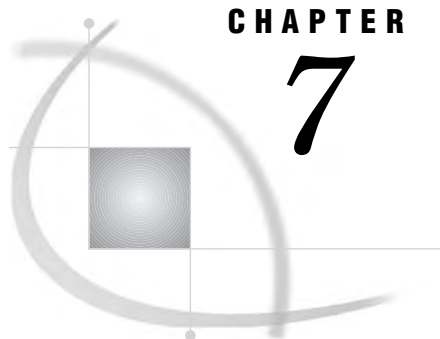
The job status data set contains information about jobs that are represented by log files on the server.

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 Integration Server:

```
proc dqsrvadm
  out=work.jobReport
  host='myhost'
  port=50001;
run;

data _null_;
  set work.joblist;
  kjrc=dqsrvkilljob (jobid, ' myhost' , 50001);
  dlrc=dqsrvdeletelog (jobid, ' myhost' , 50001);
run;
```

CHAPTER

7

The DQSRVSVC Procedure

<i>Overview: DQSRVSVC Procedure</i>	49
<i>What Does the DQSRVSVC Procedure Do?</i>	49
<i>Syntax: DQSRVSVC Procedure</i>	49
<i>PROC DQSRVSVC Statement</i>	50
<i>The Input and Output Data Sets</i>	52
<i>The Input Data Set</i>	52
<i>The Output Data Set</i>	52
<i>Examples</i>	52
<i>Example 1: Run a DataFlux dfPower Architect Service</i>	52
<i>Example 2: Macros= Option</i>	52

Overview: DQSRVSVC Procedure

What Does the DQSRVSVC Procedure Do?

The DQSRVSVC procedure runs a DataFlux dfPower Architect real-time service on a DataFlux Integration Server. DataFlux dfPower Architect real-time services are batch processes that are intended to cleanse smaller amounts of data at the point of data entry. Data processing is intended to be synchronous, when a client application requests the service and awaits a response. The DQSRVSVC 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 Integration Server in chunks of a specified size.

To cleanse or analyze larger amounts of data asynchronously, execute a DataFlux job using the functions DQSRVPROFJOB or DQSRVARCHJOB.

Syntax: DQSRVSVC Procedure

```

PROC DQSRVSVC DATA=<input-data-set>
    BLOCKSIZE=<rows-per-message>
    HOST=<host-name>
    MACROS=<macro-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 DQSRVSVC Statement

```
PROC DQSRVSVC <option(s)>;
```

Options

BLOCKSIZE=rows-per-message

specifies the number of rows of source data that are transmitted to the DataFlux Integration Server, in multiple messages. 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 Integration Server. The server processes each message separately. Output is delivered as usual in a single message.

The DataFlux dfPower Architect service program needs to be written to accommodate multiple messages.

Restriction: Services 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-machine

identifies the name of the machine hosting the DataFlux Integration Server. If the **HOST=** option is not specified, **localhost** is used by default.

Default: **localhost**

MACROS=macro-list

takes a quoted string as its value. The contents of the string is a series of name-value pairs. These pairs are passed to the service. If the service uses a macro with a name that matches a name in the macro list, the name is assigned the corresponding macro value.

Both the macro name and macro value in each pair must appear within single quotation marks. An equal sign must separate the macro name and the macro value. A comma separates each name-value pair from the next name-value pair in the **MACROS=** option list.

See: Example 2 on page 52

MISSINGVARSOK

indicates that the DataFlux 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, or required by the DataFlux real-time service.

Default: MISSINGVARSOK is not set by default.

NOPRINT

if the SERVICEINFO option is specified, suppresses writing the SERVICEINFO information to the SAS log.

OUT=*output-data-set*

identifies the name of the output data set. DataFlux dfPower Architect services always create new data sets or overwrite existing data sets.

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 Integration Server. The password can be plain text or SAS encoded.

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

See: “DataFlux Integration Server Passwords” on page 6

PORT=*port-number*

identifies the port number through which the **localhost** communicates with the DataFlux Integration Server. If this option is not specified, or if the value is zero or a negative number, the default port number 21036 is used.

Default: 21036

SERVICE=*service-name*

identifies the service on the DataFlux Integration Server.

SERVICEINFO

writes the input and output columns used by the given service to the data set 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 service information is also written to the SAS log.

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

TIMEOUT

specifies a time in seconds after which the procedure terminates if the **localhost** has not received a response from the DataFlux Integration Server. If data is delivered to the server in multiple messages using the BLOCKSIZE= option, 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.

TRIM

removes any blank spaces from the end of the input data set.

Default: TRIM is not set by default.

USERID

identifies the user according to the registry in the DataFlux Integration Server.

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

The Input and Output Data Sets

The Input Data Set

The DQSRVSVC procedure acquires the names of the columns that the service expects to receive as input from the DataFlux dfPower Architect service. In this case, the name of the input data set must match the name that is specified in the architect service. If the expected column names do not match the column names in the input data set, then the DQSRVSVC procedure terminates.

Services can also be created where any named data set can be used for input. In this case, there is no input data set name required.

The Output Data Set

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

Examples

Example 1: Run a DataFlux dfPower Architect Service

The following example runs a DataFlux dfPower Architect service on a DataFlux Integration Server that is installed on the local host:

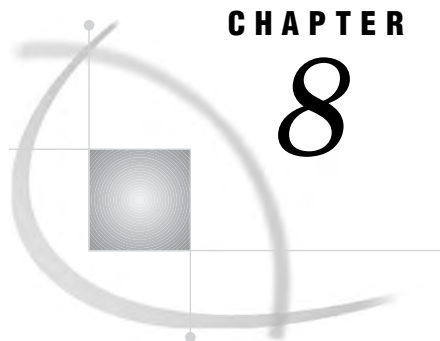
```
PROC DQSRVSVC
  SERVICE='myService'
  DATA=work.insrv
  OUT=work.outsrv;
RUN;
```

The PORT= option is not set, so the server communicates over the default port 21036. The DATA= option and the OUT= option specify that the input and output data sets are stored in the temporary WORK library. The service was previously created and uploaded to the DataFlux Integration Server.

Example 2: MACROS= Option

The value of the MACROS= option defines two macros, name1 and name2. The macro name1 has a value equal to 'value1', and the macro name2 has a value equal to 'value2'.

```
MACROS='' 'name1'='value1', 'name2' = 'value2' ''
```



CHAPTER

8

AUTOCALL Macros

AUTOCALL Macros for SAS Data Quality Server 53

AUTOCALL Macros for SAS Data Quality Server

SAS Data Quality Server software provides the following AUTOCALL macros:

- “%DQLOAD AUTOCALL Macro” on page 53
- “%DQPUTLOC AUTOCALL Macro” on page 54
- “%DQUNLOAD AUTOCALL Macro” on page 55

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

%DQLOAD AUTOCALL Macro

Sets system option values and loads locales into memory.

Syntax

%DQLOAD/DQLOCALE=(*locale-1*, ..., <*locale-n*>)

DQSETUPLOC="file-specification" | "path-specification"

DQINFO=0 | 1

DQLOCALE=(*locale-1*, ..., *locale-n*)

specifies an ordered list of locales to load into memory.

DQSETUPLOC="file-specification" | "path-specification"

identifies the location of the setup file. The setup file in turn identifies the location of the Quality Knowledge Base. The Quality Knowledge Base contains the specified locales. In that case, a setup file is not required.

Note: In Windows and UNIX operating environments, the path specification identifies the root directory of the Quality Knowledge Base.

DQINFO=0 | 1

generates additional information in the SAS log about the status of the locale load operation when DQINFO=1.

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. This loading prevents the use of an unintended default locale or locale list.

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 (if that value is not set by default at your site). When SAS is installed, the value of the DQSETUPLOC= option is set to point to the location where the setup file is installed.

Example

In this example, the %DQLOAD macro sets the value of the DQLOCALE system option. DQLOCALE loads the ENUSA and DEDUE locales into memory, in the UNIX environment. The %DQLOAD macro also sets the DQSETUPLOC system option, which points to the location where the setup file is installed.

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

%DQPUTLOC AUTOCALL Macro

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

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

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

Syntax

```
%DQPUTLOC (locale ,short ,parsedefn)
```

locale

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

The specified locale must have been loaded into memory as part of the locale list.

Default: The first locale in the locale list.

See: “Load and Unload Locales” on page 7

short

shortens the length of the entry in the SAS log. Valid values:

0

displays the descriptions of how the definitions are used.

1

removes the descriptions of how the definitions are used.

Default: 0

parsedefn

lists with each gender analysis definition and each match definition of the related parse definition, if such a parse definition exists.

0

does not display the related parse definition.

1

displays the related parse definition.

Default: 1

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. Knowing the related parse definitions enables the creation of parsed character values. See the “DQPARSETOKENPUT Function” on page 82.

It also enables the creation of match codes for parsed character values. See the “DQMATCHPARSED Function” on page 73.

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

Example

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

