

SAS[®] 9.3 Data Quality Server Reference



The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 Data Quality Server Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Data Quality Server Reference

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

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

For a hardcopy 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.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

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 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/publishing 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 in SAS 9.3 Data Quality Server</i>	<i>v</i>
<i>Recommended Reading</i>	<i>ix</i>
Chapter 1 • Overview of SAS 9.3 Data Quality Server	1
Server and Platform Interactions	1
Chapter 2 • Concepts	3
SAS Data Quality Server Concepts	3
DataFlux Jobs and Services	5
Load and Unload Locales	6
Schemes	7
Create Match Codes	10
Clusters	12
Sensitivity	13
Chapter 3 • Locale Definitions	15
Locale Definitions	15
Chapter 4 • DMSRVADM Procedure	21
Overview: DMSRVADM Procedure	21
Syntax: DMSRVADM Procedure	21
The Job Status Data Set	22
Security	23
Examples: DMSRVADM Procedure	23
Chapter 5 • DMSRVDATASVC Procedure	25
Overview: DMSRVDATASVC Procedure	25
Syntax: DMSRVDATASVC Procedure	25
The Input and Output Data Sets	28
Examples: DMSRVDATASVC Procedure	29
Chapter 6 • DMSRVPROCESSSVC Procedure	31
Overview: DMSRVPROCESSSVC Procedure	31
Syntax: DMSRVPROCESSSVC Procedure	31
The Input and Output Data Sets	33
Examples: DMSRVPROCESSSVC Procedure	34
Chapter 7 • DQMATCH Procedure	35
Overview: DQMATCH Procedure	35
Syntax: DQMATCH Procedure	36
Examples: DQMATCH Procedure	40
Chapter 8 • DQSCHEME Procedure	49
Overview: DQSCHEME Procedure	49
Syntax: DQSCHEME Procedure	50
Examples: DQSCHEME Procedure	56
Chapter 9 • AUTOCALL Macros	61
Dictionary	61

Chapter 10 • Functions and CALL Routines	65
Overview	66
Functions Listed Alphabetically	66
Functions Listed by Category	68
Dictionary	72
Chapter 11 • SAS Data Quality Server System Options	119
SAS Data Quality Server System Options	119
Dictionary	120
Appendix 1 • Deprecated Language Elements	123
Dictionary	123
Glossary	127
Index	131

What's New in SAS 9.3 Data Quality Server

Overview

SAS 9.3 Data Quality Server provides procedures and functions that enable you to administer and run jobs and services on DataFlux Data Management Platform from DataFlux (a wholly owned subsidiary of SAS Institute Inc.). This release of SAS Data Quality Server also contains language elements specific to data extraction definitions.

SAS 9.3 Data Quality Server includes the following enhancements:

- New functions and procedures that support DataFlux Data Management Server 2.1 and later.
- Deprecation or replacement of legacy functions and procedures to provide a migration path from DataFlux dfPower Studio to DataFlux Data Management Studio. Some utility functions have also been renamed to reflect this.
- Inclusion of functions to enable you to process data using extraction definitions.

New Functions and Procedures

Many of the functions and procedures added for this release of SAS Data Quality Server support DataFlux Data Management Server 2.1 and later. These new functions and procedures begin with DM and are not compatible with DataFlux Integration Servers or DataFlux dfPower Studio applications. Additional functions and procedures included in this release of SAS Data Quality Server enable you to customize extraction definitions for use with applications such as Blue Fusion 8.2 and later. Each function and procedure is listed in the following sections along with a brief description.

Server Process and Utility Functions

The following functions are new in SAS 9.3 Data Quality Server and operate with DataFlux Data Management Server 2.1 and later:

- The DMSRVBATCHJOB function runs a DataFlux Data Management batch job or a DataFlux Data Management process job and returns a job or process identifier, as appropriate. [For more information, see DMSRVBATCHJOB Function.](#)
- The DMSRVCOPYLOG function copies a log file from a DataFlux Data Management Server to the local machine. [For more information, see DMSRVCOPYLOG Function.](#)

- The DMSRVDELETELOG function deletes a job's log file from the DataFlux Data Management Server. [For more information, see DMSRVDELETELOG Function.](#)
- The DMSRVJOBSTATUS function returns the status of a job that was submitted to a DataFlux Data Management Server. [For more information, see DMSRVJOBSTATUS Function.](#)
- The DMSRVKILLJOB Function terminates a job that is running on a DataFlux Data Management Server. [For more information, see DMSRVKILLJOB Function.](#)
- The DMSRVPROFILEJOB function generates a profile from a Data Management repository. [For more information, see DMSRVPROFILEJOB Function.](#)
- The DMSRVUSER function sets the user name and password relative to the DataFlux Data Management Server. [For more information, see DMSRVUSER Function.](#)
- The DMSRVVER function returns the version of the DataFlux Data Management Server. [For more information, see DMSRVVER Function.](#)

Extraction Definition Support Functions

Extraction support functions that are new to SAS 9.3 Data Quality Server include the following:

- The DQEXTINFOGET function returns the token names in an extraction definition. [For more information, see DQEXTINFOGET Function.](#)
- The DQEXTRACT function returns an extracted character value. [For more information, see DQEXTRACT Function.](#)
- The DQEXTTOKENGET function returns a token from an extraction character value. [For more information, see DQEXTTOKENGET Function.](#)
- The DQEXTTOKENPUT Function inserts a token into an extraction character value and returns the updated extraction character value. [For more information, see DQEXTTOKENPUT Function.](#)

To support the new extraction definition functions, the macro, %DQPUTLOC, now lists extraction definitions along with all other types of definitions. [For more information, see %DQPUTLOC Macro.](#)

Server Process Procedures

Procedures new in SAS 9.3 Data Quality Server that work exclusively with DataFlux Data Management Server 2.1 and later include the following:

- PROC DMSRVADM returns a list of jobs running on a DataFlux Data Management Server. [For more information, see PROC DMSRVADM.](#)
- PROC DMSRVDATASVC runs DataFlux Architect and data services. [For more information, see PROC DMSRVDATASVC.](#)

- PROC DMSRVPROCESSSSVC runs DataFlux Data Management process services. For more information, see [PROC DMSRVPROCESSSSVC](#).

Interoperability and Legacy Support

SAS 9.3 Data Quality Server is an optimized implementation designed to work as part of a SAS offering for an integrated system or as a component within a DataFlux Data Management Platform configuration. As a component within a DataFlux Data Management Platform configuration, several functions and procedures (identified by names beginning with DM) work exclusively with DataFlux Data Management Server 2.1 and later.

Whereas SAS 9.3 continues to support all DQSRV functions and procedures, the operations will attempt to execute using a Data Management Server by default.

Remember the following as you work with SAS Data Quality 9.3:

- DMSRV functions and procedures issue an error if a host and port that are serving up a DataFlux Integration Server are specified. This code works with DataFlux Data Management Servers only.
- All DQSRV functions and procedures run on any version of either DataFlux Data Management Server or DataFlux Integration Server, but will issue a deprecation warning.

Recommended Reading

Here is the recommended reading list for this title:

- The online Help for SAS products describing SAS 9.3 Data Quality Server language reference elements.
- Documentation specific to DataFlux, available through MyDataFlux Portal at the DataFlux Web site:

<http://www.dataflux.com/MyDataFlux-Portal.aspx>

- *DataFlux Migration Guide*, Version 2.1 and later. Describes how to migrate your content from DataFlux dfPower Studio to DataFlux Data Management Platform.
- *DataFlux Expression Language Reference Guide*, Version 2.1 and later.
- *DataFlux Data Management Studio User's Guide*, Version 2.1 and later.
- *DataFlux Data Management Server User's Guide*, Version 2.1 and later.
- Depending on your role, you might want to review the Federation and Authentication server documents for users or administrators.
- 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. Also, you can access reference material related to Data Quality under the Data Quality folder in the Transformations tree of the SAS Data Integration Studio software.

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

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

x *Recommended Reading*

Chapter 1

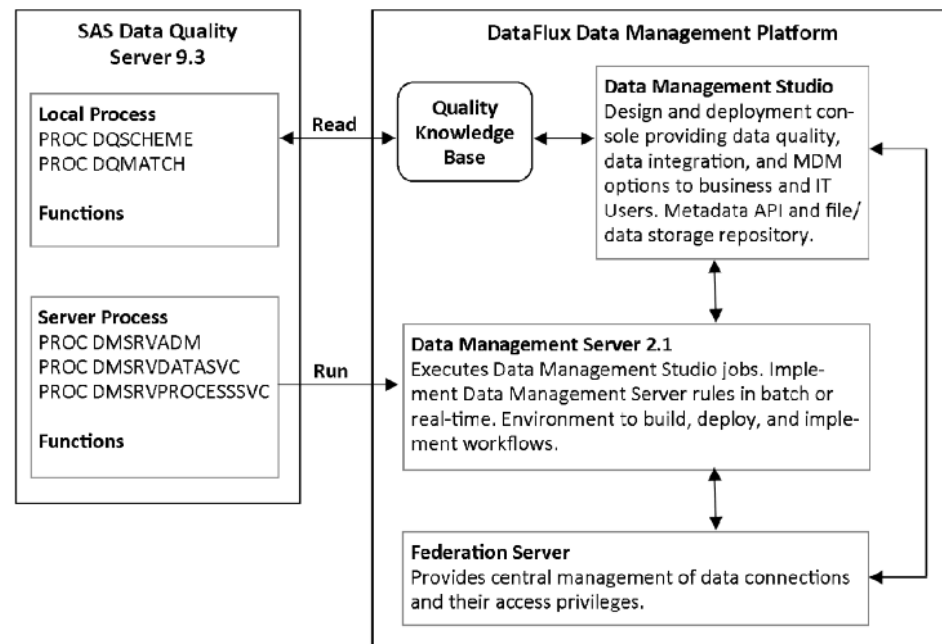
Overview of SAS 9.3 Data Quality Server

Server and Platform Interactions	1
SAS Data Quality Server Operation	1
SAS Data Quality Server Integration Components	2

Server and Platform Interactions

SAS Data Quality Server Operation

The SAS 9.3 Data Quality Server is delivered with a sample Quality Knowledge Base (QKB) and consists of SAS language elements that perform data quality operations (matching, standardization, and so on), as well as elements interoperating with a DataFlux Data Management Server. As part of the installation (and subsequent maintenance procedures), you will replace the sample QKB with the latest QKB available from the DataFlux Web site. The SAS 9.3 Data Quality Server communicates with the DataFlux Data Management Platform to provide an integrated system. The integrated system manages data assets through data quality, data integration, master data management (MDM), and federated data access.

Figure 1.1 Server and Platform Interactions (not showing DataFlux Authentication Server)

A DataFlux Data Management Platform with a DataFlux Data Management Platform 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 third-party products using the command-line interface that is available for the DataFlux server (or servers). You can also choose to add the DataFlux Data Management Platform after purchasing the SAS Data Quality Server software.

SAS Data Quality Server Integration Components

Proper interoperability between SAS Data Quality Server and DataFlux Data Management Platform is contingent upon several key components being installed or configured (or both).

- **License.** You must have the correct license to be able to use SAS 9.3 Data Quality Server with DataFlux Data Management Platform.
- **System Option Specifications.** SAS Data Quality Server is delivered with server and process language elements. Depending on your implementation and your operating system, you might need to rely on the system option elements to configure communication parameters with the QKB.
- **Current QKB.** The QKB should be updated each time a new one is released and made available through the DataFlux Web site.
- **Session and Definition Specifications.** You must make sure that your SAS Session is properly configured (that is, specify all options and file locations as needed) for Data Quality. In addition, if necessary, you must specify definitions in your SAS data cleansing program.

Each item listed above is discussed in detail in [“Concepts” on page 3](#).

Chapter 2

Concepts

SAS Data Quality Server Concepts	3
Specifying the Quality Knowledge Base	3
Configure Your SAS Session for Data Quality	4
Specify Definitions in SAS Data Cleansing Programs	4
Considerations for Installing and Updating the Software	4
DataFlux Jobs and Services	5
Creating Jobs, Servicing Jobs, and Writing Information to the SAS Log	5
Running Jobs and Services on a DataFlux Data Management Server	5
DataFlux Data Management Server Passwords	6
Load and Unload Locales	6
Schemes	7
Transport the Values of a Character Variable	7
Creating Schemes	7
Analysis Data Sets	8
Applying Schemes	8
Meta Options	9
Create Match Codes	10
Overview	10
How to Create a Match Code	11
Match Code Length	12
Clusters	12
Householding with the DQMATCH Procedure	12
Clustering with Exact Criteria	13
Sensitivity	13

SAS Data Quality Server Concepts

Specifying the Quality Knowledge Base

To access a Quality Knowledge Base, SAS programs reference the value set in the DQSETUPLOC= system option. The value of the system option is the path to the root directory of 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 each location with the DQSETUPLOC= option.

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. 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 enables two option-value pairs. The DQSRVPROTOCOL=WIRELINE pair improves the performance of the DMSRVDATASVC and DMSRVPROCESSSVC procedures by streamlining data transfer to and from the DataFlux Data Management 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 120 for additional information.

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. To find out the names of the definitions that are available in a particular local, use the “%DQPUTLOC AUTOCALL Macro” on page 62.

To display information about a locale that is currently loaded into memory, use the “DQLOCALEINFOGET Function” on page 92.

To display a list of definitions in a specified locale, use the “DQLOCALEINFOLIST Function” on page 92 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, available through MyDataFlux Portal at the DataFlux Web site, <http://www.dataflux.com/MyDataFlux-Portal.aspx>.

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 to 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 Data Management Studio software. If you install your updated Quality Knowledge Base in a new location, make sure you reference the Quality Knowledge Base by specifying the appropriate value in the DQSETUPLOC= system option.

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.

DataFlux Jobs and Services

Creating Jobs, Servicing Jobs, and Writing Information to the SAS Log

Jobs and services that are run on a DataFlux Data Management 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 Data Management Server, use the DataFlux Data Management Studio application. You can also use DataFlux Data Management Studio to create jobs to analyze the quality of your data. The DMSRVPROFILEJOB function generates a profile. The DMSRVBATCHJOB function runs data jobs as well as process jobs.

Use DataFlux Data Management Studio software to create jobs and services using a drag-and-drop interface. You can trigger the DataFlux Data Management Studio jobs with the function DMSRVBATCHJOB. You can run DataFlux Data Management services with the DMSRVDATASVC and DMSRVPROCESSVC procedures.

Jobs and services that run on the DataFlux Data Management Servers generate information that is written to the log. You can copy server logs using the DMSRVCOPYLOG function. Based on the information returned by the DMSRVJOBSTATUS function, you can terminate jobs using the DMSRVKILLJOB function.

Running Jobs and Services on a DataFlux Data Management Server

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

1. For any client session using Wireline and in the z/OS operating environment, for DataFlux Data Management Servers, configure your DataFlux Data Management Server to use the Wireline protocol. This is described in the *DataFlux Data Management Server: User's Guide*. The Wireline protocol improves data transfer performance.
2. In the z/OS operating environment, ensure that your SAS session is configured to use the Wireline protocol. This is the default setup and can be manually configured as described in [“Configure Your SAS Session for Data Quality” on page 4](#).
3. Create jobs and services using the DataFlux Data Management Studio software.
4. Upload the jobs to the DataFlux Data Management Server using the DataFlux Data Management Server Manager.
5. Create and run the SAS programs that execute or trigger the jobs and services on the DataFlux Data Management Server.

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

Refer to [Chapter 4, “DMSRVADM Procedure,” on page 21](#) and [Chapter 5, “DMSRVDATASVC Procedure,” on page 25](#) for additional information.

DataFlux Data Management Server Passwords

If security is implemented on your DataFlux Data Management 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 Data Management Server.

This example shows how to encode a password and use that password in a call to the DMSRVDATASVC 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 Data Management Server */
proc dmsrvdatasvc
  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 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 Data Management Server.

Before you run data-cleansing programs in SAS, load locales into memory as described in [“%DQLOAD AUTOCALL Macro” on page 61](#). Use the DQSETUPLOC= option to specify the location of the Quality Knowledge Base. Use the DQLOCALE= option to specify an ordered list of locales.

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 91 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](#)” on page 61 to change the values of the two data quality system options.

If you change locale files in the Quality Knowledge Base using the DataFlux Data Management Studio software, you must reload macros into memory with the %DQLOAD AUTOCALL 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 64.

New locales, and updates to existing locales, are provided periodically by DataFlux in the form of a new Quality Knowledge Base. You can download Quality Knowledge Bases from the MyDataFlux Portal at the DataFlux Web site: <http://www.dataflux.com/MyDataFlux-Portal.aspx>.

Schemes

Transport the Values of a Character Variable

A scheme is a 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 8](#), “[DQSCHEME Procedure](#),” on page 49. Use the information found in “[DQSCHEMEAPPLY Function](#)” on page 107 and “[DQSCHEMEAPPLY CALL Routine](#)” on page 110 to apply schemes as well.

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

Creating 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.

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 Data Management 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 Data Management 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.

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 [“Creating Schemes” on page 7](#) for additional information.

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, as described in [APPLY Statement on page 51](#). You can also use 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 CALL Routine” on page 110](#) for additional information.

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 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.

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

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.

The meta options are stored using:

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

lookup-method

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

IC specifies that SCHEME_LOOKUP=IGNORE_CASE.

UM specifies that SCHEME_LOOKUP=USE_MATCHDEF. 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

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

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

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

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

Create Match Codes

Overview

Match codes are encoded representations of character values that are used for analysis, transformation, and standardization of data. Use the following procedures and functions to create match codes:

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 [Chapter 7, “DQMATCH Procedure,” on page 35](#) for additional information.

The DQMATCH Function

generates match codes for tokens that have been parsed from a variable. See [“DQMATCH Function” on page 94](#) for additional information.

The DQMATCHPARSED Function

See [“DQMATCHPARSED Function” on page 96](#) for additional information.

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 50 and 95 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 7](#). 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 to Create a Match Code

You can create two types of match codes:

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

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 DQSCHEME 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 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 groups match 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 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 that matches the match

codes from another source row. The code and output for this example are provided in [“Example 5: Clustering with Multiple CRITERIA Statements” on page 46](#).

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 and the other using a match code generated from customer names. The syntax of the EXACT= option is provided in [Chapter 7, “DQMATCH Procedure,” on page 35](#).

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](#) for additional information.

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

Locale Definitions	15
Parse Definitions	15
Global Parse Definitions	16
Extraction Definitions	16
Match Definitions	17
Scheme Build Match Definitions	17
Case and Standardization Definitions	18
Standardization of Dates in the EN Locale	18
Gender Analysis, Locale Guess, and Identification Definitions	19
Pattern Analysis Definitions	19

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 routine and functions:

- [“DQPARSE CALL Routine” on page 99](#)
- [“DQPARSEINFOGET Function” on page 100](#)
- [“DQTOKEN Function” on page 115](#)
- [“DQPARSETOKENGET Function” on page 103](#)
- [“DQPARSETOKENPUT Function” on page 104](#)

For a brief example of how tokens are assigned and used, see [“Specify Definitions in SAS Data Cleansing Programs” on page 4](#).

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 as follows:

```
NAME PREFIX
empty
```

```

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

```

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

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.

Extraction Definitions

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

Extraction definitions and tokens are referenced by the following functions:

- [“DQEXTRACT Function” on page 84](#)
- [“DQEXTINFOGET Function” on page 83](#)
- [“DQEXTTOKENGET Function” on page 85](#)
- [“DQEXTTOKENPUT Function” on page 86](#)

For a brief example of how tokens are assigned and used, see [“Specify Definitions in SAS Data Cleansing Programs” on page 4](#).

Extracting a character value assigns tokens when the content in the input value meets the criteria in the extraction definition. For example, using the string **"100 Slightly used green Acme MAB-6200 telephone \$100 including touch-tone buttons"** as input results in the following output mapping between tokens and substrings.

```

QUANTITY
  100
BRAND
  "ACME"
MODEL
  "MAB-6200"

```

COLOR
“green”

PRICE
“\$100”

DESCRIPTION
“slightly used telephone including touch-tone buttons”

Extracted character values can also contain empty tokens. For example, in the illustration above, if the input string did not contain a price, then PRICE would contain an “empty” token.

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 7, “DQMATCH Procedure,” on page 35](#)
- [Chapter 8, “DQSCHEME Procedure,” on page 49](#)
- [“DQMATCH Function” on page 94](#)
- [“DQMATCHINFOGET Function” on page 95](#)
- [“DQMATCHPARSED Function” on page 96](#)

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 95](#).

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.

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 [“DQCASE Function” on page 82](#).

Standardization definitions are referenced by the [“DQSTANDARDIZE Function” on page 114](#).

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 Sample Date Standardizations

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

Two-digit year values are standardized as follows:

- If an input year is greater than 00 and less than or equal to 03, the standardized year is 2000, 2001, 2002, or 2003.
- Two-digit input year values that are greater than or equal to 04 and less than or equal to 99 are standardized into the range of 1904–1999.

For example, an input year of 03 is standardized as 2003. An input year of 04 is standardized as 1904. These standardizations are not affected by the value of the SAS system option YEARCUTOFF=.

Gender Analysis, Locale Guess, and Identification Definitions

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

- the gender of an individual based on a name value
- the locale that is the most suitable for a given character value
- the 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 87](#).

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 91](#).

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 90](#).

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: WORD and CHARACTER.

The pattern analysis definition WORD is referenced by the DQPATTERN function. This generates one character of analytical information for each word in the input character value. See [“DQPATTERN Function” on page 105](#) for additional information. The CHARACTER definition generates one character of analytical information for each character in the input character value.

Chapter 4

DMSRVADM Procedure

Overview: DMSRVADM Procedure	21
What Does the DMSRVADM Procedure Do?	21
Syntax: DMSRVADM Procedure	21
PROC DMSRVADM Statement	21
The Job Status Data Set	22
Security	23
Examples: DMSRVADM Procedure	23
Example 1: Generate a Job Status Data Set	23
Example 2: Clean Up Jobs and Logs	24

Overview: DMSRVADM Procedure

What Does the DMSRVADM Procedure Do?

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

Syntax: DMSRVADM Procedure

PROC DMSRVADM

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

PROC DMSRVADM Statement

The DMSRVADM procedure creates a status information data set.

Syntax

PROC DMSRVADM

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

Optional Arguments

HOST= *host-name*

identifies the host of the DataFlux Data Management Server.

Default: *localhost*

OUT= *output-data-set*

specifies the storage location of the job status data set.

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

PASSWORD= *password*

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

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

PORT= *port-number*

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

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

USERID= *userid*

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

The Job Status Data Set

The job status data set contains the following variables:

JOBID

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

- DMSRVCOPYLOG
- DMSRVDELETEDLOG
- DMSRVJOBSTATUS
- DMSRVKILLJOB

See “[Functions Listed Alphabetically](#)” on page 66 for a complete list of functions and call routines.

STATUS

specifies the following job status codes:

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

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

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

Security

If security is implemented on a DataFlux Data Management Server, then you might need to register username and password credentials with the function DMSRVUSER before you run PROC DMSRVADM.

See [“DMSRVUSER Function” on page 80](#).

Examples: DMSRVADM Procedure

Example 1: Generate a Job Status Data Set

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

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

Example 2: Clean Up Jobs and Logs

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

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

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

Chapter 5

DMSRVDATASVC Procedure

Overview: DMSRVDATASVC Procedure	25
What Does the DMSRVDATASVC Procedure Do?	25
Syntax: DMSRVDATASVC Procedure	25
PROC DMSRVDATASVC Statement	26
The Input and Output Data Sets	28
The Input Data Set	28
The Output Data Set	28
Examples: DMSRVDATASVC Procedure	29
Example 1: Send Input Data to Service and Output to Data Set	29
Example 2: Run a DataFlux Data Management Studio Service	29

Overview: DMSRVDATASVC Procedure

What Does the DMSRVDATASVC Procedure Do?

The DMSRVDATASVC procedure runs a DataFlux Data Management Studio real-time service on a DataFlux Data Management Server. DataFlux Data Management real-time services 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 DMSRVDATASVC procedure authenticates you on the server, requests a service, delivers input data to the server, and delivers output data to a SAS DATA set.

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

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

Syntax: DMSRVDATASVC Procedure

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

PROC DMSRVDATASVC

```

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

```

PROC DMSRVDATASVC Statement

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

Syntax**PROC DMSRVDATASVC**

```

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

```

Actions**BLOCKSIZE= rows-per-message**

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

The DataFlux Data Management Studio 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 Data Management Server. If the HOST= option is not specified, localhost is used by default.

Default: *localhost*

PARMLIST= *parameter-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 parmlist with a name that matches a name in the parmlist list, the name is assigned the corresponding parmlist value.

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

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 Data Management Studio 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 Data Management Server. The password can be plain text or encoded in SAS.

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

PORT= *port-number*

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

Default: 21036

SERVICE= *service-name*

identifies the service on the DataFlux Data Management 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 the BLOCKSIZE= option, are ignored.

TIMEOUT= *message-processing-limit*

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

Default: TRIM is not set by default.