```
%DQPUTLOC(enusa);
```

See Also

- “DQLOCALEINFOGET Function” on page 69
- “DQLOCALEINFOLIST Function” on page 69

%DQUNLOAD AUTOCALL Macro

Unloads all locales to increase the amount of free memory.

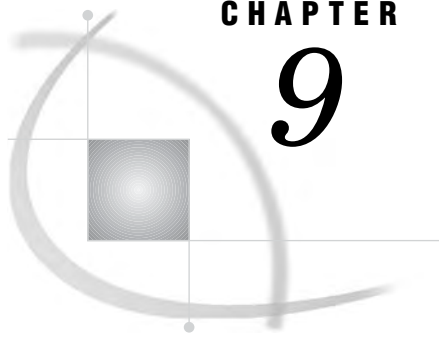
Requirement: After unloading locales from memory, load locales with the %DQLOAD AUTOCALL macro before running any data cleansing programs.

Syntax

%DQUNLOAD;

Details

The %DQUNLOAD AUTOCALL macro unloads all locales that are currently loaded into memory.



CHAPTER

9

Functions and CALL Routines

<i>Overview</i>	58
<i>Functions Listed Alphabetically</i>	58
<i>Functions Listed by Category</i>	60
<i>DataFlux Integration Server Functions</i>	60
<i>Case Functions</i>	60
<i>Gender Analysis, Locale Guessing, and Identification Functions</i>	60
<i>Matching Functions</i>	61
<i>Parsing Functions</i>	61
<i>Pattern Analysis Functions</i>	61
<i>Reporting Functions</i>	62
<i>Scheme Functions and CALL Routines</i>	63
<i>Standardization Functions</i>	63
<i>Functions: DQCASE Function</i>	63
<i>DQGENDER Function</i>	64
<i>DQGENDERINFOGET Function</i>	65
<i>DQGENDERPARSED Function</i>	66
<i>DQIDENTIFY Function</i>	67
<i>DQLOCALEGUESS Function</i>	68
<i>DQLOCALEINFOGET Function</i>	69
<i>DQLOCALEINFOLIST Function</i>	69
<i>DQMATCH Function</i>	71
<i>DQMATCHINFOGET Function</i>	72
<i>DQMATCHPARSED Function</i>	73
<i>DQOPTSURFACE Function</i>	74
<i>DQPARSE CALL Routine</i>	75
<i>DQPARSE Function</i>	76
<i>DQPARSEINFOGET Function</i>	77
<i>DQPARSEINPUTLEN Function</i>	78
<i>DQPARSERESLIMIT Function</i>	79
<i>DQPARSESCORDEPTH Function</i>	80
<i>DQPARSETOKENGET Function</i>	81
<i>DQPARSETOKENPUT Function</i>	82
<i>DQPATTERN Function</i>	83
<i>DQSCHEMEAPPLY CALL Routine</i>	84
<i>DQSCHEMEAPPLY Function</i>	88
<i>DQSRVARCHJOB Function</i>	91
<i>DQSRVCOPYLOG Function</i>	93
<i>DQSRVDELETEDLOG Function</i>	94
<i>DQSRVJOBSTATUS Function</i>	95
<i>DQSRVKILLJOB Function</i>	96
<i>DQSRVPROFJOBFILE Function</i>	97

<i>DQSRVPROFJOBREP Function</i>	99
<i>DQSRVUSER Function</i>	100
<i>DQSRVVER Function</i>	101
<i>DQSTANDARDIZE Function</i>	102
<i>DQTOKEN Function</i>	103
<i>DQVERBF Function</i>	104
<i>DQVERQKB Function</i>	105

Overview

The functions and CALL routines in the SAS Data Quality Server software enable you to cleanse data and access DataFlux Integration Servers from DataFlux (a SAS company).

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. Δ

Functions Listed Alphabetically

The “Functions: DQCASE Function” on page 63 returns a character value with standardized capitalization.

The “DQGENDER Function” on page 64 returns a gender determination from the name of an individual.

The “DQGENDERINFOGET Function” on page 65 returns the name of the parse definition that is associated with a specified gender analysis definition.

The “DQGENDERPARSED Function” on page 66 returns a gender determination from the parsed name of an individual.

The “DQIDENTIFY Function” on page 67 returns a category name from a character value.

The “DQLOCALEGUESS Function” on page 68 returns the name of the locale that is most likely represented by a character value.

The “DQLOCALEINFOGET Function” on page 69 returns information about locales.

The “DQLOCALEINFOLIST Function” on page 69 returns the names of the definitions in a locale and returns a count of those definitions.

The “DQMATCH Function” on page 71 returns a match code from a character value.

The “DQMATCHINFOGET Function” on page 72 returns the name of the parse definition that is associated with a match definition.

The “DQMATCHPARSED Function” on page 73 returns a match code from a parsed character value.

The “DQOPTSURFACE Function” on page 74 reveals or hides non-surfaced definitions.

The “DQPARSE CALL Routine” on page 75 returns a parsed character value and a status flag.

The “DQPARSE Function” on page 76 returns a parsed character value.

- The “DQPARSEINFOGET Function” on page 77 returns the token names for the specified parse definition.
- The “DQPARSEINPUTLEN Function” on page 78 sets the default length of parsed input, and returns a string indicating its previous value.
- The “DQPARSERESLIMIT Function” on page 79 sets a limit on resources consumed during parsing.
- The “DQPARSESCORDEPTH Function” on page 80 specifies how deeply to search for the best parsing score.
- The “DQPARSETOKENGET Function” on page 81 returns a token from a parsed character value.
- The “DQPARSETOKENPUT Function” on page 82 inserts a token into a parsed character value and returns the updated parsed character value.
- The “DQPATTERN Function” on page 83 returns a pattern analysis from an input character value.
- The “DQSCHEMEAPPLY CALL Routine” on page 84 applies a scheme and returns a transformed value and a transformation flag.
- The “DQSCHEMEAPPLY Function” on page 88 applies a scheme and returns a transformed value after applying a scheme.
- The “DQSRVARCHJOB Function” on page 91 runs a DataFlux dfPower Architect job on a DataFlux Integration Server and returns a job identifier.
- The “DQSRVCOPYLOG Function” on page 93 copies a job’s log file from a DataFlux Integration Server.
- The “DQSRVDELETELOG Function” on page 94 deletes a job’s log file from a DataFlux Integration Server.
- The “DQSRVJOBSTATUS Function” on page 95 returns the status of a job that was submitted to a DataFlux Integration Server.
- The “DQSRVKILLJOB Function” on page 96 terminates a job that is running on a DataFlux Integration Server.
- The “DQSRVPROFJOBFILE Function” on page 97 runs a file type Profile job on a DataFlux Integration Server and returns a job identifier.
- The “DQSRVPROFJOBREP Function” on page 99 runs a repository–type Profile job on a DataFlux Integration Server and returns a job identifier.
- The “DQSRVUSER Function” on page 100 begins a session on a secure DataFlux Integration Server.
- The “DQSRVVER Function” on page 101 returns the version of the DataFlux Integration Server.
- The “DQSTANDARDIZE Function” on page 102 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.
- The “DQTOKEN Function” on page 103 returns a token from a character value.
- The “DQVERBF Function” on page 104 returns the version of Blue Fusion.
- The “DQVERQKB Function” on page 105 returns the version of the currently loaded Quality Knowledge Base.

Functions Listed by Category

DataFlux Integration Server Functions

The “DQSRVARCHJOB Function” on page 91 runs a DataFlux dfPower Architect job on a DataFlux Integration Server and returns a job identifier.

The “DQSRVCOPYLOG Function” on page 93 copies a job’s log file from a DataFlux Integration Server.

The “DQSRVDELETELOG Function” on page 94 deletes a job’s log file from a DataFlux Integration Server.

The “DQSRVJOBSTATUS Function” on page 95 returns the status of a job that was submitted to a DataFlux Integration Server.

The “DQSRVKILLJOB Function” on page 96 terminates a job that is running on a DataFlux Integration Server.

The “DQSRVPROFJOBFILE Function” on page 97 runs a file-type Profile job on a DataFlux Integration Server and returns a job identifier.

The “DQSRVPROFJOBREP Function” on page 99 runs a repository-type Profile job on a DataFlux Integration Server and returns a job identifier.

The “DQSRVUSER Function” on page 100 begins a session on a secure DataFlux Integration Server.

The “DQSRVVER Function” on page 101 returns the version of the DataFlux Integration Server.

The “DQVERBF Function” on page 104 returns the version of Blue Fusion.

The “DQVERQKB Function” on page 105 returns the version of the currently loaded Quality Knowledge Base.

Case Functions

The “Functions: DQCASE Function” on page 63 returns a character value with standardized capitalization.

The “DQSTANDARDIZE Function” on page 102 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.

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” on page 64 returns a gender determination from the name of an individual.

The “DQGENDERINFOGET Function” on page 65 returns the name of the parse definition that is associated with a specified gender analysis definition.

The “DQGENDERPARSED Function” on page 66 returns a gender determination from the parsed name of an individual.

The “DQIDENTIFY Function” on page 67 returns a category name from a character value.

The “DQLOCALEGUESS Function” on page 68 returns the name of the locale that is most likely represented by a character value.

The “DQLOCALEINFOGET Function” on page 69 returns information about locales.

The “DQLOCALEINFOLIST Function” on page 69 returns the names of the definitions in a locale and returns a count of those definitions.

Matching Functions

The “DQMATCH Function” on page 71 returns a match code from a character value.

The “DQMATCHINFOGET Function” on page 72 returns the name of the parse definition that is associated with a match definition.

The “DQMATCHPARSED Function” on page 73 returns a match code from a parsed character variable.

Parsing Functions

The “DQGENDERINFOGET Function” on page 65 returns the name of the parse definition that is associated with a specified gender analysis definition.

The “DQGENDERPARSED Function” on page 66 returns a gender determination from the parsed name of an individual.

The “DQMATCHPARSED Function” on page 73 returns a match code from a parsed character value.

The “DQPARSE CALL Routine” on page 75 returns a parsed character value and a status flag.

The “DQPARSE Function” on page 76 returns a parsed character value.

The “DQPARSEINFOGET Function” on page 77 returns the token names for the specified parse definition.

The “DQPARSEINPUTLEN Function” on page 78 sets the default length of parsed input. DQPARSEINPUTLEN also returns a string indicating its previous value.

The “DQPARSERESLIMIT Function” on page 79 sets a limit on resources consumed during parsing.

The “DQPARSESCORDEPTH Function” on page 80 specifies how deeply to search for the best parsing score.

The “DQPARSETOKENGET Function” on page 81 returns a token from a parsed character value.