USERID=*user-name*

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

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

The Input and Output Data Sets

The Input Data Set

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

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

Features: PROC DMSRVDATASVC
 HOST=
 PORT=
 SERVICE=
 DATA=
 OUT=
 TIMEOUT=
 USERID=
 PASSWORD=

This example reads in a data file to a specified service, and the output from the data service appears in the indicated output data set. The HOST= option names the server, and the PORT= option specifies the port as the default, so the server communicates over port 21036 of 'myserver'. The SERVICE= option is the service to which the input data is sent, and the DATA= option specifies the input data set. The OUT= option specifies the output data set. The TIMEOUT= option is the length of time, in seconds, that the job is allowed to run before being terminated. The USERID= and PASSWORD= options are the user credentials under which the job is being executed. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

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

PROC DMSRVDATASVC
  HOST='myserver'
  PORT=21036
  SERVICE='analysis.ddf'
  DATA=dqsio.dfsample
  OUT=outsrv17
  TIMEOUT=360
  USERID='myname'
  PASSWORD='mypassword';
RUN;
```

Example 2: Run a DataFlux Data Management Studio Service

Features: PROC DMSRVDATASVC
 SERVICE=
 DATA=
 OUT=

This example runs a DataFlux Data Management Studio service on a DataFlux Data Management Server that is installed on the local host. The PORT= option is not set, so the server communicates over the default port 21036. The DATA= option specifies the

input data set. The OUT= option specifies the output data sets. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

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


Chapter 6

DMSRVPROCESSSSVC Procedure

Overview: DMSRVPROCESSSSVC Procedure	31
What Does the DMSRVPROCESSSSVC Procedure Do?	31
Syntax: DMSRVPROCESSSSVC Procedure	31
PROC DMSRVPROCESSSSVC Statement	32
The Input and Output Data Sets	33
The Input Data Set	33
The Output Data Set	34
Example: Run a DataFlux Data Management Service	34

Overview: DMSRVPROCESSSSVC Procedure

What Does the DMSRVPROCESSSSVC Procedure Do?

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

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

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

Syntax: DMSRVPROCESSSSVC Procedure

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

PROC DMSRVPROCESSSSVC

```

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

```

PROC DMSRVPROCESSSSVC Statement

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

Syntax**PROC DMSRVPROCESSSSVC**

```

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

```

Actions**HOST= *host-machine***

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

Default: *localhost*

PARMLIST= *parameter-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 parmlist with a name that matches a name in the parmlist list, the name is assigned the corresponding parmlist value.

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

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 Data Management Server. The password can be plain text or encoded in SAS.

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

PORT= *port-number*

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

Default: 21036

SERVICE= *service-name*

identifies the service on the DataFlux Data Management 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 the BLOCKSIZE= option, are ignored.

TIMEOUT= *message-processing-limit*

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

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

USERID= *user-name*

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

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

The Input and Output Data Sets

The Input Data Set

The DMSRVPROCESSSVC procedure acquires the names of the columns that the service expects to receive as input from the DataFlux Data Management service. Since

services can be created where any named data set is used for input, no input data set name is required for the DMSRVPROCESS SVC procedure.

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.

Example: Run a DataFlux Data Management Service

Features: PROC DMSRVPROCESS SVC
HOST=
PORT=
SERVICE=
TIMEOUT=
USERID=
PASSWORD=

This example runs a DataFlux Data Management service on a DataFlux Data Management Server that is installed on the default port, Port 21036, of the 'myserver' server. In the example, the service is specified and, with the time-out value of 360, the job will terminate after 360 seconds if it does not complete within that time. The job is executed under the credentials (user ID and password) that are specified in the procedure. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

```
PROC DMSRVPROCESS SVC
  HOST='myserver'
  PORT=21036
  SERVICE='concatenate.djf'
  TIMEOUT=360
  USERID='myname'
  PASSWORD='mypassword'
RUN;
```

Chapter 7

DQMATCH Procedure

Overview: DQMATCH Procedure	35
What Does the DQMATCH Procedure Do?	35
Syntax: DQMATCH Procedure	36
PROC DQMATCH Statement	36
CRITERIA Statement	38
Examples: DQMATCH Procedure	40
Example 1: Generate Composite Match Codes	40
Example 2: Matching Values Using Mixed Sensitivity Levels	41
Example 3: Matching Values Using Minimal Sensitivity	43
Example 4: Creating Match Codes for Parsed Values	44
Example 5: Clustering with Multiple CRITERIA Statements	46
Example 6: Generating Multiple Simple Match Codes	47

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, and they 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

Requirement: 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

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

Syntax

```
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>;
```

Optional Arguments

CLUSTER=*variable-name*

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

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

CLUSTER_BLANKS | NO_CLUSTER_BLANKS

specifies how to process blank values.

CLUSTER_BLANKS

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

NO_CLUSTER_BLANKS

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

Default: CLUSTER_BLANKS

CLUSTERS_ONLY

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

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

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

DATA= *data-set-name*

specifies the name of the input SAS data set.

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

DELIMITER | NODELIMITER

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

DELIMITER

when multiple CRITERIA statements are specified, DELIMITER specifies that exclamation points (!) separate the individual *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.

Default: (SAS) uses a delimiter.

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

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

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.

Syntax

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

Optional Arguments

CONDITION=integer

groups CRITERIA statements to constrain the assignment of cluster numbers.

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

Default: 1

Restriction: If you specify a value for the MATCHCODE= option in the DQMATCH procedure, and you specify more than one CONDITION= value, 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: “DQMATCHINFOGET Function” on page 95.

DELIMSTR | VAR

specifies the name of a variable.

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.

Restrictions:

The values of this variable cannot contain delimiter added with the DQPARSE function or the DQPARSETOKENPUT function.

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

See:

[“DQPARSE Function” on page 98](#) for additional information.

[“DQPARSETOKENPUT Function” on page 104](#) for additional information.

EXACT | MATCHDEF

assigns a cluster number.

EXACT

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

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

MATCHDEF= *match-definition*

specifies the *match-definition* that is used to create the *match code* for the specified variable.

Restrictions:

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

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

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*.

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.

Restrictions:

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

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 *matchcodes* 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.

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.

Examples: DQMATCH Procedure

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=clustergrp locale='ENUSA';
    criteria matchdef='Name' var=customer;
    criteria matchdef='Address' var=address;
run;
```

```

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

```

Output 7.1 PROC Print Output

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

Details

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

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

“[Example 2: Matching Values Using Mixed Sensitivity Levels](#)” on [page 41](#) shows how the use of non-default sensitivity levels increases the accuracy of the analysis.

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

Example 2: Matching Values Using Mixed Sensitivity Levels

The following example is similar to “[Example 1: Generate Composite Match Codes](#)” on [page 40](#) 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.

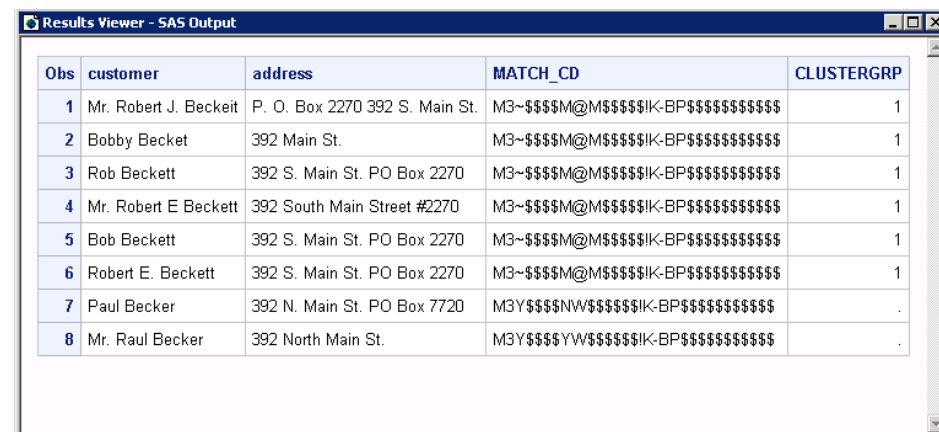
```
/* Create the input data set. */
data cust_db;
  length customer $ 22;
  length address $ 31;
  input customer $char22. address $char31.;

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. Beckeit P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett 392 South Main Street #2270
Mr. Raul Becker       392 North Main St.
;
run;

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

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

Output 7.2 PROC Print Output



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

Details

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

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

This result is different than in “[Example 1: Generate Composite Match Codes](#)” on page 40, where only five values were clustered based on NAME and ADDRESS. This difference is caused by the lower sensitivity setting for the ADDRESS criteria in the current example. This makes the matching less sensitive to variations in the address field. Therefore, the value Bobby Becket has now been included in Cluster 1. 392 Main St. is considered a match with 392 S. Main St. PO Box 2270 and the other variations, this was not true at a sensitivity of 85.

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

Example 3: Matching Values Using Minimal Sensitivity

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

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

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

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db3 matchcode=match_cd
  cluster=clustergrp locale='ENUSA';
  criteria matchdef='Name' var=customer sensitivity=50;
  criteria matchdef='Address' var=address sensitivity=50;
run;

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

Output 7.3 PROC Print Output

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

Details

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

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

Mr. Raul BeckettPaul Becker

A higher sensitivity level would not cluster these observations.

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

Example 4: Creating 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.

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

```

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

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

```

```

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

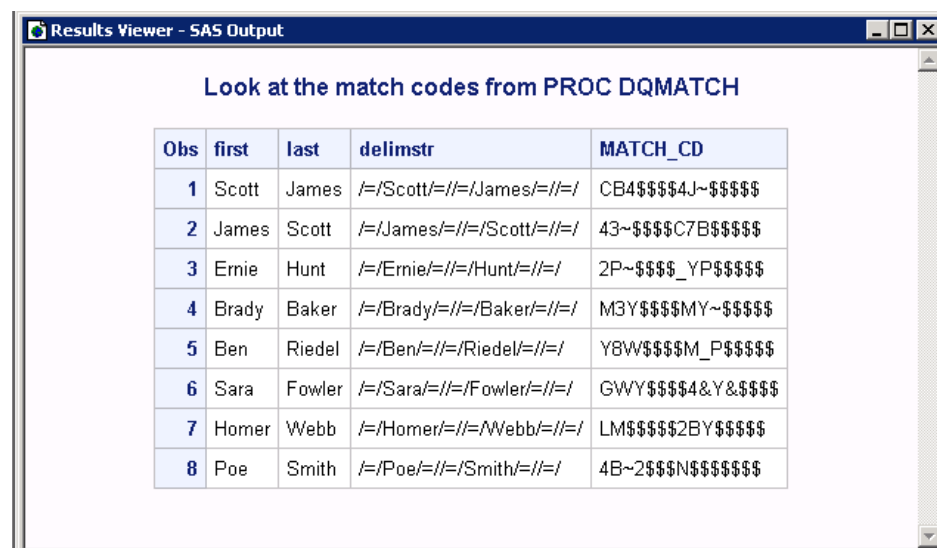
/* Create parsed character values. */
data parsedview;
  set parsed;
  length delimstr $ 100;

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

/* Generate 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;

```

Output 7.4 PROC Print Output


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

Example 5: Clustering with Multiple CRITERIA Statements

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

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

%dqload(dqlocale=(enusa))

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

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

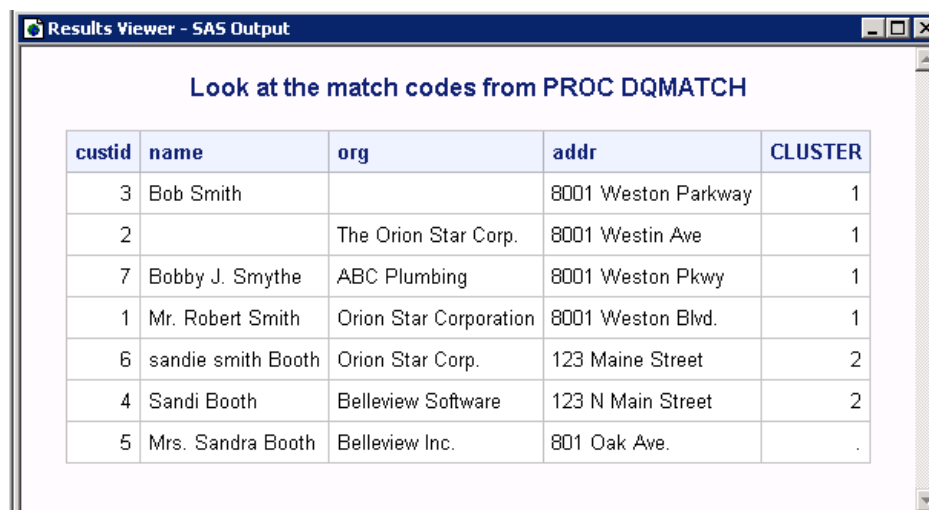
/* Generate the cluster data. Because more than one condition
   is defined, a variable named CLUSTER is created automatically */

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

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

/* Print the result. */

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


Output 7.5 PROC Print Output


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

Details

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

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

Example 6: Generating Multiple Simple 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.

```

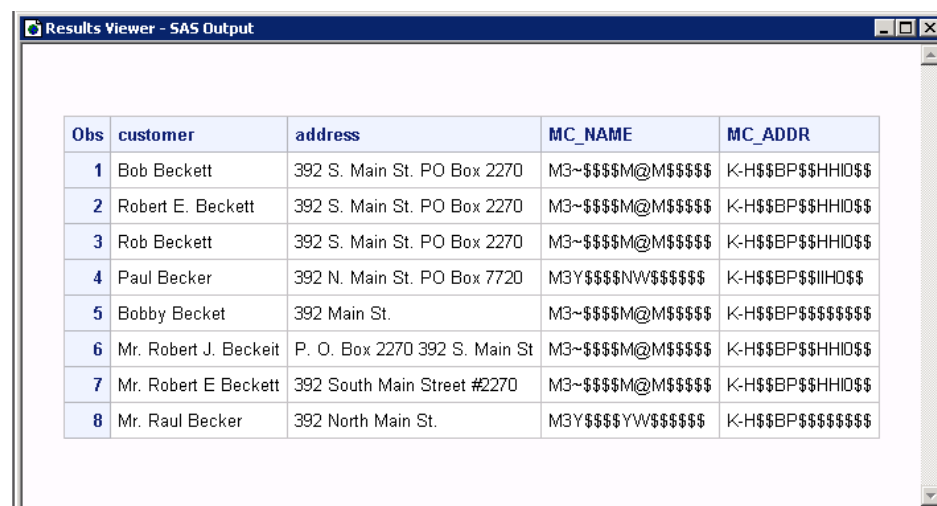
```

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;

```

Output 7.6 PROC Print Output


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

Details

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

Using simple or composite 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: Generate Composite Match Codes” on page 40](#) to compare the examples.

Note: This example is available in the SAS Sample Library under the name DQMCDL2.