The “DQPARSETOKENPUT Function” on page 82 inserts a token into a parsed character value and returns the updated parsed character value.

Pattern Analysis Functions

The “DQPATTERN Function” on page 83 returns a pattern analysis from an input character value.

Reporting Functions

- The “DQGENDER Function” on page 64 returns a gender determination from the name of an individual.
- The “DQGENDERPARSED Function” on page 66 returns a gender determination from the parsed name of an individual.
- The “DQGENDERINFOGET Function” on page 65 returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “DQIDENTIFY Function” on page 67 returns a category name from a character value.
- The “DQLOCALEGUESS Function” on page 68 returns the name of the locale that is most likely represented by a character value.
- The “DQLOCALEINFOGET Function” on page 69 returns information about locales.
- The “DQLOCALEINFOLIST Function” on page 69 returns the names of the definitions in a locale and returns a count of those definitions.
- The “DQMATCH Function” on page 71 returns a match code from a character value.
- The “DQMATCHINFOGET Function” on page 72 returns the name of the parse definition that is associated with a match definition.
- The “DQMATCHPARSED Function” on page 73 returns a match code from a parsed character value.
- The “DQPARSE CALL Routine” on page 75 returns a parsed character value and a status flag.
- The “DQPARSE Function” on page 76 returns a parsed character value.
- The “DQPARSEINFOGET Function” on page 77 returns the token names for the specified parse definition.
- The “DQPARSETOKENGET Function” on page 81 returns a token from a parsed character value.
- The “DQPARSETOKENPUT Function” on page 82 inserts a token into a parsed character value and returns the updated parsed character value.
- The “DQPATTERN Function” on page 83 returns a pattern analysis from an input character value.
- The “DQSCHEMEAPPLY CALL Routine” on page 84 applies a scheme and returns a transformed value and a transformation flag.
- The “DQSCHEMEAPPLY Function” on page 88 applies a scheme and returns a transformed value after applying a scheme.
- The “DQSTANDARDIZE Function” on page 102 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.
- The “DQTOKEN Function” on page 103 returns a token from a character value.
- The “DQVERBF Function” on page 104 returns the version of Blue Fusion.
- The “DQVERQKB Function” on page 105 returns the version of the currently loaded Quality Knowledge Base.

Scheme Functions and CALL Routines

The “DQSCHEMEAPPLY Function” on page 88 applies a scheme and returns a transformed value.

The “DQSCHEMEAPPLY CALL Routine” on page 84 applies a scheme and returns a transformed value and a transformation flag.

Standardization Functions

The “Functions: DQCASE Function” on page 63 returns a character value with standardized capitalization.

The “DQSTANDARDIZE Function” on page 102 returns a character value after standardizing the casing, spacing, and format, and after applying a common representation to certain words and abbreviations.

Functions: DQCASE Function

Returns a character value with standardized capitalization.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is transformed, according to the specified case definition.

case-definition

is the character constant, variable, or expression to search. The definition must be in the locale that is used. If the value of *char* is represented by a case definition, the use of that definition is recommended, over the generic case definition.

If the value of *char* is a street address and you are using the ENUSA locale; the recommended case definition is PROPER-ADDRESS. This is used instead of the generic case definition PROPER.

locale

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

Default: The default locale is the first locale in the locale list. 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

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.

DQGENDER Function

Returns a gender determination from the name of an individual.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

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, that must exist in the specified locale. The value must be the name of a character variable, in quotation marks. Also valid, an expression that evaluates to a variable name, or a quoted value.

locale

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

Default: The default locale is the first locale in the locale list. 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

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

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

See Also

Functions:

“DQGENDERPARSED Function” on page 66

DQGENDERINFOGET Function

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

gender-analysis-definition

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

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Example

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

Functions:

“DQGENDER Function” on page 64

“DQGENDERPARSED Function” on page 66

“DQPARSE Function” on page 76

“DQPARSETOKENPUT Function” on page 82

DQGENDERPARSED Function

Returns the gender of an individual.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

parsed-char

is the value that is analyzed to determine the gender of an individual. The value must be the name of a character variable, or a character value in quotation marks. Also valid, an expression that evaluates to a variable name or quoted value.

gender-analysis-definition

specifies the name of the gender analysis definition. The analysis definition must exist in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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

Functions:

“DQGENDER Function” on page 64

“DQGENDERINFOGET Function” on page 65

DQIDENTIFY Function

Returns a category name from a character value.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is analyzed to determine that category of the content.

identification-definition

is the name of the identification definition. The definition must be in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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 character value. The available categories and return values depend on your choice of identification definition and locale.

Example

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: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is analyzed to determine the locale, according to the specified guess definition.

locale-guess-definition

specifies a character constant, variable, or expression that contains the locale-guess-definition.

Details

The DQLOCALEGUESS function evaluates the input character value using the specified locale guess definition in each of the locales that are loaded into memory. An applicability score is generated for each locale in the locale list. If multiple locales hold the highest score definition, or none of the locales have the guess definition, the return value is the first locale in the locale list. The name of the locale that is returned depends on which locales are loaded into memory.

Example

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

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

See Also

Function:

“DQLOCALEINFOGET Function” on page 69

“Load and Unload Locales” on page 7

DQLOCALEINFOGET Function

Returns information about locales.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQLOCALEINFOGET (<'info-type'>)

Arguments

info-type

(optional) is 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 locale names. The ordered list contains the names of the locales that are currently loaded into memory. These locales are available for use in data cleansing.

Example

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

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

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

See Also

Function and autocall macro:

“DQLOCALEINFOLIST Function” on page 69

“%DQPUTLOC AUTOCALL Macro” on page 54

DQLOCALEINFOLIST Function

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

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
- GENDER
- GUESS
- IDENTIFICATION
- MATCH
- PARSE
- PATTERN
- STANDARDIZATION

locale

specifies a character constant, variable, or expression that contains the locale name. If no value is specified, the default locale is used.

Default: The default locale is the first locale in the locale list.

Details

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.

Examples

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

Functions:

“Load and Unload Locales” on page 7

“DQLOCALEINFOGET Function” on page 69

DQMATCH Function

Returns a match code from a character value.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value for which a match code is created, according to the specified match definition.

match-definition

specifies the name of the match definition. The definition must exist in the locale that is used.

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, because matches require greater similarity between input values.

locale

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

Default: The default locale is the first locale in the locale list. 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 match codes. At lower sensitivities, two values produce the same match codes despite their dissimilarities.

Example

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

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

See Also

Functions:

Chapter 4, “The DQMATCH Procedure,” on page 19

DQMATCHINFOGET Function

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

match-definition

is the name of the match definition. The definition must exist in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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 returns a missing value.

Example

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

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



```

run;

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

```

DQMATCHPARSED Function

Returns a match code from a parsed character value.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

match-definition

specifies the name of the match definition. The definition must exist in the locale that is used.

parsed-char

specifies a character constant, variable, or expression that contains the value that is the name of the parsed-definition that is associated with the match definition.

To determine the name of the associated parse definition, use the DQMATCHINFOGET function. To determine the tokens that are enabled by that parse definition, use the DQPARSEINFOGET function.

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; input values must be more similar to receive the same match codes.

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Example

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 4, “The DQMATCH Procedure,” on page 19

“Create Match Codes” on page 11

“DQMATCHINFOGET Function” on page 72

“DQPARSEINFOGET Function” on page 77

“DQTOKEN Function” on page 103

DQOPTSURFACE Function

Reveals or hides non-surfaced definitions.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQOPTSURFACE (*'surface-definition'*)

Arguments

surface-definition

specifies the policy for the surface definitions.

Details

DQOPTSURFACE 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

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 CALL Routine

Returns a parsed character value and a status flag.

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.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

```
CALL DQPARSE ( parse-definition, parse-result, parse-return-code, parse-string, <locale> )
```

Arguments

parse-definition

is the name of the parse definition. The definition must exist in the locale that is used.

parse-result

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

parse-return-code

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

parse-string

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

locale

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

Default: The default locale is the first locale in the locale list. 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 delimiters that identify the elements in the value that correspond to the tokens that are enabled by the parse

definition. The delimiters in the value allow functions such as DQPARSETOKENGET to access the elements in the value based on specified token names.

Example

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

Functions:

“DQPARSEINFOGET Function” on page 77

“DQTOKEN Function” on page 103

DQPARSE Function

Returns a parsed character value.

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

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.

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

parse-definition

is the name of the parse definition. The definition must exist in the locale that is used.

parse-string

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

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

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

Example

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

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

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

See Also

Functions:

“DQPARSEINFOGET Function” on page 77

“DQTOKEN Function” on page 103

DQPARSEINFOGET Function

Returns the token names in a parse definition.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments***parse-definition***

specifies the name of the parse definition. The definition must exist in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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

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

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

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

See Also

Functions:

“DQPARSETOKENGET Function” on page 81

“DQPARSETOKENPUT Function” on page 82

“DQTOKEN Function” on page 103

DQPARSEINPUTLEN Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQPARSEINPUTLEN (*input-length*)

Arguments***input-length***

specifies the input length for parsing functions. DQPARSEINPUTLEN returns the previous value of the input length.

Details

The DQPARSEINPUTLEN function specifies the input length anticipated by parsing functions. If REMOVE is specified, the override value is removed and the input limit is set to the default value. Valid values for the input length are as follows:

- SHORT
- LONG
- AUTO
- REMOVE

The DQPARSEINPUTLEN function returns a value indicating the previous value of the input length. If the value NOTSET is returned, the override value is not set.

Possible values for the previous input length are as follows:

- SHORT
- LONG
- AUTO
- NOTSET

Example

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

Sets a limit on resources consumed during parsing.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQPARSERESLIMIT (*resource-limit*)

Arguments

resource limit

specifies the resource limit a parsing operation is allowed to consume.

Details

The DQPARSERESLIMIT function sets the level of resource consumption in force during parsing operations. If REMOVE is specified, the override value is removed and the resource limit is set to the default value. Valid values are as follows:

- VERYLOW
- LOW
- MEDIUM
- HIGH
- VERYHIGH
- INTENSIVE
- REMOVE

The DQPARSERESLIMIT function returns a value indicating the previous value of the resource limit. If the value NOTSET is returned, the override value is not set. Possible return values are as follows:

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

Example

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

Specifies how deeply to search for the best parsing score.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQPARSESCORDEPTH (*level*)

Arguments

level

is the maximum depth permitted during scoring.

Details

The DQPARSESCORDEPTH function sets the level of how deeply to search for the best parsing score. LEVEL must be in the range from five to ten inclusive, or zero. If at least one of the conditions is true, DQPARSESCORDEPTH overrides the default scoring depth value. If zero is returned, there is no override value in force.

The DQPARSESCORDEPTH function returns the previous value of the override solutions depth.

Example

The following example sets DQPARSESCORDEPTH to eight. The numeric variable **oldDEFAULT** contains the scoring depth previously in force.

```
oldDefault=DQPARSESCORDEPTH(8);
```

DQPARSETOKENGET Function

Returns a token from a parsed character value.

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

parsed-char

specifies a character constant, variable, or expression that contains the value that is the parsed character value from which the value of the specified token is returned.

To determine how the parse definition inserts delimiters, use the DQPARSEINFOGET function.

parse-definition

is the name of the parse definition. The definition must exist in the locale that is used. The parse definition must be the same as the parse definition that originally parsed the PARSED-CHAR value.

token

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

locale

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

Default: The default locale is the first locale in the locale list. 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

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 76

“DQPARSEINFOGET Function” on page 77

“DQTOKEN Function” on page 103

DQPARSETOKENPUT Function

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

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

Valid in: DATA step and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is the parsed character value that receives the new token value.

parse-definition

is the name of the parse definition. The definition must exist in the locale that is used. The parse definition must be the same definition that was used to parse the *parsed-char* value.

token-name

is the name of the token. The specified token must be enabled by the parse definition.

token-value

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

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQPARSETOKENPUT function enables you to insert a new value that is associated with a specified token into a parsed 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 *parsed-char*, and then assign the return value from DQPARSETOKENPUT to the same variable.

See Also

Functions:

“DQGENDERINFOGET Function” on page 65

“DQGENDERPARSED Function” on page 66

“DQMATCHPARSED Function” on page 73

“DQPARSETOKENGET Function” on page 81

DQPATTERN Function

Returns a pattern analysis from an input character value.

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

Valid in: DATA step and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is the name of the input character value that is analyzed.

pattern-analysis-definition

is the name of the pattern analysis definition. The definition must exist in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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

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 CALL Routine

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

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

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

Valid in: DATA step and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is the input value to which the scheme is applied.

output-variable

is the character variable that receives the transformed input value.

scheme

is the scheme that is applied to the input value. A SAS format scheme is a filename specification that includes; a pathname and the SAS data set name enclosed in quotation marks.

Blue Fusion Data format, 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 pathname and the filename that ends in .sch.bfd.

Requirement: Lowercase letters 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 formats are as follows:

BFD

is the Blue Fusion Data format. This is the default value.

NOBFD

is the SAS data format. See “Schemes” on page 8.

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*, PHRASE is used.

If the value of *ofscheme-lookup-method* is USE_MATCHDEF, and a value is not specified for *mode*, 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 *match codes* 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 match codes 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.

scheme-lookup-method

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

EXACT

this default value specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces 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 *match-code* of the input value is to be compared to the *match-code* of the DATA values in the scheme. A transformation occurs when the two match codes are identical.

Specifying USE_MATCHDEF enables you to modify the values of *locale*, *match-definition*, and *sensitivity*.

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

match-definition

is the name of the match definition. The definition must exist in the locale that is used to create match codes during the application of the scheme.

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

If USE_MATCHDEF is specified and *match-definition* is not specified, the default match definition is stored in the scheme.

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

sensitivity

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

Note: *Sensitivity* is valid only when the value of the *scheme-lookup-method* is `USE_MATCHDEF`. Δ

When `USE_MATCHDEF` is specified and *sensitivity* is not specified, the *sensitivity* value that is stored in the scheme is used. If there is no *sensitivity* in the scheme, the sensitivity value is 85, the default.

locale

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

Default: The default locale is the first locale in the locale list. 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`.

transform-count-variable

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

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 *transform-count-variable* is 0.

If the input variable is transformed, then the value is a positive integer that represents the number of elements in the input value that are transformed.

Note: 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).

Details

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

Example

The following example generates a scheme using Blue Fusion Data format with the `DQSCHEME` procedure and then applies that scheme to a data set with `CALL DQSCHEMEAPPLY`. The example assumes that `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 Permanente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;

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

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

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

```

The value of the NUMTRANS variable is 0 if the organization name is not transformed. 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

Functions:

Chapter 5, “The DQSCHHEME Procedure,” on page 33

“DQSCHHEMEAPPLY Function” on page 88

DQSCHHEMEAPPLY Function

Applies a scheme and returns a transformed value.

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSCHHEMEAPPLY (*char*, '*scheme*', '*scheme-format*', < '*match-definition*'>, < '*mode*'>, < '*scheme-lookup-method*'>, < '*sensitivity*'>, < '*locale*'>)

Arguments

char

specifies a character constant, variable, or expression that contains the value to which the specified scheme is applied.

scheme

identifies the scheme that is applied to the input value. For schemes using SAS format, the *scheme* argument includes both the path and the filename of the SAS data set, in quotation marks.

For schemes using Blue Fusion Data format, the *scheme* argument is the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a file specification that includes both the path and the filename that ends in .sch.bfd.

Requirement: Lowercase letters 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 file format of the scheme. Valid values are as follows:

BFD

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

NOBFD

indicates that the scheme is stored in SAS format.

See: “Applying Schemes” on page 9

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

match-definition

is the name of the match definition. The definition must exist in the locale that is used to create match codes during the application of the scheme. If

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

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 9

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*, PHRASE is used.

If the value of *ofscheme-lookup-method* is USE_MATCHDEF, and a value is not specified for *mode*, 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 *match codes* 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 match codes 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.

scheme-lookup-method

specifies one of three mutually exclusive methods of applying the scheme.

EXACT

this default value specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces 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 *match-code* of the input value is to be compared to the *match-code* of the DATA values in the scheme. A transformation occurs when the *match-codes* 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 the *scheme-lookup-method* function is USE_MATCHDEF.

See: “Applying Schemes” on page 9

sensitivity

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

Default: When *use_matchdef* is specified and the sensitivity argument is not specified, the default sensitivity is the sensitivity value that is stored in the scheme. When USE_MATCHDEF is specified and a sensitivity value is not stored in the scheme, the default sensitivity value is 85.

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 DQSCHHEMEAPPLY CALL routine.

See: “Meta Options” on page 9

Valid values: 50 to 95

Details

The *locale* argument is valid only when the value of the *scheme-lookup-method* argument is USE_MATCHDEF. The DQSCHHEMEAPPLY function transforms an input value by applying a scheme. The scheme can be in SAS format or Blue Fusion Data format. SAS format schemes can be created with the DQSCHHEME procedure. Create schemes using Blue Fusion Data format with the DQSCHHEME procedure or with the DataFlux dfPower Studio software from DataFlux (a SAS company).

Example

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

```
/* 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 Permanente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;

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

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

/* Apply the scheme and display the results. */
data suppliers;
  set suppliers;
  length outCompany $ 50;
  outCompany=dqSchemeApply(company,'myscheme','bfd','phrase','EXACT');
  put 'Before applying the scheme: ' company /
    'After applying the scheme: ' outCompany;
run;

```

See Also

Chapter 5, “The DQSCHEME Procedure,” on page 33
 “DQSCHEMEAPPLY CALL Routine” on page 84

DQSRVARCHJOB Function

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

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

Requirement: The character variable that receives the return value must have a minimum length of 52.

Note: Since the DATA step creates a character variable automatically when the DQSRVARCHJOB function is called, there is no need to create a variable with a 'length' statement to hold the return code. If you choose to create a variable, then the variable must be at least 52 characters in length or errors are generated. △

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVARCHJOB (*job-name*<,<*host*>><,<*port*>><,<*macro-name1*>><,<*macro-value1*>><,<*macro-name2*>><,<*macro-value2*...>>)

Arguments

job-name

is the DataFlux dfPower Architect job as it exists on the specified DataFlux Integration Server.

host

(optional) is the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used.

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

macro-name1

is the DataFlux dfPower Architect macro that exists on the DataFlux Integration Server. The value of *macro-name1* can be specified as text, or as the name of a character variable. The value of the character variable is used as the macro name.

macro-value1

is the character value that is used by the associated DataFlux dfPower Architect macro. *Macro-value1* is used by *macro-name1*. The value of *macro-value1* can be specified as text, or as the name of a character variable.