Chapter 8

DQSCHEME Procedure

Overview: DQSCHEME Procedure	49
What Does the DQSCHEME Procedure Do?	49
Syntax: DQSCHEME Procedure	50
PROC DQSCHEME Statement	50
APPLY Statement	51
CONVERT Statement	52
CREATE Statement	53
Examples: DQSCHEME Procedure	56
Example 1: Creating an Analysis Data Set	56
Example 2: Creating Schemes	57
Example 3: Creating BFD Schemes	58
Example 4: Applying Schemes	59

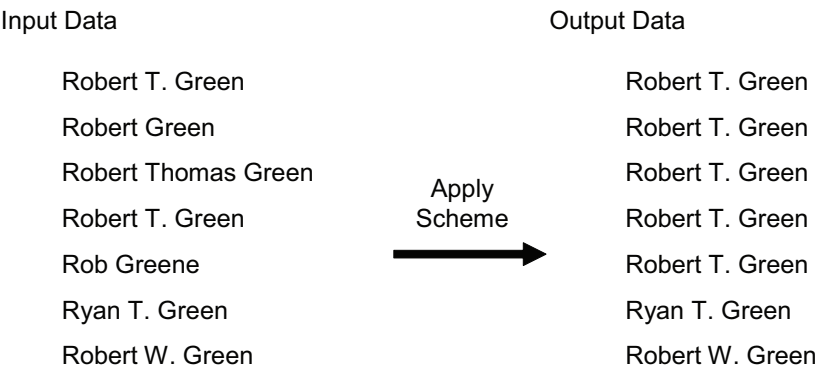
Overview: DQSCHEME Procedure

What Does the DQSCHEME Procedure Do?

PROC DQSCHEME creates scheme data sets and analysis data sets and applies schemes to input data sets. You can also apply schemes with the DQSCHEMEAPPLY function or CALL routine. See [“DQSCHEMEAPPLY CALL Routine” on page 110](#).

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

Figure 8.1 Transform Similar Data Values



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

Syntax: DQSCHEME Procedure

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

PROC DQSCHEME Statement

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

Syntax

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

Optional Arguments

BFD | NOBFD

specifies whether the schemes will be in BFD format or SAS format.

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

Default: BFD

Restrictions:

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

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

See: [“Meta Options” on page 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 schemes are applied.

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

OUT= OUTPUT-DATA-SET

specifies the *output-data-set*.

Interactions:

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

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

If you specify OUT= without any APPLY statements, an empty output data set is created.

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

APPLY Statement

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

See: [“Applying Schemes” on page 8](#) for additional information.

Syntax

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

Optional Arguments

LOCALE=*locale-name*

specifies the name of the match definition, in the specified locales, that is used to create match codes that run 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 that run 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 8](#).

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*

specifies 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.

Requirement: All options are required.

See: [“Applying Schemes” on page 8](#) for additional information.

Syntax

```
CONVERT <BFDTOSAS | SASTOBFD>
      <IN=input-data-set>
      <OUT=output-data-set>;
```

Required 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 DQSCHEME procedure.

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

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

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

OUT=converted-scheme-data-set

specifies the name of the data set with the converted scheme.

Requirements:

If SASTOBFDF 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**.

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

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

CREATE Statement

Creates a scheme or an analysis data set.

Syntax

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

Optional Arguments

ANALYSIS=analysis-data-set

Names the output data set that stores analytical data.

Restriction: This option is required if the SCHEME= option is not specified.

See: “Concepts” on page 3 for additional information.

INCLUDE_ALL

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

- with unique 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 6](#) for additional information.

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 CALL routine.

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

See: [“Meta Options” on page 9](#) for additional information.

MODE= ELEMENT | PHRASE

specifies a mode of scheme application. This information is stored in the scheme as metadata, which specifies a default mode when the scheme is applied. The default mode is superseded by a mode in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine. See [“Applying Schemes” on page 8](#) for additional information.

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

(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.

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 on page 50](#) for additional information.

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

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

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

IGNORE_CASE

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

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

USE_MATCHDEF

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

Interactions:

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

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

The value of the SCHEME_LOOKUP= option is stored in the scheme as a meta option. This specifies a default lookup method when the scheme is applied. The default supplied by this meta option 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 for additional information.

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. The value of the SENSITIVITY= option is stored in the scheme as a meta option. This provides a default sensitivity value when the scheme is applied.

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

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

Default: 85

Interactions:

The default value supplied by this meta option is superseded by a sensitivity value specified in the APPLY statement, or in the DQSCHEMEAPPLY CALL routine.

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

See: “Meta Options” on page 9 for additional information.

VAR=input-character-variable

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

Examples: DQSCHEME Procedure

Example 1: Creating an Analysis Data Set

Overview

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

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

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

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

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

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

/* Create the analysis data set. */
```

```

proc dqscheme data=vendors;
  create analysis=a_state
    matchdef='State (Scheme Build)'
    var=state
    locale='ENUSA';
run;

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

```

Output 8.1 PROC Print Output



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

Example 2: Creating Schemes

The following example generates three schemes in SAS format. 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;

```

Details

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

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

Example 3: Creating 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 DataFlux Data Management Studio 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.

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

```

/* 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

```

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';

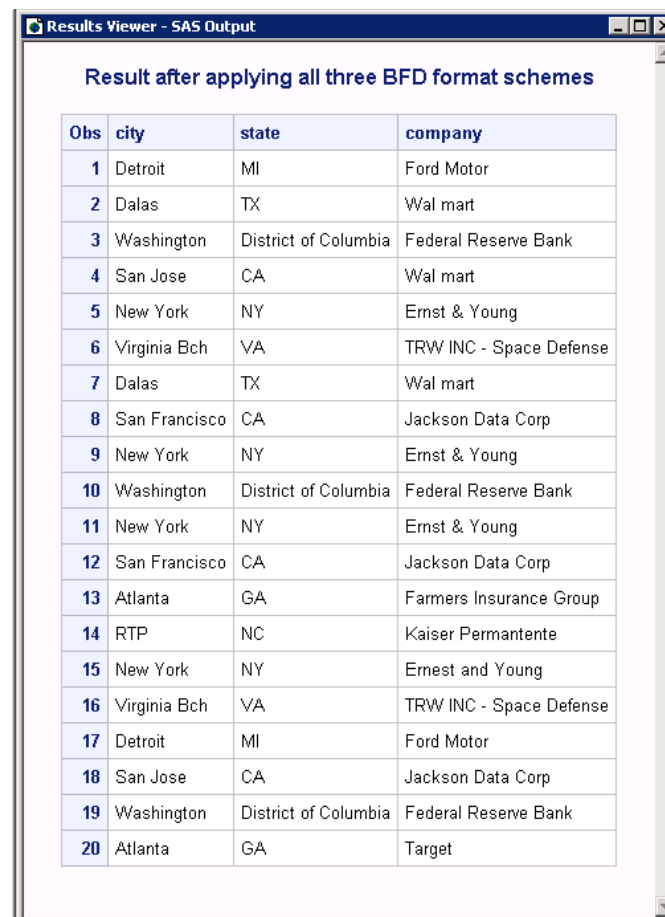
```

```

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;

```

Output 8.2 PROC Print Output


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

Details

Note that the 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. When you use the default scheme apply mode MODE=PHRASE, 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.

Chapter 9

AUTOCALL Macros

Dictionary	61
%DQLOAD AUTOCALL Macro	61
%DQPUTLOC AUTOCALL Macro	62
%DQUNLOAD AUTOCALL Macro	64
Macro Resources	64

Dictionary

%DQLOAD AUTOCALL Macro

Sets system option values and loads locales into memory.

Syntax

%DQLOAD *option(s)*

Summary of Optional Arguments

status information

DQINFO= 0 | 1

Required Arguments

DQSETUPLOC = "**path-specification**"

specifies the location (root directory) of the Quality Knowledge Base. The Quality Knowledge Base contains the specified locales.

"path-specification" contains a series of selections that the system follows to reach a specified folder.

Windows Specifics

the path specification identifies the root directory of the Quality Knowledge Base.

UNIX Specifics

the path specification identifies the root directory of the Quality Knowledge Base.

DQLOCALE = (LOCALE1,...,LOCALEN)

specifies an ordered list of locales to load into memory.

Optional Argument**DQINFO= 0 | 1**

specifies that when DQINFO=1 the additional information that is generated about the status of the locale load operation is written to the SAS log.

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 default location of the sample Quality Knowledge Base.

Example

The following example uses the DQLOCALE option to specify an ordered list of locales to load into memory. DQSETUPLOC specifies the location of the Quality Knowledge Base.

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

%DQPUTLOC AUTOCALL Macro

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

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

Syntax

%DQPUTLOC *option(s)*

Summary of Optional Arguments

lists related parse definition

PARSEDEFN= 0 | 1

shortens length of log

SHORT= 0 | 1

specifies the locale of interest

LOCALE

Optional Arguments

LOCALE

specifies the locale of interest. The value can be a locale name in quotation marks or an expression that evaluates to a locale name. The specified locale must have been loaded into memory as part of the locale list.

Default: Locale is the first locale in the locale list.

Requirement: Locale must be loaded into memory before this macro is called.

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

See: “DQSETUPLOC” on page 121 for additional information.

PARSEDEFN= 0 | 1

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

Specify PARSEDEFN = 0 if you do not want to display the related parse definitions.

Specify PARSEDEFN = 1 to display the related parse definition.

Default: 1

SHORT= 0 | 1

Shortens the length of the entry in the SAS log.

Specify SHORT =0 to display the descriptions of how the definitions are used.

Specify SHORT =1 to remove the descriptions of how the definitions are used.

Default: 0

Details

The %DQPUTLOC AUTOCALL macro displays the contents of the specified locale in the SAS log. Locale contents include all definitions, parse tokens, related functions, and the names of the parse definitions that are related to each match definition. Knowing the related parse definitions enables the creation of parsed character values. See “DQPARSETOKENPUT Function” on page 104 for additional information.

It also enables the creation of match codes for parsed character values. See “DQMATCHPARSED Function” on page 96 for additional information.

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

Example

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

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

See Also

References

- [“DQLOCALEINFOGET Function” on page 92](#)
- [“DQLOCALEINFOLIST Function” on page 92](#)

%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. After unloading memory, be sure to load locales again with the %DQLOAD AUTOCALL macro before running any data cleansing programs.

Macro Resources

Macro Resources

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

Chapter 10

Functions and CALL Routines

Overview	66
Functions Listed Alphabetically	66
Functions Listed by Category	68
DataFlux Data Management Server Functions	68
Case Functions	68
Gender Analysis, Locale Guessing, and Identification Functions	69
Matching Functions	69
Parsing Functions	69
Extraction Functions	70
Pattern Analysis Functions	70
Reporting Functions	70
Scheme Functions and CALL Routines	71
Standardization Functions	71
Dictionary	72
DMSRVBATCHJOB Function	72
DMSRVCOPYLOG Function	73
DMSRVDELETELOG Function	74
DMSRVJOBSTATUS Function	76
DMSRVKILLJOB Function	77
DMSRVPROFILEJOB Function	79
DMSRVUSER Function	80
DMSRVVER Function	81
DQCASE Function	82
DQEXTINFOGET Function	83
DQEXTRACT Function	84
DQEXTTOKENGET Function	85
DQEXTTOKENPUT Function	86
DQGENDER Function	87
DQGENDERINFOGET Function	88
DQGENDERPARSED Function	89
DQIDENTIFY Function	90
DQLOCALEGUESS Function	91
DQLOCALEINFOGET Function	92
DQLOCALEINFOLIST Function	92
DQMATCH Function	94
DQMATCHINFOGET Function	95
DQMATCHPARSED Function	96
DQOPTSURFACE Function	97
DQPARSE Function	98
DQPARSE CALL Routine	99

DQPARSEINFOGET Function	100
DQPARSEINPUTLEN Function	101
DQPARSERESLIMIT Function	102
DQPARSESCORDEPTH Function	103
DQPARSETOKENGET Function	103
DQPARSETOKENPUT Function	104
DQPATTERN Function	105
DQSCHEMEAPPLY Function	107
DQSCHEMEAPPLY CALL Routine	110
DQSTANDARDIZE Function	114
DQTOKEN Function	115
DQVERBF Function	116
DQVERQKB Function	117

Overview

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

Functions Listed Alphabetically

- The “[DMSRVBATCHJOB Function](#)” on [page 72](#) runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.
- The “[DMSRVCOPYLOG Function](#)” on [page 73](#) copies a job's log file from a DataFlux Data Management Server.
- The “[DMSRVDELETELOG Function](#)” on [page 74](#) deletes a job's log file from a DataFlux Data Management Server.
- The “[DMSRVJOBSTATUS Function](#)” on [page 77](#) returns the status of a job that was submitted to a DataFlux Data Management Server.
- The “[DMSRVKILLJOB Function](#)” on [page 77](#) terminates a job that is running on a DataFlux Data Management Server.
- The “[DMSRVPROFILEJOB Function](#)” on [page 79](#) generates a profile from a Data Management Server repository.
- The “[DMSRVUSER Function](#)” on [page 80](#) registers credentials (username and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.
- The “[DMSRVVER Function](#)” on [page 81](#) returns the version of the DataFlux Data Management Server.

- The “[DQCASE Function](#)” on page 82 returns a character value with standardized capitalization.
- The “[DQCASE Function](#)” on page 82 returns the token names in an extraction definition.
- The “[DQCASE Function](#)” on page 82 returns an extracted character value.
- The “[DQCASE Function](#)” on page 82 returns a token from an extraction character value.
- The “[DQCASE Function](#)” on page 82 inserts a token into an extraction character value and returns the updated extraction character value.
- The “[DQGENDER Function](#)” on page 87 returns a gender determination from the name of an individual.
- The “[DQGENDERINFOGET Function](#)” on page 88 returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “[DQGENDERPARSED Function](#)” on page 89 returns a gender determination from the parsed name of an individual.
- The “[DQIDENTIFY Function](#)” on page 90 returns a category name from a character value.
- The “[DQLOCALEGUESS Function](#)” on page 91 returns the name of the locale that is most likely represented by a character value.
- The “[DQLOCALEINFOGET Function](#)” on page 92 returns information about locales.
- The “[DQMATCHPARSED Function](#)” on page 96 returns the names of the definitions in a locale and returns a count of those definitions.
- The “[DQMATCH Function](#)” on page 94 returns a match code from a character value.
- The “[DQMATCHINFOGET Function](#)” on page 95 returns the name of the parse definition that is associated with a match definition.
- The “[DQMATCHPARSED Function](#)” on page 96 returns a match code from a parsed character value.
- The “[DQOPTSURFACE Function](#)” on page 97 reveals or hides non-surfaced definitions.
- The “[DQPARSE CALL Routine](#)” on page 99 returns a parsed character value and a status flag.
- The “[DQPARSE Function](#)” on page 98 returns a parsed character value.
- The “[DQPARSEINFOGET Function](#)” on page 100 returns the token names for the specified parse definition.
- The “[DQPARSEINPUTLEN Function](#)” on page 101 sets the default length of parsed input, and returns a string indicating its previous value.
- The “[DQPARSERESLIMIT Function](#)” on page 102 sets a limit on resources consumed during parsing.
- The “[DQPARSESCORDEPTH Function](#)” on page 103 specifies how deeply to search for the best parsing score.
- The “[DQPARSETOKENGET Function](#)” on page 103 returns a token from a parsed character value.

- The “[DQPARSETOKENPUT Function](#)” on page 104 inserts a token into a parsed character value and returns the updated parsed character value.
- The “[DQPATTERN Function](#)” on page 105 returns a pattern analysis from an input character value.
- The “[DQSCHEMEAPPLY CALL Routine](#)” on page 110 applies a scheme and returns a transformed value and a transformation flag.
- The “[DQSCHEMEAPPLY Function](#)” on page 107 applies a scheme and returns a transformed value after applying a scheme.
- The “[DQSTANDARDIZE Function](#)” on page 114 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 115 returns a token from a character value.
- The “[DQVERBF Function](#)” on page 116 returns the version of Blue Fusion.
- The “[DQVERQKB Function](#)” on page 117 returns the version of the currently loaded Quality Knowledge Base.

Functions Listed by Category

DataFlux Data Management Server Functions

- The “[DMSRVBATCHJOB Function](#)” on page 72 runs a DataFlux Data Management Studio process or data job on a DataFlux Data Management Server and returns a job identifier.
- The “[DMSRVCOPYLOG Function](#)” on page 73 copies a job's log file from a DataFlux Data Management Server.
- The “[DMSRVDELETELOG Function](#)” on page 74 deletes a job's log file from a DataFlux Data Management Server.
- The “[DMSRVJOBSTATUS Function](#)” on page 76 returns the status of a job that was submitted to a DataFlux Data Management Server.
- The “[DMSRVKILLJOB Function](#)” on page 77 terminates a job that is running on a DataFlux Data Management Server.
- The “[DMSRVPROFILEJOB Function](#)” on page 79 generates a profile from a repository on a DataFlux Data Management Server.
- The “[DMSRVUSER Function](#)” on page 80 registers credentials (username and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.
- The “[DMSRVVER Function](#)” on page 81 returns the version of the DataFlux Data Management Server.

Case Functions

- The “[DQCASE Function](#)” on page 82 returns a character value with standardized capitalization.

- The “[DQSTANDARDIZE Function](#)” on [page 114](#) 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 87](#) returns a gender determination from the name of an individual.
- The “[DQGENDERINFOGET Function](#)” on [page 88](#) returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “[DQGENDERPARSED Function](#)” on [page 89](#) returns a gender determination from the parsed name of an individual.
- The “[DQIDENTIFY Function](#)” on [page 90](#) returns a category name from a character value.
- The “[DQLOCALEGUESS Function](#)” on [page 91](#) returns the name of the locale that is most likely represented by a character value.
- The “[DQLOCALEINFOGET Function](#)” on [page 92](#) returns information about locales.
- The “[DQMATCHPARSED Function](#)” on [page 96](#) returns the names of the definitions in a locale and returns a count of those definitions.

Matching Functions

- The “[DQMATCH Function](#)” on [page 94](#) returns a match code from a character value.
- The “[DQMATCHINFOGET Function](#)” on [page 95](#) returns the name of the parse definition that is associated with a match definition.
- The “[DQMATCHPARSED Function](#)” on [page 96](#) returns a match code from a parsed character variable.

Parsing Functions

- The “[DQGENDERINFOGET Function](#)” on [page 88](#) returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “[DQGENDERPARSED Function](#)” on [page 89](#) returns a gender determination from the parsed name of an individual.
- The “[DQMATCHPARSED Function](#)” on [page 96](#) returns a match code from a parsed character value.
- The “[DQPARSE CALL Routine](#)” on [page 99](#) returns a parsed character value and a status flag.
- The “[DQPARSE Function](#)” on [page 98](#) returns a parsed character value.
- The “[DQPARSEINFOGET Function](#)” on [page 100](#) returns the token names for the specified parse definition.

- The “[DQPARSEINPUTLEN Function](#)” on page 101 sets the default length of parsed input. DQPARSEINPUTLEN also returns a string indicating its previous value.
- The “[DQPARSERESLIMIT Function](#)” on page 102 sets a limit on resources consumed during parsing.
- The “[DQPARSESCORDEPTH Function](#)” on page 103 specifies how deeply to search for the best parsing score.
- The “[DQPARSETOKENGET Function](#)” on page 103 returns a token from a parsed character value.
- The “[DQPARSETOKENPUT Function](#)” on page 104 inserts a token into a parsed character value and returns the updated parsed character value.

Extraction Functions

- The “[DQEXTINFOGET Function](#)” on page 83 returns the token names in an extraction definition.
- The “[DQEXTRACT Function](#)” on page 84 returns an extracted character value.
- The “[DQEXTTOKENGET Function](#)” on page 85 returns a token from an extraction character value.
- The “[DQEXTTOKENPUT Function](#)” on page 86 inserts a token into an extraction character value and returns the updated extraction character value.

Pattern Analysis Functions

The “[DQPATTERN Function](#)” on page 105 returns a pattern analysis from an input character value.

Reporting Functions

- The “[DQGENDER Function](#)” on page 87 returns a gender determination from the name of an individual.
- The “[DQGENDERPARSED Function](#)” on page 89 returns a gender determination from the parsed name of an individual.
- The “[DQGENDERINFOGET Function](#)” on page 88 returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “[DQIDENTIFY Function](#)” on page 90 returns a category name from a character value.
- The “[DQLOCALEGUESS Function](#)” on page 91 returns the name of the locale that is most likely represented by a character value.
- The “[DQLOCALEINFOGET Function](#)” on page 92 returns information about locales.
- The “[DQMATCHPARSED Function](#)” on page 96 returns the names of the definitions in a locale and returns a count of those definitions.
- The “[DQMATCH Function](#)” on page 94 returns a match code from a character value.

- The “[DQMATCHINFOGET Function](#)” on page 95 returns the name of the parse definition that is associated with a match definition.
- The “[DQMATCHPARSED Function](#)” on page 96 returns a match code from a parsed character value.
- The “[DQPARSE CALL Routine](#)” on page 99 returns a parsed character value and a status flag.
- The “[DQPARSE Function](#)” on page 98 returns a parsed character value.
- The “[DQPARSEINFOGET Function](#)” on page 100 returns the token names for the specified parse definition.
- The “[DQPARSETOKENGET Function](#)” on page 103 returns a token from a parsed character value.
- The “[DQPARSETOKENPUT Function](#)” on page 104 inserts a token into a parsed character value and returns the updated parsed character value.
- The “[DQPATTERN Function](#)” on page 105 returns a pattern analysis from an input character value.
- The “[DQSCHEMEAPPLY CALL Routine](#)” on page 110 applies a scheme and returns a transformed value and a transformation flag.
- The “[DQSCHEMEAPPLY Function](#)” on page 107 applies a scheme and returns a transformed value after applying a scheme.
- The “[DQSTANDARDIZE Function](#)” on page 114 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 115 returns a token from a character value.
- The “[DQVERBF Function](#)” on page 116 returns the version of Blue Fusion.
- The “[DQVERQKB Function](#)” on page 117 returns the version of the currently loaded Quality Knowledge Base.

Scheme Functions and CALL Routines

- The “[DQSCHEMEAPPLY Function](#)” on page 107 applies a scheme and returns a transformed value.
- The “[DQSCHEMEAPPLY CALL Routine](#)” on page 110 applies a scheme and returns a transformed value and a transformation flag.

Standardization Functions

- The “[DQCASE Function](#)” on page 82 returns a character value with standardized capitalization.
- The “[DQSTANDARDIZE Function](#)” on page 114 returns a character value after standardizing the casing, spacing, and format, and after applying a common representation to certain words and abbreviations.

Dictionary

DMSRVBATCHJOB Function

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

Valid in: DATA step, PROC SQL, and SCL

Requirements: If specified, the locale must be loaded into memory as part of the locale list. The character variable that receives the return value must have a minimum length of 52.

Syntax

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

Required Arguments

job-name

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

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

/* Correct */
dmsrvBatchJob(jobname, dmsrvhostname, 21036);
```

port

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

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

Example:

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

/* Port 21036 or 21037 is used */
dmsrvBatchJob('jobname', dmsrvhostname, 0);

/* Correct */
dmsrvBatchJob('jobname', dmsrvhostname, 21036);
```

Optional Argument

parameter-list

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

Details

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

- You can specify any number of macro value pairs.

Example: DMSRVBATCHJOB Function

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

```
data _null_;
  jobid = dmsrvBatchJob ('myjob.djf', 'myserver', 21036);
run;
```

See Also

Functions

- [“DMSRVCOPYLOG Function” on page 73](#)
- [“DMSRVDELETEDLOG Function” on page 74](#)
- [“DMSRVJOBSTATUS Function” on page 76](#)

DMSRVCOPYLOG Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required Arguments

job-ID

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

```

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

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

/* Correct */
dmsrvCopyLog('jobid', dmsrvhostname, 21036, 'filename');

```

port

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

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

Example:

```

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

/* Port 21036 or 21037 is used */
dmsrvCopyLog('jobid', dmsrvhostname, 0, 'filename');

/* Correct */
dmsrvCopyLog('jobid', dmsrvhostname, 21036, 'filename');

```

filename

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

Details

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

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

Example: DMSRVCOPYLOG Function

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

```
copyrc= dmsrvCopyLog(jobid,'dmServer1', 5001,'dmServer1.log');
```

See Also**Functions**

- [“DMSRVBATCHJOB Function” on page 72](#)
- [“DMSRVDELETEDLOG Function” on page 74](#)
- [“DMSRVJOBSTATUS Function” on page 76](#)

DMSRVDELETEDLOG Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required Arguments

job-ID

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

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

/* Correct */
dmsrvDeleteLog('jobid', dmsrvhostname, 21036);
```

port

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

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

Example:

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

/* Port 21036 or 21037 is used */
dmsrvDeleteLog('jobid', dmsrvhostname, 0);

/* Correct */
dmsrvDeleteLog('jobid', dmsrvhostname, 21036);
```

Details

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

- DMSRVDELETEDLOG 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: DMSRVDELETEDLOG FUNCTION

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

```
delrc= dmsrvDeleteLog(jobid,'dmServer1', 5001);
```

See Also

Functions

- “DMSRVBATCHJOB Function” on page 72
- “DMSRVCOPYLOG Function” on page 73
- “DMSRVJOBSTATUS Function” on page 76

DMSRVJOBSTATUS Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required Arguments

job-ID

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

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

/* Correct */
dmsrvJobStatus('jobid', dmsrvhostname, 21036, 20, 5);
```

port

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

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

Example:

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

/* Port 21036 or 21037 is used */
```

```
dmsrvJobStatus('jobid', dmsrvhostname, 0, 20, 5);

/* Correct */
dmsrvJobStatus('jobid', dmsrvhostname, 21036, 20, 5);
```

time-out

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

- | | |
|--------------------------|---|
| -1 | returns status information about when the job is finished. Return values are 0 (job completed successfully) or 1 (job failed). This value invalidates the <i>interval</i> argument. |
| 0 | returns status information immediately. Return values are 0 (job completed successfully), 1 (job failed), or 2 (job running). This value invalidates the <i>interval</i> argument. |
| <i>greater-than-zero</i> | specifies a time limit for the <i>interval</i> argument. If the job is still running after the <i>time-out</i> value, another value is returned only when the job is finished. |

interval

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

Details

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

Return values are 0 (job completed successfully), 1 (job failed), or 2 (job running).

Example: DMSRVJOBSTATUS Function

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

```
status= dmsrvJobStatus(jobid,'dmServer1', 5001, 20, 5);
```

See Also**Functions**

- [“DMSRVBATCHJOB Function” on page 72](#)
- [“DMSRVDELETELOG Function” on page 74](#)
- [“DMSRVKILLJOB Function” on page 77](#)

DMSRVKILLJOB Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required Arguments

job-ID

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

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

/* Correct */
dmsrvKillJob('jobid', dmsrvhostname, 21036);
```

port

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

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

Example:

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

/* Port 21036 or 21037 is used */
dmsrvKillJob('jobid', dmsrvhostname, 0);

/* Correct */
dmsrvKillJob('jobid', dmsrvhostname, 21036);
```

Details

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

Example: DMSRVKILLJOB Function

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

```
killrc= dmsrvKillJob(jobid,'dmServer1',5001);
```


See Also

Functions

- [“DMSRVBATCHJOB Function” on page 72](#)
- [“DMSRVJOBSTATUS Function” on page 76](#)

DMSRVPROFILEJOB Function

Generates a profile from a Data Management server repository.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Arguments

job-name

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

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

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

/* Correct */
dmsrvProfileJob(jobname, dmsrvhostname, 21036, 0);
```

port

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

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

Example:

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

/* Port 21036 or 21037 is used */
dmsrvProfileJob('jobname', dmsrvhostname, 0, 1);

/* Correct */
```

```
dmsrvProfileJob('jobname', dmsrvhostname, 21036, 1);
```

append-flag

appends or overwrites job results.

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

Optional Argument***description-character***

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

Details

The DMSRVPROFILEJOB function generates a profile from a Data Management server repository.

Example: DMSRVPROFILEJOB Function

The following example generates a profile from the specified repository.