Details

The DQSRVARCHJOB 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 DQSRVJOBSTATUS, DQSRVCOPYLOG, DQSRVDELETEDLOG, and DQSRVKILLJOB.

- You can specify any number of macro value pairs.
- For information about how to run a DataFluxDataFlux dfPower Architect service on a DataFlux Integration Server, see Chapter 7, “The DQSRVSVC Procedure,” on page 49.

Example

The following example runs a DataFlux dfPower Architect job on a DataFlux Integration Server. The job standardizes a character value that represents the name of an operating environment. The server returns a job identifier that can be used to manage the job’s log file.

```
jobid= dqsrvArchJob('opsysCleanse','archServer1', 5001,'osMacro','Unicks');
```

See Also

Functions:

“DQSRVCOPYLOG Function” on page 93

“DQSRVDELETEDLOG Function” on page 94

“DQSRVJOBSTATUS Function” on page 95

DQSRVCOPYLOG Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVCOPYLOG (*job-ID* <,host> <,.port> ,filename)

Arguments

job-ID

identifies the job that is submitted to a DataFlux Integration Server. The identifier is previously returned by a function such as DQSRVARCHJOB or DQSRVPROFJOBFILE.

host

identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used.

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

filename

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

Details

To capture log information for a particular job, use the DQSRVJOBSTATUS 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

The following example copies a log file from a DataFlux Integration 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= dqsrvCopyLog(jobid,'archServer1', 5001,'archServer1.log');
```

See Also

Functions:

“DQSRVARCHJOB Function” on page 91

“DQSRVDELETELOG Function” on page 94

“DQSRVJOBSTATUS Function” on page 95

DQSRVDELETELOG Function

Deletes a job’s log file from a DataFlux Integration Server.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVDELETELOG (*job-ID*, <,*host*><,*port*>)

Arguments

job-ID

identifies the job submitted to a DataFlux Integration Server. The identifier is set by a function such as DQSRVARCHJOB or DQSRVPROFJOBFILE.

host

identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

Details

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

- DQSRVDELETELOG 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

The following example deletes a log file from a DataFlux Integration 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= dqsrvDeleteLog(jobid,'archServer1', 5001);
```

See Also

Functions:

- “DQSRVARCHJOB Function” on page 91
- “DQSRVCOPYLOG Function” on page 93
- “DQSRVJOBSTATUS Function” on page 95

DQSRVJOBSTATUS Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVJOBSTATUS(*job-ID* <,*host*> <,*port*>,*time-out* ,*interval*)

Arguments

Job-ID

identifies the job that was submitted to a DataFlux Integration Server. The identifier is previously set by a function such as DQSRVARCHJOB or DQSRVPROFJOBFILE.

host

identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

time-out

is 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

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

Details

Use the DQSRVJOBSTATUS 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, DQSRVJOBSTATUS 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

The following example returns a status number for a job that ran or is running on a DataFlux Integration 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= dqsrvJobStatus(jobid,'archServer1', 5001, 20, 5);
```

See Also

Functions:

“DQSRVARCHJOB Function” on page 91

“DQSRVDELETELOG Function” on page 94

“DQSRVKILLJOB Function” on page 96

DQSRVKILLJOB Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVKILLJOB (*job-ID* <,*host*> <,*port*>)

Arguments

job-ID

identifies the job submitted to a DataFlux Integration Server. The identifier is set by a function such as DQSRVARCHJOB or DQSRVPROFJOBFILE.

host

identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

Details

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

Example

The following example terminates a job that is running on a DataFlux Integration 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= dqsrvKillJob(jobid,'archServer1',5001);
```

See Also

Functions:

“DQSRVARCHJOB Function” on page 91

“DQSRVJOBSTATUS Function” on page 95

DQSRVPROFJOBFILE Function

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

Requirement: The character variable that receives the return value must have a minimum length of 52 characters.

Valid in: DATA step, PROC SQL, and SCL

Syntax

```
DQSRVPROFJOBFILE (job-name<,&i>host><,&i>port>,&i>results-filename ,append ,  
description<,&i>macro-name1> <,&i>macro-value1> <,&i>macro-name2> <,&i>macro-value2...>)
```

Arguments

append

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.

description

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 is appended to the bottom of the results file.

host

(optional) identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used

job-name

identifies the DataFlux dfPower Profile job as it exists on the specified DataFlux Integration Server.

macro-name1

identifies a DataFlux dfPower Profile macro that exists on the DataFlux Integration Server. The value of *macro-name1* can be specified as text or as the name of a character variable. The value of the variable specifies the name of the macro.

macro-value1

specifies the character value that is used by the associated DataFlux dfPower Profile macro. *Macro-value1* is used by *macro-name1*. The value of *macro-value1* can be specified as text, or as the name of a character variable.

macro-name2

identifies a DataFlux dfPower Profile macro that exists on the DataFlux Integration Server. The value of *macro-name2* can be specified as text or as the name of a character variable. The value of the variable specifies the name of the macro.

macro-value2

specifies the character value that is used by the associated DataFlux dfPower Profile macro. *Macro-value2* is used by *macro-name2*. The value of *macro-value2* can be specified as text, or as the name of a character variable.

port

(optional) identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

results-filename

identifies the file that receives job results.

Example

The following example runs a data analysis job on a file of customer data using a DataFlux Integration Server and the DataFlux dfPower Profile software. A macro identifies the location of the data.

```
jobid=dqsrvProfJobFile('ProfileCustomerData','profServer2',5001,
'custprof.pfo',1,'Profile of Customer Data','datapath',
'/dept/marketing/customers');
```

See Also

Functions:

- “DQSRVJOBSTATUS Function” on page 95
- “DQSRVKILLJOB Function” on page 96
- “DQSRVPROFJOBREP Function” on page 99

DQSRVPROFJOBREP Function

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

Requirement: The character variable that receives the return value must have a minimum length of 52 characters.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVPROFJOBREP

```
(job-name<,host><,<port>,repository,report,description<,macro-name1>
<,macro-value1> <,macro-name2> <,macro-value2...>)
```

Arguments

description

is 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 is appended to the bottom of the results file.

host

identifies the host of the DataFlux Integration Server. If this value is not specified, then **localhost** is used

job-name

is the DataFlux dfPower Profile job as it exists on the specified DataFlux Integration Server.

macro-name1

identifies a DataFlux dfPower Profile macro that exists on the DataFlux Integration Server. The value of *macro-name1* can be specified as text or as the name of a character variable. The value of the variable specifies the name of the macro.

macro-value1

specifies the character value that is used by the associated DataFlux dfPower Profile macro. *Macro-value1* is used by *macro-name1*. The value of *macro-value1* can be specified as text, or as the name of a character variable.

macro-name2

identifies a DataFlux dfPower Profile macro that exists on the DataFlux Integration Server. The value of *macro-name2* can be specified as text or as the name of a character variable. The value of the variable specifies the name of the macro.

macro-value2

specifies the character value that is used by the associated DataFlux dfPower Profile macro. *Macro-value2* is used by *macro-name2*. The value of *macro-value2* can be specified as text, or as the name of a character variable.

port

identifies the port through which the local host communicates with the DataFlux Integration Server. If this value is not specified, or if the value is 0 or a negative number, then the default port number 21036 is used.

report

is the file that receives the report that is generated by the DataFlux dfPower Profile job.

repository

is the repository on the DataFlux Integration Server that contains the DataFlux dfPower Profile job.

Example

The following example runs a data analysis job on a repository of sales data using a DataFlux Integration Server and the DataFlux dfPower Profile software. A macro identifies the location of the data.

```
jobid= dqsrvProfJobRep('ProfileSalesRepository','profServer2',5001,
    'Sales_Repository',1,'Profile of Sales Data','datapath',
    '/dept/sales/Q3');
```

See Also

Functions:

“DQSRVPROFJOBFILE Function” on page 97

“DQSRVJOBSTATUS Function” on page 95

“DQSRVKILLJOB Function” on page 96

DQSRVUSER Function

Authenticates a user on a DataFlux Integration Server.

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments***user-ID***

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

password

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

Details

The DQSRVUSER function authenticates a user on a secure DataFlux Integration Server. A return value of zero indicates successful authentication. A return value of 1 indicates a failure to authenticate.

- Call this function as needed in a single DATA step to access different Integration Servers or to change authorizations within a single Integration Server.
- If security has not been configured on a DataFlux Integration Server, then the DQSRVUSER function has no effect.
- Return values are 0 (successful authentication) or 1 (failed to authenticate).

Example

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

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

DQSRVVER Function

Returns the version of the DataFlux Integration Server.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQSRVVER (<host> <,port>)

Arguments

host

is the host of the DataFlux Integration Server. If this value is not specified, **localhost** is used.

port

is the port through which the local host communicates with the DataFlux Integration Server. If the value is not specified, the default 21036 is used.

Details

The DQSRVVER 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.

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

Example

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

```
version=DQSRVVER ('myhost',19525);
```

See Also

“DQSRVARCHJOB Function” on page 91

DQSTANDARDIZE Function

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

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is standardized according to the specified standardization definition.

standardization-definition

specifies the name of the standardization definition. The definition must exist in the locale that is used.

locale

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

Default: The default locale is the first locale in the locale list. 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 might standardize certain words and abbreviations. The order of the elements in the return value might differ from the order of the elements in the input character value.

Example

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

```
data _null_;
  length name stdName $ 50;
```

```

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

```

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

```

Name:    HOUSE, KEN
StdName: Ken House

Name:    House, Kenneth
StdName: Kenneth House

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

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

```

DQTOKEN Function

Returns a token from a character value.