```
data _null_;
  jobid = dmsrvProfileJob('myfolder/prof_job', 'myserver', 21036, 1);
run;
```

See Also**Functions**

- [“DMSRVJOBSTATUS Function” on page 77](#)
- [“DMSRVKILLJOB Function” on page 78](#)

DMSRVUSER Function

Registers a user on a DataFlux Data Management Server.

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required Arguments***user-ID***

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

password

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

Details

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

- Call this function as needed in a single DATA step to access different Data Management Servers or to change the registered user credentials within a single Data Management Server.
- If security has not been configured on a DataFlux Data Management Server, then the DMSRVUSER function has no effect.
- Return values are 0 (successful registration of credentials) or 1 (failed to register credentials).

Example: DMSRVUSER Function

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

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

DMSRVVER Function

Returns the version of the DataFlux Data Management Server.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DMSRVVER (*host* , *port*)

Required Arguments

host

identifies the host of the DataFlux Data Management Server.

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

Example:

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

port

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

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

Example:

```
/* Port 21036 or 21037 is used */
dmsrvVer('dmsrvhostname', 0);
```

Details

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

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

Example: DMSRVVER Function

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

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

See Also

[“DMSRVBATCHJOB Function” on page 72](#)

DQCASE Function

Returns a character value with standardized capitalization.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Arguments

char

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

case-definition

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.

Optional Argument

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: DQCASE Function

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

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

After this function call, the value of ORGNAME is Bill's Plumbing & Heating.

DQEXTINFOGET Function

Returns the token names in an extraction definition.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Argument

extraction-definition

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

Optional Argument

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

Example: DQEXTINFOGET Function

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

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

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

See Also

Functions

- “DQEXTTOKENGET Function” on page 85
- “DQEXTTOKENPUT Function” on page 86
- “DQEXTRACT Function” on page 84

DQEXTRACT Function

Returns an extracted character value.

Valid in:	DATA step, PROC SQL, and SCL
Restriction:	Always use the DQEXTTOKENGET function to retrieve tokens from extracted 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.

Syntax

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

Required Arguments

extraction-string

the value that is extracted according to the specified extraction 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.

extraction-definition

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

Optional Argument

locale

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

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

Details

The DQEXTRACT function returns an extracted character value. The return value contains delimiters that identify the elements in the value that correspond to the tokens that are enabled by the extraction definition. The delimiters in the value allow functions such as DQEXTTOKENGET to access the elements in the value based on specified token names.

Example

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

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

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

See Also

- [“DQEXTINFOGET Function” on page 83](#)
- [“DQTOKEN Function” on page 115](#)

DQEXTTOKENGET Function

Returns a token from an extraction character value.

Valid in: DATA step, PROC SQL, and SCL

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

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

Syntax

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

Required Arguments

extraction-char

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

To determine how the extraction definition inserts delimiters, use the DQEXTINFOGET function.

token

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

extraction-definition

the name of the extraction definition. The definition must exist in the locale that is used. The extraction definition must be the same as the extraction definition that originally extracted the EXTRACTION-CHAR value.

Optional Argument

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

Example

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

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

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

See Also

Functions

- [“DQEXTRACT Function” on page 84](#)
- [“DQEXTINFOGET Function” on page 83](#)
- [“DQTOKEN Function” on page 115](#)

DQEXTTOKENPUT Function

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

Valid in: DATA step and SCL

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

Syntax

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

Required Arguments

extraction-char

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

token-value

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

token-name

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

extraction-definition

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

Optional Argument

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

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

See Also

Functions

- [“DQGENDERINFOGET Function” on page 88](#)
- [“DQGENDERPARSED Function” on page 89](#)
- [“DQMATCHPARSED Function” on page 96](#)
- [“DQEXTTOKENGET Function” on page 85](#)

DQGENDER Function

Returns a gender determination from the name of an individual.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required 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, which must exist in the specified locale. The value must be the name of a character variable, in quotation marks. An expression that evaluates to a variable name, or a quoted value is also valid.

Optional Argument

locale

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

Default: 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: DQGENDER Function

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 89](#)

DQGENDERINFOGET Function

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

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Argument

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.

Optional Argument

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: DQGENDERINFOGET Function

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

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

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

See Also

Functions

- [“DQGENDER Function” on page 87](#)
- [“DQGENDERPARSED Function” on page 89](#)
- [“DQPARSE Function” on page 98](#)
- [“DQPARSETOKENPUT Function” on page 104](#)

DQGENDERPARSED Function

Returns the gender of an individual.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Arguments

parsed-char

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.

Optional Argument***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 87](#)
- [“DQGENDERINFOGET Function” on page 88](#)

DQIDENTIFY Function

Returns a category name from a character value.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required 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

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

Optional Argument***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: DQIDENTIFY Function

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

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

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

DQLOCALEGUESS Function

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

Valid in: DATA step, PROC SQL, and SCL

Syntax

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

Required 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: DQLOCALEGUESS Function

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 92](#)
- [“Load and Unload Locales” on page 6](#)

DQLOCALEINFOGET Function

Returns information about locales.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQLOCALEINFOGET (<*info-type*>)

Optional Argument

info-type

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

Details

The DQLOCALEINFOGET function returns a comma-delimited list of 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: DQLOCALEINFOGET Function

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

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

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

See Also

Function and autocall macro

- [“DQLOCALEINFOLIST Function” on page 92](#)
- [“%DQPUTLOC AUTOCALL Macro” on page 62](#)

DQLOCALEINFOLIST Function

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

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Arguments

DEFINITION-TYPE

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

Definition types are as follows:

ALL

CASE

EXTRACTION

GENDER

GUESS

IDENTIFICATION

MATCH

PARSE

PATTERN

STANDARDIZATION

LOCALE

specifies a 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.

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

See:

“%DQPUTLOC AUTOCALL Macro” on page 62 for additional information.

“DQLOCALEINFOGET Function” on page 92 for additional information.

Example: DQLOCALEINFOLIST Function

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

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

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

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

DQMATCH Function

Returns a match code from a character value.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required 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.

Optional Arguments

sensitivity

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

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

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


See Also

Functions

- [Chapter 7, “DQMATCH Procedure,” on page 35](#)

DQMATCHINFOGET Function

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

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Argument

match-definition

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

Optional Argument

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: DQMATCHINFOGET Function

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

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

```

        parsedValue=dqParseTokenPut(parsedValue, 'Joel', 'Given Name', 'Name');
        parsedValue=dqParseTokenPut(parsedValue, 'Alston', 'Family Name', 'Name');
        matchCode=dqMatchParsed(parsedValue, 'Name');
        put matchCode=;
    run;

```

DQMATCHPARSED Function

Returns a match code from a parsed character value.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Arguments

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.

match-definition

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

Optional Arguments

sensitivity

specifies an integer value that determines the amount of information in the returned match code. Valid values range from 50 to 95. The default value is 85. A higher sensitivity value inserts more information in the match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster. 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: DQMATCHPARSED Function

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

```

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

```

```
matchCode=dqMatchParsed(parsedName, 'name', 90, 'enusa');
run;
```

See Also

Functions

- [Chapter 7, “DQMATCH Procedure,” on page 35](#)
- [“How to Create a Match Code” on page 11](#)
- [“DQMATCHINFOGET Function” on page 95](#)
- [“DQPARSEINFOGET Function” on page 100](#)
- [“DQTOKEN Function” on page 115](#)

DQOPTSURFACE Function

Reveals or hides non-surfaced definitions.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQOPTSURFACE (*'surface-definition'*)

Required Argument

surface-definition

specifies the policy for the surface definitions.

Details

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

YES

reveals the non-surfaced definitions.

NO

hides the non-surfaced definitions.

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

Example: DQOPTSURFACE Function

The following example specifies that non-surfaced definitions are revealed. The character value `oldDEFAULT` contains the value of the previous setting.

```
oldDefault=DQOPTSURFACE('YES');
```

DQPARSE Function

Returns a parsed character value.

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

Syntax

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

Required Arguments

parse-string

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.

parse-definition

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

Optional Argument

locale

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

Default: 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: DQPARSE Function

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

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

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

See Also

- [“DQPARSEINFOGET Function” on page 100](#)
- [“DQTOKEN Function” on page 115](#)

DQPARSE CALL Routine

Returns a parsed character value and a status flag.

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

Syntax

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

Required Arguments

parse-string

the input value that is parsed according to the specified parse definition. The value 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.

parse-definition

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

parse-result

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

parse-return-code

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

Optional Argument

locale

specifies a character constant, variable, or expression that 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: DQPARSE CALL Routine

The following example parses the name of an individual.

```
data a;
  length parsename $ 40;
  call dqparse (name, 'Name', parsename, solution);
  if solution= 1 then
    put 'found solution';
  else
    put 'no solution';
run;
```

See Also

Functions

- [“DQPARSEINFOGET Function” on page 100](#)
- [“DQTOKEN Function” on page 115](#)

DQPARSEINFOGET Function

Returns the token names in a parse definition.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required Argument

parse-definition

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

Optional Argument

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: DQPARSEINFOGET Function

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, which are the names of the three tokens in this parse definition.

See Also

Functions

- [“DQPARSETOKENGET Function” on page 103](#)
- [“DQPARSETOKENPUT Function” on page 104](#)
- [“DQTOKEN Function” on page 115](#)

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'*)

Required Argument

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 that indicates 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: DQPARSEINPUTLEN Function

The following example sets the default input length to SHORT. The previous value of the parse input length is returned as the value of `oldDEFAULT`.

```
oldDefault= dqParseInfPutLen('short');
```

DQPARSERESLIMIT Function

Sets a limit on resources consumed during parsing.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQPARSERESLIMIT (*resource-limit*)

Required Argument

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 that indicates 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: DQPARSERESLIMIT Function

The following example sets the default resource limit to INTENSIVE. The value of `oldDEFAULT` is the previous value of the resource limit.

```
oldDefault=DQPARSERESLIMIT('intensive');
```

DQPARSESCORDEPTH Function

Specifies how deeply to search for the best parsing score.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQPARSESCORDEPTH (*level*)

Required Argument

level

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: DQPARSESCORDEPTH Function

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.

Valid in: DATA step, PROC SQL, and SCL

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

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

Syntax

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

Required 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.

token

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

parse-definition

the name of the parse definition. The definition must 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.

Optional Argument***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: DQPARSETOKENGET Function

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

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

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

See Also**Functions**

- [“DQPARSE Function” on page 98](#)
- [“DQPARSEINFOGET Function” on page 100](#)
- [“DQTOKEN Function” on page 115](#)

DQPARSETOKENPUT Function

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

Valid in: DATA step and SCL

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

Syntax

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

Required Arguments

parsed-char

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

token-value

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

token-name

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

parse-definition

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.

Optional Argument

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 88](#)
- [“DQGENDERPARSED Function” on page 89](#)
- [“DQMATCHPARSED Function” on page 96](#)
- [“DQPARSETOKENGET Function” on page 103](#)

DQPATTERN Function

Returns a pattern analysis from an input character value.

Valid in: DATA step and SCL

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

Syntax

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

Required 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

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

Optional Argument

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: DQPATTERN Function

The following example analyzes the words in the input character value. The results are written to the SAS log using the PUT statement.

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

The DQPATTERN function returns A N* M. Using the CHARACTER pattern analysis definition returns AAAAAAA N* NNAA.

DQSCHEMEAPPLY Function

Applies a scheme and returns a transformed value.

Valid in: DATA step, PROC SQL, and SCL

Requirements: If specified, the locale must be loaded into memory as part of the locale list.
Schemes using SAS format are required in the z/OS operating environment.

Syntax

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

Required 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 8](#)

Optional Arguments

mode

specifies how the scheme is to be applied to the values of the input character variable.

If the value of *scheme-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.

Default: The mode that is stored in the scheme. If a mode is not stored in the scheme, the default value of PHRASE is used.

scheme-lookup-method

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

EXACT

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

IGNORE_CASE

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

USE_MATCHDEF

specifies that the *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 *scheme-lookup-method* is `USE_MATCHDEF`.

See: [“Applying Schemes” on page 8](#)

match-definition

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](#)

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.

Range: 50 to 95

Restriction: The `sensitivity` argument is valid only when the value of the `scheme-lookup-method` argument is `USE_MATCHDEF`.

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

See: “Meta Options” on page 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.

Details

The `locale` argument is valid only when the value of the `scheme-lookup-method` argument is `USE_MATCHDEF`. The `DQSCHEMEAPPLY` 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 `DQSCHEME` procedure. Create schemes using Blue Fusion Data format with the `DQSCHEME` procedure or with the DataFlux Data Management Studio software from DataFlux (a SAS company).

Example: DQSCHEMEAPPLY Function

The following example generates a scheme with the `DQSCHEME` procedure and then applies that scheme to a data set with the `DQSCHEME` function. The example assumes that the `ENUSA` locale has been loaded into memory as 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 8, “DQSCHEME Procedure,” on page 49](#)
- [“DQSCHEMEAPPLY CALL Routine” on page 110](#)

DQSCHEMEAPPLY CALL Routine

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

Valid in: DATA step and SCL

Requirements: If specified, the locale must be loaded into memory as part of the locale list.
Schemes using SAS format are required in the z/OS operating environment.

Syntax