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Arguments

char

specifies a character constant, variable, or expression that contains the value that is the value from which the specified token is returned, according to the specified parse definition.

parse-definition

is the name of the parse definition. The definition must exist in the locale that is used.

token

identifies the token that is returned.

locale

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

Default: The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

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

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

Example

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.**

See Also

Functions:

“DQPARSE Function” on page 76

“DQPARSETOKENGET Function” on page 81

DQVERBF Function

Returns the version of Blue Fusion.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQVERBF ()

Details

The DQVERBF function returns the version number of Blue Fusion that has been loaded by the SAS Data Quality product.

Example

The following example returns the Blue Fusion version.

```
version=DQVERBF ();
```

DQVERQKB Function

Returns the version of the currently loaded QKB.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQVERQKB ()

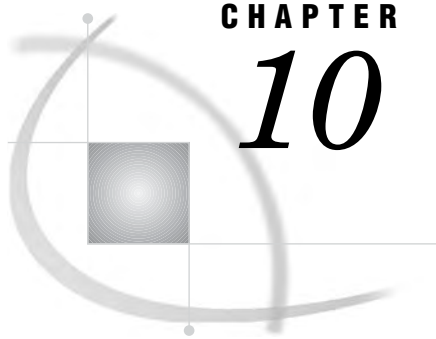
Details

The DQVERQKB function returns a five-character string containing 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

The following example returns the version of the currently loaded QKB.

```
version= QVERQKB ();
```

CHAPTER

10

SAS Data Quality Server System Options

SAS Data Quality Server System Options 107

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. See “%DQLOAD AUTOCALL Macro” on page 53.

Note: It is not recommended that you specify these system options by any means other than invoking the AUTOCALL macro %DQLOAD. Failure to use %DQLOAD or misapplied use of default settings for these system options can result in data that is cleansed with inappropriate locales. Δ

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.

DQLOCALE

Specifies an ordered list of locales.

Requirements: You must specify at least one locale.

Syntax

DQLOCALE= (locale1,<locale2><...localeN>)

LOCALE1, ...<LOCALE2, ...LOCALEN>

specifies an ordered list of locales. The list determines how the data is cleansed.

Locales are applied to the data in the order in which they are specified. All locales in the list must exist in the Quality Knowledge Base.

Details

The DQLOCALE= system option identifies the locales that are referenced during data cleansing. The order of the locales in the list affects the locale matching scheme of the DQMATCH procedure.

Unlike other system options, the value of the DQLOCALE= system option must be loaded into memory. Normally, system option values go into the system options table only. 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.

Note: It is recommended that you invoke the AUTOCALL macro %DQLOAD at the beginning of each data cleansing program or session. Failure to do so 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 53.

DQOPTIONS

Specifies SAS session parameters for data quality programs.

Restriction: You cannot create or apply schemes in BFD format in z/OS.

Valid in: The configuration file and as SAS start-up option.

Syntax

DQOPTIONS = (*label1=value1*)

DQSRVPROTOCOL

specifies the SAS Data Quality Server protocol. In operating environments, other than z/OS, the default SOAP protocol is recommended.

WIRELINE

specifies the Wireline protocol, which is required in the z/OS operating environment for DataFlux Integration Servers version 8.1.1 or newer. The Wireline protocol improves data transfer performance in z/OS. In the SAS Data Quality Server software, z/OS support encompasses the DQSRVSVC procedure and all functions.

Requirement: The Wireline protocol must be specified in the z/OS operating environment.

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.

DQSETUPLOC

Specifies the location of the SAS Data Quality Server setup file or the root directory of the Quality Knowledge Base.

Syntax

DQSETUPLOC=<'file-specification' | 'path-specification' >;

FILE-SPECIFICATION=*path.dqsetup.txt*

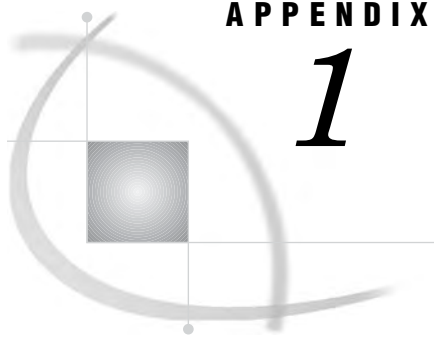
identifies the path and name of the setup file, named dqsetup.txt in the Windows and UNIX operating environments. The setup file defines the contents of the Quality Knowledge Base.

PATH-SPECIFICATION=*Quality Knowledge Base root-directory*

identifies the directory that is the root of the Quality Knowledge Base.

Restriction: When a path is specified instead of a file, a setup file is not referenced.

See Also: “Load and Unload Locales” on page 7.



Recommended Reading

Recommended Reading 111

Recommended Reading

The following titles are available through the Customer Care Portal at www.dataflux.com:

- DataFlux dfPower Studio Online Help
- DataFlux dfPower Studio Getting Started Guide
- DataFlux Integration Server User's Guide
- DataFlux Expression Language Reference

To learn about the data quality transformations in the SAS Data Integration Studio software, see that product's online Help for **Create Match Codes** and **Apply Lookup Standardization**.

For a complete list of SAS publications, refer to the current *SAS Publishing Catalog*. The catalog is produced twice a year. To order books or to receive a free copy of the catalog, write, call, or fax the Institute. Or access the online version of the *SAS Publishing Catalog* via the World Wide Web.

When you order a title, this provides you with the most current edition that is available.

SAS Institute
Fulfillment Services Dept.
SAS Campus Dr.
Cary, NC 27513
Telephone: 1-800-727-3228*
Fax: 919-677-8166
E-mail: sasbook@sas.com
Web site: www.sas.com/pubs

* For other SAS Institute business, call 919-677-8000.

Customers outside the U.S. should contact their local SAS office.

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.

Blue Fusion data format

a file format for schemes that can be created and applied in data quality software from SAS and from DataFlux (a SAS company). Schemes in Blue Fusion data format are sometimes referred to as BFD schemes. Schemes can also be created in SAS format.

case definition

a part of a locale that is referenced during data cleansing to impose a capitalization scheme on a character variable.

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 match codes from values from two or more input character variables in the same observation. A delimiter can be specified to separate the individual match codes in the concatenation.

compound match code

a match code that consists of a concatenation of match codes that are created for each token in a delimited or parsed string. Within a compound match code, individual match codes 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 definitions

are contained in the Quality Knowledge Base for a number of locales. Data definitions specify how categories of data are processed.

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 match codes internally to create clusters. All values in each cluster are then transformed to the standardization value that is specified in the scheme for each cluster.

delimiter

a character that separates words or phrases in 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. This 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

provide data definitions for a national language and geographical region. The locale reflects the language, local conventions, and culture for a geographic region. Local conventions can include specific formatting rules for dates, times, and numbers, and a currency symbol for the country or region. Collating sequences, paper sizes, and conventions for postal addresses and telephone numbers are also typically specified for each locale. 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 match codes or identical match code components. Identical match codes are assigned to clusters. See also match code, match code component, and cluster.

match code

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

match definition

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

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

in SAS data quality, a process that inserts into a character value a series of delimiters, as determined by a specified parse definition.

parse definition

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

parse token

a named element that can be assigned a value during parsing. Tokens are assigned values based on the specified parse definition. The value can then be manipulated using the name of the token. See also 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. See also delimited string.

Quality Knowledge Base

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

SAS data format

a file format for schemes that can be created and applied in data quality software from SAS and from DataFlux (a SAS company). Schemes in SAS data format are sometimes referred to as ??? schemes.

scheme

in SAS data quality, a reusable collection of match codes and standardization values that is applied to input character values for the purposes of transformation or analysis. Schemes can be created in Blue Fusion data format or SAS data format. See also Blue Fusion data format.

sensitivity

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

standardization definition

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

standardize

in SAS data quality, to impose a specified format on character values. Standardization definition is used to standardize 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. See also parse token.

transformation

in SAS Data Quality, a process that converts a group of similar data values to the single value that is most commonly present in the group.

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

! (exclamation point)
 as delimiter 21

A

analysis data sets 8, 33
 creating 37, 40
 ANALYSIS= option
 CREATE statement (DQSCHHEME) 37
 apply mode 9, 10
 APPLY statement, DQSCHHEME procedure 35
 authentication 46, 51, 100
 autocall macros 53

B

BFD format
 See Blue Fusion Data (BFD) format
 BFD option
 PROC DQSCHHEME statement 34
 BFDTOSAS option
 CONVERT statement (DQSCHHEME) 36
 blank spaces, removing 51
 blank values 20
 BLOCKSIZE= option
 PROC DQSRV SVC statement 50
 Blue Fusion Data (BFD) format 8, 9, 34
 converting SAS schemes to 8, 36
 creating BFD schemes 42
 version of BFD 104

C

CALL routines 58
 scheme CALL routines 63
 case definitions 17, 63
 case functions 60
 case of character values 63
 category names
 from character values 67
 character values
 after standardization 102
 case and standardization definitions 17
 case of 63
 category names from 67
 gender analysis, locale guess, and identification definitions 18
 inserting tokens into parsed values 82
 locale names from 68

 match codes from 71
 match codes from parsed values 73
 parsed 75, 76
 pattern analysis definitions 18
 pattern analysis from input values 83
 tokens from 103
 tokens from parsed values 81
 updated parsed values 82
 character variables
 transforming 8
 cleaning up jobs and logs 47
 cleansing data
 real-time 49
 cluster numbers 19
 assigning 23
 creating 24
 CLUSTER= option
 PROC DQMATCH statement 20
 CLUSTER_BLANKS option
 PROC DQMATCH statement 20
 clusters 12
 householding 12
 minimal sensitivity 27
 mixed sensitivity 25
 with exact criteria 13
 with multiple CRITERIA statements 29
 CLUSTERS_ONLY option
 PROC DQMATCH statement 20
 columns, input and output 51
 composite match codes 12, 24
 CONDITION= option
 CRITERIA statement (DQMATCH) 22
 configuration
 SAS sessions for data quality 4
 CONVERT statement, DQSCHHEME procedure 36
 converting schemes 8, 36
 count of locale definitions 70
 CREATE statement, DQSCHHEME procedure 37
 CRITERIA statement, DQMATCH procedure 22
 clustering with exact criteria 13
 clustering with multiple 29