```

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

```

Required 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

the character variable that receives the transformed input value.

scheme

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 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.

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

the Blue Fusion Data format.

Default: BFD

NOBFD

the SAS data format. See [“Schemes” on page 7](#) for more information.

Optional Arguments**mode**

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

If the value of *scheme-lookup-method* is USE_MATCHDEF, and 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.

transform-count-variable

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

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

Interactions:

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

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

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

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

scheme-lookup-method

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

EXACT

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

IGNORE_CASE

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

USE_MATCHDEF

specifies that the *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

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.

Interactions:

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

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

If USE_MATCHDEF is specified and 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.

Default: 85

Range: 50 to 95

Interactions:

Sensitivity is valid only when the value of the *scheme-lookup-method* is USE_MATCHDEF.

If *sensitivity* is not in the scheme and USE_MATCHDEF is specified, the *sensitivity* value that is stored in the scheme 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.

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

Details

The DQSCHEMEAPPLY CALL routine transforms an input value by applying a scheme. The scheme can be in SAS format or 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: DQSCHEMEAPPLY CALL Routine

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

[Chapter 8, “DQSCHEME Procedure,” on page 49](#)

DQSTANDARDIZE Function

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

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required 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.

Optional Argument

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: DQSTANDARDIZE Function

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.

Valid in: DATA step, PROC SQL, and SCL

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

Syntax

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

Required 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

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

token

identifies the token that is returned.

Optional Argument

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: DQTOKEN Function

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

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

After the DQTOKEN call, the value for the PREFIX variable is **Mrs.**

See Also

Functions

- [“DQPARSE Function” on page 98](#)
- [“DQPARSETOKENGET Function” on page 103](#)

DQVERBF Function

Returns the version of Blue Fusion.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQVERBF ()

Details

The DQVERBF function takes no arguments and returns the version number of Blue Fusion that has been loaded by the SAS Data Quality product.

Example: DQVERBF Function

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 takes no arguments and returns a five-character string that contains the version of the currently loaded QKB. If the version cannot be determined (as with QKB versions before 2005A), the value UNKNW is returned.

Example: DQVERQKB Function

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

```
version= DQVERQKB ();
```


Chapter 11

SAS Data Quality Server System Options

SAS Data Quality Server System Options	119
Dictionary	120
DQLOCALE	120
DQOPTIONS	120
DQSETUPLOC	121

SAS Data Quality Server System Options

The SAS Data Quality Server system options DQLOCALE= and DQSETUPLOC= must be asserted before you run data cleansing programs. The DQOPTIONS= system option is used at SAS invocation to set data quality parameters.

To specify values for the DQLOCALE= and DQSETUPLOC= system options, use the “[%DQLOAD AUTOCALL Macro](#)” on page 61.

CAUTION:

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

System Options:

- DQLOCALE must be run prior to running data cleansing programs. See “[DQLOCALE](#)” on page 120 for additional information.
- The DQOPTIONS= system option enables you to optimize your SAS session for data quality. The value of the system option is a set of option-value pairs that you specify on the SAS start-up command or in the SAS configuration file. The data quality system options can be referenced by the OPTIONS procedure by specifying GROUP=DATAQUALITY. See: “[DQOPTIONS](#)” on page 120.
- DQSETUPLOC must be run prior to running data cleansing programs. See “[DQSETUPLOC](#)” on page 121 for additional information.

Dictionary

DQLOCALE

Specifies an ordered list of locales.

Requirement: You must specify at least one locale.

Syntax

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

Action

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.

CAUTION:

It is recommended that you invoke the AUTOCALL macro `%DQLOAD` at the beginning of each data cleansing program or session. Failure to do so might generate unintended output.

SAS specifies no default value for the DQLOCALE= system option. It is recommended that you *not* use an AUTOEXEC to load default locales when you invoke SAS. Loading default locales can enable you to apply the wrong locales to your data, which generates unintended output. Loading default locales also wastes resources when you are not cleansing data. Instead of loading default locales, invoke the `%DQLOAD` macro at the beginning of each data cleansing program or session. See “[%DQLOAD AUTOCALL Macro](#)” on page 61 for additional information.

DQOPTIONS

Specifies SAS session parameters for data quality programs.

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

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

Syntax

DQOPTIONS = (*label*=*value*)

Required Arguments

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 Data Management Servers. The Wireline protocol improves data transfer performance in z/OS. In the SAS Data Quality Server software, z/OS support encompasses the DMSRVDATASVC procedure and all functions.

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

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 root directory of the Quality Knowledge Base.

Syntax

DQSETUPLOC= ('path-specification')

Required Argument

PATH-SPECIFICATION= QUALITY KNOWLEDGE BASE ROOT-DIRECTORY

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

See: [“Load and Unload Locales” on page 6.](#)

Appendix 1

Deprecated Language Elements

Dictionary	123
PROC DQSRVADM	123
PROC DQSRVSVC	123
DQSRVARCHJOB Function	124
DQSRVCOPYLOG Function	124
DQSRVDELETELOG Function	124
DQSRVJOBSTATUS Function	124
DQSRVKILLJOB Function	125
DQSRVPROFJOBFILE Function	125
DQSRVPROFJOBREP Function	125
DQSRVUSER Function	125
DQSRVVER Function	126

Dictionary

PROC DQSRVADM

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

See Also

- PROC DMSRVADM
- *SAS Data Quality Server 9.2*

PROC DQSRVSVC

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

See Also

- PROC DMSRVDATASVC

- *SAS Data Quality Server 9.2*

DQSRVARCHJOB Function

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

See Also

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

DQSRVCOPYLOG Function

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

See Also

- DMSRVCOPYLOG function
- *SAS Data Quality Server 9.2*

DQSRVDELETELOG Function

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

See Also

- DMSRVDELETELOG function
- *SAS Data Quality Server 9.2*

DQSRVJOBSTATUS Function

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

See Also

- DMSRVJOBSTATUS function
- *SAS Data Quality Server 9.2*

DQSRVKILLJOB Function

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

See Also

- DMSRVKILLJOB function
- *SAS Data Quality Server 9.2*

DQSRVPROFJOBFILE Function

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

See Also

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

DQSRVPROFJOBREP Function

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

See Also

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

DQSRVUSER Function

Deprecated. Registers a user on a DataFlux Integration Server.

See Also

- DMSRVUSER function
- *SAS Data Quality Server 9.2*

DQSRVVER Function

Deprecated. Returns the version of the DataFlux Integration Server.

See Also

- DMSRVVER function
- *SAS Data Quality Server 9.2*

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 on character values a consistent usage of uppercase and lowercase letters.

cleanse

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

cluster

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

composite match code

a match code that consists of a concatenation of 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 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 serves as a boundary that separates the elements of a text string.

gender definition

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

guess definition

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

identification definition

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

locale

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

locale list

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

match

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

match code

an encoded version of a character value that is created as a basis for data analysis and data cleansing. 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. In the ENUSA locale. For example, match definitions are provided for names, e-mail addresses, and street addresses, among others.

name prefix

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

name suffix

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

parse

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

parse definition

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

parse token

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

parsed string

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

Quality Knowledge Base

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

scheme

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

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

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

token

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

transformation

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

transformation value

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

Index

Special Characters

! (exclamation point)
 as delimiter 37
 %DQPUTLOC autocall macro 4, 62
 %DQUNLOAD autocall macro 64
 “Cluster Definition “ 12

A

analysis data sets 8
 creating 53, 56
 ANALYSIS= option
 CREATE statement (DQSCHEME) 53
 apply mode 9, 10
 applying
 schemes 8
 authentication 27, 33, 80

B

BFD format
 See Blue Fusion Data (BFD) format
 BFDTOSAS option
 CONVERT statement (DQSCHEME)
 52
 blank spaces, removing 28
 blank values 36
 BLOCKSIZE= option
 PROC DMSRVDATASVC statement
 26
 Blue Fusion Data (BFD) format 7, 9, 50
 converting SAS schemes to 7, 52
 creating BFD schemes 58
 version of BFD 117

C

CALL routine 110
 CALL routines 66
 scheme CALL routines 71
 case definitions 18, 82

case functions 68
 case of character values 82
 category names
 from character values 90
 character values
 after standardization 114
 case and standardization definitions 18
 case of 82
 category names from 90
 gender analysis, locale guess, and
 identification definitions 19
 inserting tokens into extraction values
 86
 inserting tokens into parsed values 104
 locale names from 91
 match codes from 94
 match codes from parsed values 96
 parsed 99
 pattern analysis definitions 19
 pattern analysis from input values 105
 tokens from 115
 tokens from extraction values 85
 tokens from parsed values 103
 updated extraction values 86
 updated parsed values 104
 cleaning up jobs and logs 24
 cleansing data
 real-time 25
 CLUSTER_BLANKS option
 PROC DQMATCH statement 36
 cluster numbers 35
 assigning 39
 creating 40
 CLUSTER= option
 PROC DQMATCH statement 36
 clustering with exact criteria
 DQMATCH procedure, CRITERIA
 statement 13
 clusters
 householding 12
 minimal sensitivity 43

- mixed sensitivity 41
- with exact criteria 13
- with multiple CRITERIA statements 46
- CLUSTERS_ONLY option
 - PROC DQMATCH statement 37
- composite match codes 11, 40
- CONDITION= option
 - CRITERIA statement (DQMATCH) 38
- configuration
 - SAS sessions for data quality 4
- CONVERT statement, DQSCHEME
 - procedure 52
- converting schemes 7, 52
- count of locale definitions 92
- CREATE statement, DQSCHEME
 - procedure 53
- CRITERIA statement, DQMATCH
 - procedure 38
 - clustering with multiple 46

D

- data cleansing
 - real-time 25
 - specifying definitions in programs 4
- data quality
 - SAS session parameters for programs 120
- data sets
 - analysis 8, 53, 56
 - creating 53
 - input 28, 33
 - job status 21, 22, 23
 - output 28, 34
 - scheme 7, 8, 9, 53
- data values, similar
 - transforming 49
- DATA= option
 - PROC DMSRV DATASVC statement 27
 - PROC DQMATCH statement 37
 - PROC DQSCHEME statement 51
- DataFlux Data Management
 - running services 34
- DataFlux Data Management Profile
 - running profile jobs 79
- DataFlux Data Management Server
 - authenticating users 33, 80
 - cleaning up jobs and logs 24
 - copying log files 73
 - deleting log files 74
 - functions 68
 - host machine 27, 32
 - host of 22
 - input and output columns 27, 33
 - job information 21
 - job status 76
 - jobs and services 5
 - passwords 6
 - port number 22, 27, 33
 - processing services 31
 - running jobs and services 5
 - running real-time services 25
 - s service identification 27
 - security 23
 - service identification 33
 - version of 81
- DataFlux Data Management Studio 50
 - creating jobs and services 5
 - job identifiers 72
 - running jobs 72
 - running real-time services 25
 - running services 29
- DataFlux Data Management Server
 - generating Profile jobs 79
 - running Data Management Studio jobs 72
- DataFlux DataManagement Server
 - authenticating users 27
- DataFlux dfPower Architect 2
- DataFlux dfPower Profile 2
- DataFlux Integration Server 1
- DataFlux Management Server
 - terminating jobs 77
- DataFlux Profile jobs 79
- date standardization
 - in EN locale 18
- default length of parsed input 101
- definitions
 - available in locales 4
 - case 82
 - case and standardization 18
 - date standardization in EN locale 18
 - displaying information about locale 4
 - extraction 16
 - gender 88
 - gender analysis, locale guess, and identification 19
 - match definitions 17
 - parse 15
 - pattern analysis 19
 - revealing or hiding non-surfaced 97
 - scheme-build match 17
 - specifying in data cleansing programs 4
- delimiter
 - exclamation point as 37
- DELIMITER option
 - PROC DQMATCH statement 37
- DELIMSTR option
 - CRITERIA statement (DQMATCH) 38
- DMSRVADM procedure 21
 - cleaning up jobs and logs 24

- examples 23
 - job status data sets 22, 23
 - PROC DMSRVADM statement 22
 - security and 23
 - syntax 22
 - DMSRVBATCHJOB function 72
 - DMSRVCOPYLOG function 73
 - DMSRVDATASVC procedure
 - examples 29
 - input and output data sets 28
 - syntax 26
 - timeout 28
 - DMSRVDATASVC Procedure 25
 - DMSRVDELETELOG function 74
 - DMSRVJOBSTATUS function 76
 - DMSRVKILLJOB function 77
 - DMSRVPROCESSSVC procedure
 - examples 34
 - input and output data sets 33
 - DMSRVPROCESSSVC Procedure 31
 - DMSRVPROFILEJOB function 79
 - DMSRVUSER function 80
 - DMSRVVER function 81
 - DQCASE function 82
 - DQEXTINFOGET function 83
 - DQEXTTOKENGET function 85
 - DQEXTTOKENPUT function 86
 - DQGENDER function 87
 - DQGENDERINFOGET function 88
 - DQGENDERPARSED function 89
 - DQIDENTIFY function 90
 - DQLOCALE= system option 120
 - DQLOCALEGUESS function 91
 - DQLOCALEINFOGET function 92
 - DQLOCALEINFOLIST function 92
 - DQMATCH function 94
 - DQMATCH procedure 35
 - clustering with minimal sensitivity 43
 - clustering with mixed sensitivities 41
 - clustering with multiple CRITERIA statements 46
 - CRITERIA statement 38, 46
 - examples 40
 - generating composite match codes 40
 - generating multiple simple match codes 47
 - householding 12
 - match codes for parsed values 44
 - matching values with mixed sensitivity levels 41
 - PROC DQMATCH statement 36
 - syntax 36
 - DQMATCHINFOGET function 95
 - DQMATCHPARSED function 96
 - DQOPTIONS= system option 120
 - configuring SAS sessions 4
 - DQOPTSURFACE function 97
 - DQPARSE CALL routine 99
 - DQPARSEINFOGET function 100
 - DQPARSEINPUTLEN function 101
 - DQPARSERESLIMIT function 102
 - DQPARSESCORDEPTH function 103
 - DQPARSETOKENGET function 103
 - DQPARSETOKENPUT function 104
 - DQPATTERN function 105
 - DQSCHEME procedure 49
 - APPLY statement 51
 - applying schemes 59
 - CONVERT statement 52
 - CREATE statement 53
 - creating analysis data sets 56
 - creating BFD schemes 58
 - creating schemes 57
 - examples 56
 - PROC DQSCHEME statement 50
 - scheme-build match definitions 17
 - syntax 50
 - DQSETUPLOC= system option 121
 - DQSRVSVC procedure
 - timeout 33
 - DQSTANDARDIZE function 114
 - DQTOKEN function 115
 - DQVERBF function 117
 - DQVERQKB function 117
- E**
- element mode 9
 - EN locale
 - date standardization in 18
 - encoded passwords 6
 - EXACT option
 - CRITERIA statement (DQMATCH) 39
 - exclamation point
 - as delimiter 37
 - extraction character values
 - inserting tokens into 86
 - tokens from 85
 - updated 86
 - extraction definition functions 70
 - extraction definitions
 - extract 16
 - token names in 83
 - extraction input 16
 - extraction values
 - updated 86
- F**
- functions 66
 - case 68
 - DataFlux Data Management Server 68

- extraction 70
- gender analysis, locale guessing, and identification 69
- listed alphabetically 66
- listed by category 68
- matching 69
- parsing 69
- pattern analysis 70
- reporting 70
- scheme 71
- standardization 71

G

- gender analysis definitions 19
- gender analysis functions 69
- gender definitions
 - parse definitions associated with 88
- gender determination
 - from name of an individual 87
 - from parsed name 89
- generating jobs
 - Profile jobs 79
- global parse
 - global 16

H

- hiding non-surfaced definitions 97
- host
 - for DataFlux Data Management Server 22
- host machine
 - DataFlux Data Management Server 27, 32
- HOST= option
 - PROC DMSRVADM statement 22
 - PROC DMSRVDATASVC statement 27
 - PROC DMSRVPROCESSVC statement 32

I

- identification definitions 19
- identification functions 69
- IGNORE_CASE option
 - APPLY statement (DQSCHEME) 52
 - CREATE statement (DQSCHEME) 55
- IN= option
 - CONVERT statement (DQSCHEME) 52
- INCLUDE_ALL option
 - CREATE statement (DQSCHEME) 53
- input
 - default length of parsed 101

- extraction values 16
 - parsed values 15
- input character values
 - pattern analysis from 105
- input columns, input and output 27, 33
- input data sets
 - DMSRVDATASVC procedure 28
 - DMSRVPROCESSVC procedure 33
- installation 4

J

- job identifiers
 - DataFlux Data Management Studio jobs 72
 - Profile jobs 79
- job status data sets 21, 22
 - generating 23
 - location of 22
- jobs 5
 - cleaning up 24
 - copying log files 73
 - creating 5
 - deleting log files 74
 - generating Profile jobs 79
 - information about 21
 - running 5
 - running DataFlux Data Management Studio jobs 72
 - status of 76
 - terminating 77

K

- killing jobs 77
 - terminating 5

L

- length
 - default length of parsed input 101
- Local Process group 1
- locale definitions
 - case and standardization 18
 - count of 92
 - date standardization in EN locale 18
 - gender analysis, locale guess, and identification 19
 - global parse 16
 - locale 15
 - match 17
 - name of 92
 - parse 15
 - pattern analysis 19
 - scheme-build match 17
- locale guess definitions 19

locale guessing functions 69
 locale names
 from character values 91
 LOCALE= option
 APPLY statement (DQSCHEME) 51
 CREATE statement (DQSCHEME) 54
 PROC DQMATCH statement 37
 locales
 definitions available in 4
 displaying information about 4
 displaying information in SAS log 62
 getting information about 92
 ordered list of 120
 unloading to use free memory 64
 updating 7
 log files
 copying 73
 deleting 74
 logs
 cleaning up 24
 displaying locale information 62
 for jobs and services 5
 lookup method 9, 10, 52, 55

M

match codes 35
 composite 40
 Composite 11
 creating 10, 38
 creating for parsed values 44
 from character values 94
 from parsed character values 96
 generating multiple simple codes 47
 length of 12
 match definitions and 17
 sensitivity 13
 simple 11, 47
 truncation of 12
 match definitions
 parse definitions associated with 95
 scheme-build 17
 MATCH-DEFINITION= option
 APPLY statement (DQSCHEME) 51
 MATCHCODE= option
 CRITERIA statement (DQMATCH) 39
 PROC DQMATCH statement 37
 MATCHDEF option
 CRITERIA statement (DQMATCH) 39
 MATCHDEF= option
 CREATE statement (DQSCHEME) 54
 matching functions 69
 matching values
 default sensitivity 40
 minimal sensitivity 43
 mixed sensitivity 41

memory
 unloading locales from 64
 meta options 9
 MISSINGVARSOK option
 PROC DMSRVDATASVC statement 27
 mode of scheme application 51, 54
 MODE=ELEMENT option
 APPLY statement (DQSCHEME) 51
 CREATE statement (DQSCHEME) 54

N

named tokens 15, 16, 83, 100
 names
 gender determination and 87, 89
 locale names from character values 91
 of locale definitions 92
 of parse definitions 88, 95
 parsed 89
 NO CLUSTER_BLANKS option
 PROC DQMATCH statement 36
 NODELIMITER option
 PROC DQMATCH statement 37
 non-surfaced definitions
 revealing or hiding 97
 NOPRINT option
 PROC DMSRVDATASVC statement 27
 PROC DMSRVPROCESSVC statement 32

O

option
 PROC DQSCHEME statement 50
 OUT= option
 CONVERT statement (DQSCHEME) 53
 PROC DMSRVADM statement 22
 PROC DMSRVDATASVC statement 27
 PROC DMSRVPROCESSVC statement 32
 PROC DQMATCH statement 38
 PROC DQSCHEME statement 51
 output columns 27, 33
 output data sets
 DMSRVDATASVC procedure 28
 DMSRVPROCESSVC procedure 34

P

PARMLIST option
 PROC DMSRVDATASVC statement 27

- PARMLIST= option
 - PROC DMSRVPROCESSVC
 - statement 32
- parse definitions 15, 16
 - associated with gender definitions 88
 - associated with match definitions 95
 - name of 88, 95
 - token names in 100
- parsed character values 99
 - inserting tokens into 104
 - match codes from 96
 - tokens from 103
 - updated 104
- parsed input 15
 - default length of 101
- parsed names
 - gender determination from 89
- parsed values
 - match codes for 44
 - updated 104
- parsing
 - resource limit during 102
- parsing functions 69
- parsing scores
 - searching for 103
- PASSWORD= option
 - PROC DMSRVDATASVC statement 27
 - PROC DMSRVPROCESSVC
 - statement 33
- passwords
 - for DataFlux Data Management Server 6
- pattern analysis
 - from input character values 105
- pattern analysis definitions 19
- pattern analysis functions 70
- phrase mode 9
- PHRASE option
 - APPLY statement (DQSCHEME) 51
 - CREATE statement (DQSCHEME) 54
- port number 22, 27, 33
- PORT= option
 - PROC DMSRVADM statement 22
 - PROC DMSRVDATASVC statement 27
 - PROC DMSRVPROCESSVC
 - statement 33
- PROC DMSRVADM statement 22
- PROC DMSRVDATASVC statement 26
- PROC DQMATCH statement 36
- procedure 49
- process
 - Data Management service 31
 - service 31

Q

- Quality Knowledge Base (QKB) 1
 - location of root directory 121
 - system access 3
 - version of currently loaded 117

R

- real-time data cleansing 25
- reporting functions 70
- resource limit
 - during parsing 102
- revealing non-surfaced definitions 97
- root directory
 - specifying location of 121
- running jobs 5
 - DataFlux Data Management Studio 72
- running services 5
 - Data Management Server 31
 - real-time 25

S

- SAS Data Quality Server 1
 - concepts 3
 - functional overview 1
 - installing 4
 - Quality Knowledge Base 3
 - updating 4
- SAS format schemes
 - converting BFD format to 7, 52
- SAS language elements 1
- SAS log
 - displaying locale information 62
- SAS Macro 64
- SAS sessions
 - configuring for data quality 4
 - parameters for data quality programs 120
- SASTOBFD option
 - CONVERT statement (DQSCHEME) 52
- SCHEME_LOOKUP= EXACT option
 - APPLY statement (DQSCHEME) 52
 - CREATE statement (DQSCHEME) 55
- scheme CALL routines 71
- scheme data sets
 - applying schemes 8
 - format of 9
- scheme functions 71
- SCHEME= option
 - APPLY statement (DQSCHEME) 52
 - CREATE statement (DQSCHEME) 54
- schemes
 - analysis data sets 8
 - apply mode 9, 10

- applying 59, 110
- applying to transform values of a single variable 51
- converting between formats 7, 52
- creating 7, 57
- creating BFD schemes 58
- format of 50
- meta options 9
- mode of application 51, 54
- returning transformation flag 110
- returning transformed value 110
- searching
 - for parsing scores 103
- security
 - DMSRVADM procedure and 23
- sensitivity level 13
 - analysis data sets and 8
 - default value 40
 - match codes and 35, 39
 - match definitions and 17
 - meta options and 10
 - minimal 43
 - mixed 41
- SENSITIVITY= option
 - APPLY statement (DQSCHEME) 52
 - CREATE statement (DQSCHEME) 55
 - CRITERIA statement (DQMATCH) 39
- Server Process group 1
- SERVICE= option
 - PROC DMSRVDATASVC statement 27
 - PROC DMSRVPROCESSVC statement 33
- SERVICEINFO option
 - PROC DMSRVDATASVC statement 27
 - PROC DMSRVPROCESSVC statement 33
- services 5
 - creating 5
 - identification of 27, 33
 - logs for 5
 - running 5, 29, 31, 34
 - running real-time 25
- setup file
 - specifying location of 121
- similar data values 49
 - transforming 49
- simple match codes 11
 - generating multiple 47
- standardization 114
 - date standardization in EN locale 18
- standardization definitions 18
- standardization functions 71
- system options 119

T

- terminating jobs 77
 - logs for 5
- TIMEOUT option
 - PROC DMSRVDATASVC statement 28
 - PROC DMSRVPROCESSVC statement 33
- TIMEOUT= option
 - PROC DMSRVDATASVC statement 28
 - PROC DMSRVPROCESSVC statement 33
- token names 15, 16, 83, 100
- tokens
 - from character values 115
 - from extraction character values 85
 - from parsed character values 103
 - inserting into extraction character values 86
 - inserting into parsed character values 104
 - updated extraction character values 86
 - updated parsed character values 104
- transformation flags 110
- transforming
 - applying schemes to transform values of a single variable 51
 - returning transformed values after applying a scheme 110
- TRIM option
 - PROC DMSRVDATASVC statement 28
- truncated match codes 12

U

- unloading locales 64
- updated extraction values 86
- updated parsed values 104
- updated tokens 86, 104
- updating
 - SAS Data Quality Server 4
- USE_MATCHDEF option
 - APPLY statement (DQSCHEME) 52
 - CREATE statement (DQSCHEME) 55
- user authentication 27, 33, 80
- USERID= option
 - PROC DMSRVDATASVC statement 28
 - PROC DMSRVPROCESSVC statement 33

V

- VAR option

- CRITERIA statement (DQMATCH) [38](#)
- VAR= option
 - APPLY statement (DQSCHEME) [52](#)
 - CREATE statement (DQSCHEME) [56](#)
- variables
 - applying schemes to transform values of a single variable [51](#)
- version
 - Blue Fusion Data [117](#)
 - currently loaded QKB [117](#)
 - DataFlux Data Management Server [81](#)