D

data cleansing
 real-time 49
 specifying definitions in programs 5
 DATA= option
 PROC DQMATCH statement 21

- PROC DQSCHEME statement 35
- PROC DQSRVSV statement 50
- data quality
 - configuring SAS sessions for 4
 - SAS session parameters for programs 108
- data sets
 - analysis 8, 33, 37, 40
 - input 52
 - job status 45, 46, 47
 - output 52
 - scheme 8, 9, 33, 37
- data transfer 4
- data values, similar
 - transforming 33
- DataFlux dfPower Architect 1
 - creating jobs and services 5
 - job identifiers 92
 - job information 45
 - running jobs 92
 - running real-time services 49
 - running services 52
- DataFlux dfPower Profile 1
 - creating jobs and services 5
 - job information 45
 - running file-type jobs 97
 - running repository-type jobs 99
- DataFlux dfPower Studio 34
- DataFlux Integration Server 1
 - authenticating users 46, 51, 100
 - cleaning up jobs and logs 47
 - copying log files 93
 - deleting log files 94
 - functions 60
 - host machine 50
 - host of 46
 - input and output columns 51
 - job status 95
 - jobs and services 5
 - passwords 6
 - port number 46, 51
 - running dfPower Architect jobs 92
 - running file-type Profile jobs 97
 - running jobs and services 6
 - running real-time services 49
 - running repository-type Profile jobs 99
 - security 46
 - service identification 51
 - terminating jobs 96
 - version of 101
- date standardization
 - in EN locale 17
- default length of parsed input 78
- definitions
 - available in locales 5
 - case 63
 - case and standardization 17
 - date standardization in EN locale 17
 - displaying information about locale 5
 - gender 65
 - gender analysis, locale guess, and identification 18
 - global parse 16
 - locale 15
 - match 16
 - parse 15
 - pattern analysis 18
 - revealing or hiding non-surfaced 74
 - scheme-build match 16
 - specifying in data cleansing programs 5
- delimiter
 - exclamation point as 21
- DELIMITER option
 - PROC DQMATCH statement 21
- DELIMSTR option
 - CRITERIA statement (DQMATCH) 22
- DQCASE function 63
- DQGENDER function 64
- DQGENDERINFOGET function 65
- DQGENDERPARSED function 66
- DQIDENTIFY function 67
- %DQLOAD autocall macro 53
- DQLOCALE= system option 107
- DQLOCALEGUESS function 68
- DQLOCALEINFOGET function 69
- DQLOCALEINFOLIST function 70
- DQMATCH function 71
- DQMATCH procedure 19
 - clustering with minimal sensitivity 27
 - clustering with mixed sensitivities 25
 - clustering with multiple CRITERIA statements 29
 - CRITERIA statement 22, 29
 - examples 24
 - generating composite match codes 24
 - generating multiple simple match codes 30
 - householding 12
 - match codes for parsed values 28
 - matching values with mixed sensitivity levels 25
 - PROC DQMATCH statement 20
 - syntax 20
- DQMATCHINFOGET function 72
- DQMATCHPARSED function 73
- DQOPTIONS= system option 108
 - configuring SAS sessions 4
- DQOPTSURFACE function 74
- DQPARSE CALL routine 75
- DQPARSE function 76
- DQPARSEINFOGET function 77
- DQPARSEINPUTLEN function 78
- DQPARSERESLIMIT function 79
- DQPARSESCORDEPTH function 80
- DQPARSETOKENGET function 81
- DQPARSETOKENPUT function 82
- DQPATTERN function 83
- %DQPULOC autocall macro 5, 54
- DQSCHEME procedure 33
 - APPLY statement 35
 - applying schemes 42
 - CONVERT statement 36
 - CREATE statement 37
 - creating analysis data sets 40
 - creating BFD schemes 42
 - creating schemes 41
 - examples 40
 - PROC DQSCHEME statement 34
 - scheme-build match definitions and 16
 - syntax 34
- DQSCHEMEAPPLY CALL routine 84
- DQSCHEMEAPPLY function 88
- DQSETUPLOC= system option 109
- DQSRVADM procedure 45
 - cleaning up jobs and logs 47
 - examples 47
 - job status data sets 46, 47

- PROC DQSRVADM statement 46
 - security and 46
 - syntax 46
- DQSRVARCHJOB function 92
- DQSRVCOPYLOG function 93
- DQSRVDELETELOG function 94
- DQSRVJOBSTATUS function 95
- DQSRVKILLJOB function 96
- DQSRVPROFJOBFILE function 97
- DQSRVPROFJOBREP function 99
- DQSRVSVC procedure 49
 - examples 52
 - input and output data sets 52
 - performance of 4
 - PROC DQSRVSVC statement 50
 - syntax 50
 - timeout 51
- DQSRVUSER function 100
- DQSRVVER function 101
- DQSTANDARDIZE function 102
- DQTOKEN function 103
- %DQUNLOAD autocall macro 56
- DQVERBF function 104
- DQVERQKB function 105

E

- element mode 9
- EN locale
 - date standardization in 17
- encoded passwords 6
- EXACT option
 - CRITERIA statement (DQMATCH) 23
- exclamation point
 - as delimiter 21

F

- file-type DataFlux dfProfile jobs 97
- functions 58
 - case 60
 - DataFlux Integration Server 60
 - gender analysis, locale guessing, and identification 60
 - listed alphabetically 58
 - listed by category 60
 - matching 61
 - parsing 61
 - pattern analysis 61
 - reporting 62
 - scheme 63
 - standardization 63

G

- gender analysis definitions 18
- gender analysis functions 60
- gender definitions
 - parse definitions associated with 65
- gender determination
 - from name of an individual 64
 - from parsed name 66
- global parse definitions 16

H

- hiding non-surfaced definitions 74

- host
 - for DataFlux Integration Server 46
- host machine
 - DataFlux Integration Server 50
- HOST= option
 - PROC DQSRVADM statement 46
 - PROC DQSRVSVC statement 50
- householding 12

I

- identification definitions 18
- identification functions 60
- IGNORE_CASE option
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 39
- IN= option
 - CONVERT statement (DQSCHEME) 37
- INCLUDE_ALL option
 - CREATE statement (DQSCHEME) 37
- input
 - default length of parsed 78
 - parsed values 15
- input character values
 - pattern analysis from 83
- input columns 51
- input data sets
 - DQSRVSVC procedure 52
- installation 5

J

- job identifiers
 - DataFlux dfPower Architect jobs 92
 - file-type Profile jobs 97
 - repository-type Profile jobs 99
- job status data sets 45, 46
 - generating 47
 - location of 46
- jobs 5
 - cleaning up 47
 - copying log files 93
 - creating 5
 - deleting log files 94
 - information about 45
 - logs for 6
 - running 6
 - running DataFlux dfPower Architect jobs 92
 - running file-type Profile jobs 97
 - running repository-type Profile jobs 99
 - status of 95
 - terminating 6, 96

K

- killing jobs 6, 96

L

- length
 - default length of parsed input 78
- loading locales 7
 - into memory 53
- Local Process group 1
- locale definitions 15
 - case and standardization 17

- count of 70
- date standardization in EN locale 17
- gender analysis, locale guess, and identification 18
- global parse 16
- match 16
- name of 70
- parse 15
- pattern analysis 18
- scheme-build match 16
- locale guess definitions 18
- locale guessing functions 60
- locale names
 - from character values 68
- LOCALE= option
 - APPLY statement (DQSCHHEME) 35
 - CREATE statement (DQSCHHEME) 38
 - PROC DQMATCH statement 21
- locales
 - definitions available in 5
 - displaying information about 5
 - displaying information in SAS log 54
 - getting information about 69
 - loading and unloading 7
 - loading into memory 53
 - ordered list of 107
 - unloading to increase free memory 56
 - updating 7
- log files
 - copying 93
 - deleting 94
- logs
 - cleaning up 47
 - displaying locale information 54
 - for jobs and services 6
- lookup method 9, 10, 36, 39

M

- macros, autocall 53
- MACROS= option
 - PROC DQSRVSV statement 50, 52
- match codes 19
 - composite 12, 24
 - creating 11, 22
 - creating for parsed values 28
 - from character values 71
 - from parsed character values 73
 - generating multiple simple codes 30
 - length of 12
 - match definitions and 16
 - sensitivity 13
 - simple 11, 30
 - truncation of 12
- MATCH-DEFINITION= option
 - APPLY statement (DQSCHHEME) 35
- match definitions 16
 - parse definitions associated with 72
 - scheme-build 16
- MATCHCODE= option
 - CRITERIA statement (DQMATCH) 23
 - PROC DQMATCH statement 21
- MATCHDEF option
 - CRITERIA statement (DQMATCH) 23
 - CREATE statement (DQSCHHEME) 38
- matching functions 61

- matching values
 - default sensitivity 24
 - minimal sensitivity 27
 - mixed sensitivity 25
- memory
 - loading locales into 53
 - unloading locales from 56
- meta options 9
- MISSINGVARVOK option
 - PROC DQSRVSV statement 50
- mode of scheme application 35, 38
- MODE=ELEMENT option
 - APPLY statement (DQSCHHEME) 35
 - CREATE statement (DQSCHHEME) 38

N

- named tokens 15, 77
- names
 - gender determination and 64, 66
 - locale names from character values 68
 - of locale definitions 70
 - of parse definitions 65, 72
 - parsed 66
- NO CLUSTER_BLANKS option
 - PROC DQMATCH statement 20
- NOBFD option
 - PROC DQSCHHEME statement 34
- NODELIMITER option
 - PROC DQMATCH statement 21
- non-surfaced definitions
 - revealing or hiding 74
- NOPRINT option
 - PROC DQSRVSV statement 51

O

- option-value pairs 4
- OUT= option
 - CONVERT statement (DQSCHHEME) 37
 - PROC DQMATCH statement 21
 - PROC DQSCHHEME statement 35
 - PROC DQSRVADM statement 46
 - PROC DQSRVSV statement 51
- output columns 51
- output data sets
 - DQSRVSV procedure 52

P

- parse definitions 15
 - associated with gender definitions 65
 - associated with match definitions 72
 - global 16
 - name of 65, 72
 - token names in 77
- parsed character values 75, 76
 - inserting tokens into 82
 - match codes from 73
 - tokens from 81
 - updated 82
- parsed input 15
 - default length of 78
- parsed names
 - gender determination from 66

- parsed values
 - match codes for 28
 - updated 82
 - parsing
 - resource limit during 79
 - parsing functions 61
 - parsing scores
 - searching for 80
 - PASSWORD= option
 - PROC DQSRVADM statement 46
 - PROC DQSRVSV statement 51
 - passwords
 - for DataFlux Integration Server 6
 - pattern analysis
 - from input character values 83
 - pattern analysis definitions 18
 - pattern analysis functions 61
 - performance
 - DQSRVSV procedure 4
 - phrase mode 9
 - PHRASE option
 - APPLY statement (DQSCHEME) 35
 - CREATE statement (DQSCHEME) 38
 - port number 46, 51
 - PORT= option
 - PROC DQSRVADM statement 46
 - PROC DQSRVSV statement 51
 - PROC DQMATCH statement 20
 - PROC DQSCHEME statement 34
 - PROC DQSRVADM statement 46
 - PROC DQSRVSV statement 50
 - Profile jobs 5
- Q**
- Quality Knowledge Base (QKB) 1
 - location of root directory 109
 - setup file and 3
 - version of currently loaded 105
- R**
- real-time data cleansing 49
 - reporting functions 62
 - repository-type Profile jobs 99
 - resource limit
 - during parsing 79
 - revealing non-surfaced definitions 74
 - root directory
 - specifying location of 109
 - running jobs 6
 - DataFlux dfPower Architect 92
 - file-type Profile jobs 97
 - repository-type Profile jobs 99
 - running services 6
 - real-time 49
- S**
- SAS Data Quality Server 1
 - concepts 3
 - functional overview 1
 - installing 5
 - setup file 3
 - updating 5
 - SAS format schemes
 - converting BFD format to 8, 36
 - SAS language elements 1
 - SAS log
 - displaying locale information 54
 - SAS sessions
 - configuring for data quality 4
 - parameters for data quality programs 108
 - SASTOBFD option
 - CONVERT statement (DQSCHEME) 36
 - scheme-build match definitions 16
 - scheme CALL routines 63
 - scheme data sets 33
 - applying schemes 9
 - creating 37
 - format of 8, 9
 - scheme functions 63
 - SCHEME= option
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 38
 - SCHEME_LOOKUP=EXACT option
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 39
 - schemes 8, 33
 - analysis data sets 8
 - apply mode 9, 10
 - applying 9, 42, 84, 88
 - applying to transform values of a single variable 35
 - converting between formats 8, 36
 - creating 8, 41
 - creating BFD schemes 42
 - format of 34
 - meta options 9
 - mode of application 35, 38
 - returning transformation flag 84
 - returning transformed value 84, 88
 - searching
 - for parsing scores 80
 - security
 - DQSRVADM procedure and 46
 - sensitivity level 13
 - analysis data sets and 8
 - default value 24
 - match codes and 19, 23
 - match definitions and 16
 - meta options and 10
 - minimal 27
 - mixed 25
 - SENSITIVITY= option
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 39
 - CRITERIA statement (DQMATCH) 23
 - Server Process group 1
 - SERVICE= option
 - PROC DQSRVSV statement 51
 - SERVICEINFO option
 - PROC DQSRVSV statement 51
 - services 5
 - creating 5
 - identification of 51
 - logs for 6
 - running 6, 52
 - running real-time 49
 - setup file 3
 - editing 4
 - specifying location of 109

- similar data values
 - transforming 33
- simple match codes 11
 - generating multiple 30
- standardization 102
 - date standardization in EN locale 17
 - standardization definitions 17
 - standardization functions 63
- system options 107
 - setting values 53

T

- terminating jobs 6, 96
- TIMEOUT option
 - PROC DQSRVSVSVC statement 51
 - PROC DQSRVSVSVC statement 51
- token names 15, 77
- tokens
 - from character values 103
 - from parsed character values 81
 - inserting into parsed character values 82
 - updated parsed character values 82
- transcoding errors 5
- transferring data 4
- transformation flags 84
- transforming
 - applying schemes to transform values of a single variable 35
 - character values 8
 - returning transformed values after applying a scheme 84, 88
 - similar data values 33
- TRIM option
 - PROC DQSRVSVSVC statement 51
- truncated match codes 12

U

- unloading locales 7, 56
- updated parsed values 82
- updated tokens 82
- updating
 - locales 7
 - SAS Data Quality Server 5
- USE_MATCHDEF option
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 39
- user authentication 46, 51, 100
- USERID= option
 - PROC DQSRVSVSVC statement 51

V

- VAR option
 - CRITERIA statement (DQMATCH) 22
 - APPLY statement (DQSCHEME) 36
 - CREATE statement (DQSCHEME) 39
- variables
 - applying schemes to transform values of a single variable 35
 - transforming character variables 8
- version
 - Blue Fusion Data 104
 - currently loaded QKB 105
 - DataFlux Integration Server 101

W

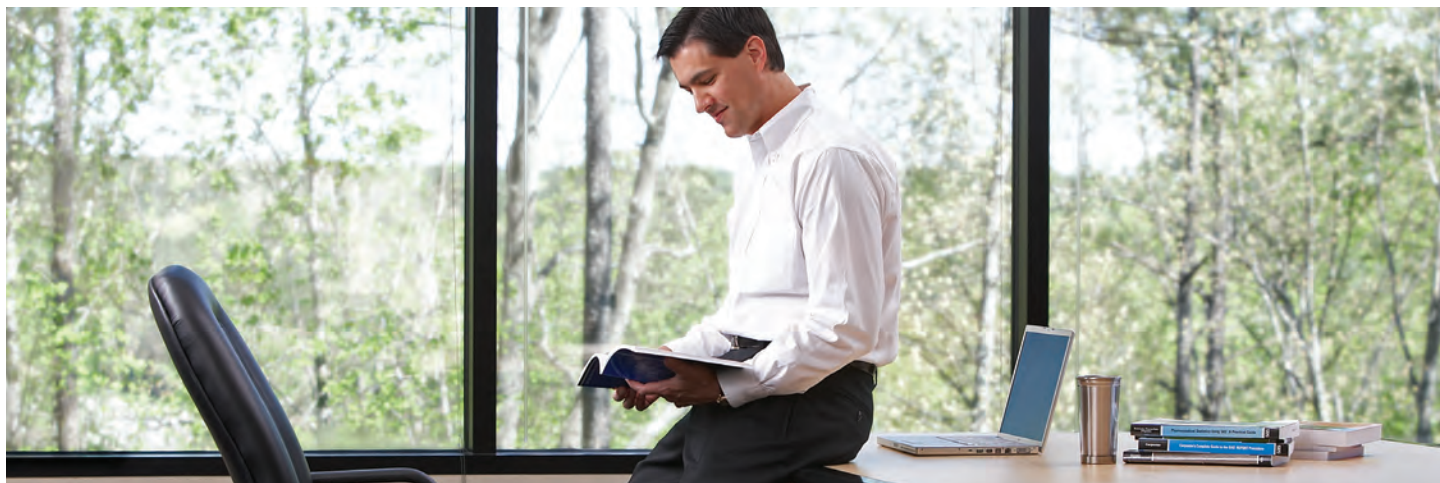
- Wireline protocol 6

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to yourturn@sas.com. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to suggest@sas.com.

SAS® Publishing Delivers!



SAS Publishing provides you with a wide range of resources to help you develop your SAS software expertise. Visit us online at support.sas.com/bookstore.

SAS® PRESS

SAS Press titles deliver expert advice from SAS® users worldwide. Written by experienced SAS professionals, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® DOCUMENTATION

We produce a full range of primary documentation:

- Online help built into the software
- Tutorials integrated into the product
- Reference documentation delivered in HTML and PDF formats—free on the Web
- Hard-copy books

support.sas.com/documentation

SAS® PUBLISHING NEWS

Subscribe to SAS Publishing News to receive up-to-date information via e-mail about all new SAS titles, product news, special offers and promotions, and Web site features.

support.sas.com/spn

SOCIAL MEDIA: JOIN THE CONVERSATION!

Connect with SAS Publishing through social media. Visit our Web site for links to our pages on Facebook, Twitter, and LinkedIn. Learn about our blogs, author podcasts, and RSS feeds, too.

support.sas.com/socialmedia



**THE
POWER
TO KNOW®**