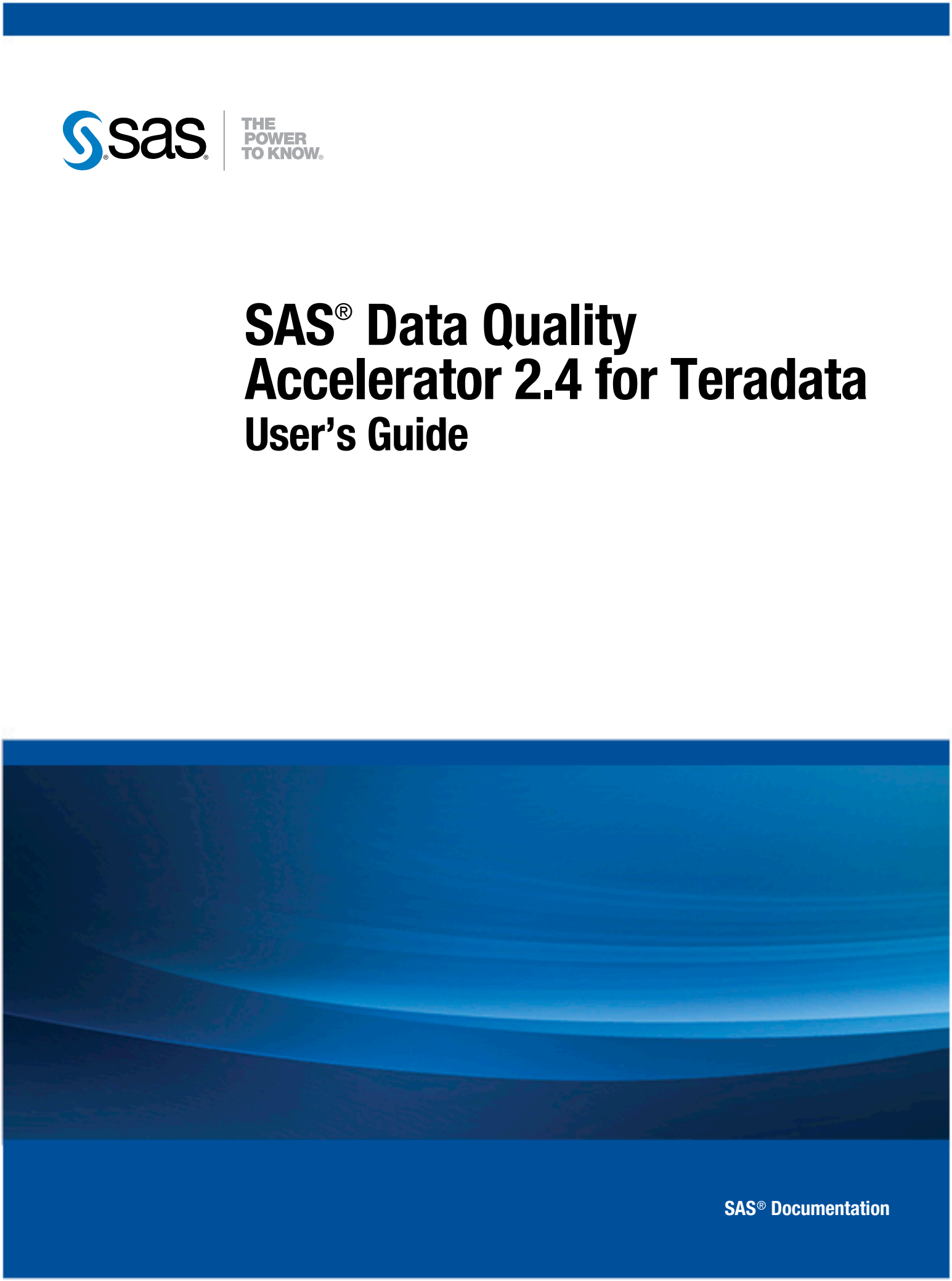


SAS® Data Quality Accelerator 2.4 for Teradata

User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® Data Quality Accelerator 2.4 for Teradata: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® Data Quality Accelerator 2.4 for Teradata: User's Guide

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

September 2013

SAS 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 **support.sas.com/bookstore** or call 1-800-727-3228.

Contents

Accessibility Features of SAS Data Quality Accelerator for Teradata

Overview

SAS Data Quality Accelerator for Teradata has a command-line-only interface that is accessible using a keyboard or alternative keyboard assistive technologies. For this release, no accessibility testing was done and no additional features were added to address accessibility. If you have specific questions about the accessibility of SAS products, send them to **accessibility@sas.com** or call SAS Technical Support.

For information about the accessibility of the other products mentioned in this document, including DataFlux Data Management Studio and DataFlux Quality Knowledge Database, see the documentation for that product.

Documentation Format

Please contact **accessibility@sas.com** if you need this document in an alternative digital format.

Recommended Reading

Here is the recommended reading list for this title:

- *SAS In-Database Products: Administrator's Guide*
- *SAS In-Database Products: User's Guide*
- *SAS Data Quality Server: Reference*
- *SAS/ACCESS for Relational Databases: Reference*
- *DataFlux Data Management Studio: User's Guide*
- DataFlux Quality Knowledge Base documentation that is available from SAS Documentation:
  - What's New in DataFlux Quality Knowledge Base
  - DataFlux Quality Knowledge Base Online Help

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

SAS Books
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

Chapter 1

Introduction

Introduction to the SAS Data Quality Accelerator for Teradata

SAS applications are often built to work with large volumes of data in environments that demand rigorous IT security and management. When the data is stored in an external database, such as a Teradata database, the transfer of large data sets to the computers that run the SAS System can cause a performance bottleneck. There are also possible unwanted security and resource management consequences for local data storage. SAS Data Quality Accelerator for Teradata addresses these challenges by moving computational tasks closer to the data and by improving the integration between the SAS System and the database management system (DBMS).

SAS Data Quality Accelerator for Teradata provides in-database data quality functionality. The data quality functionality is provided as Teradata stored procedures. A stored procedure is a subroutine that is stored in the database and is available to applications that access a relational database.

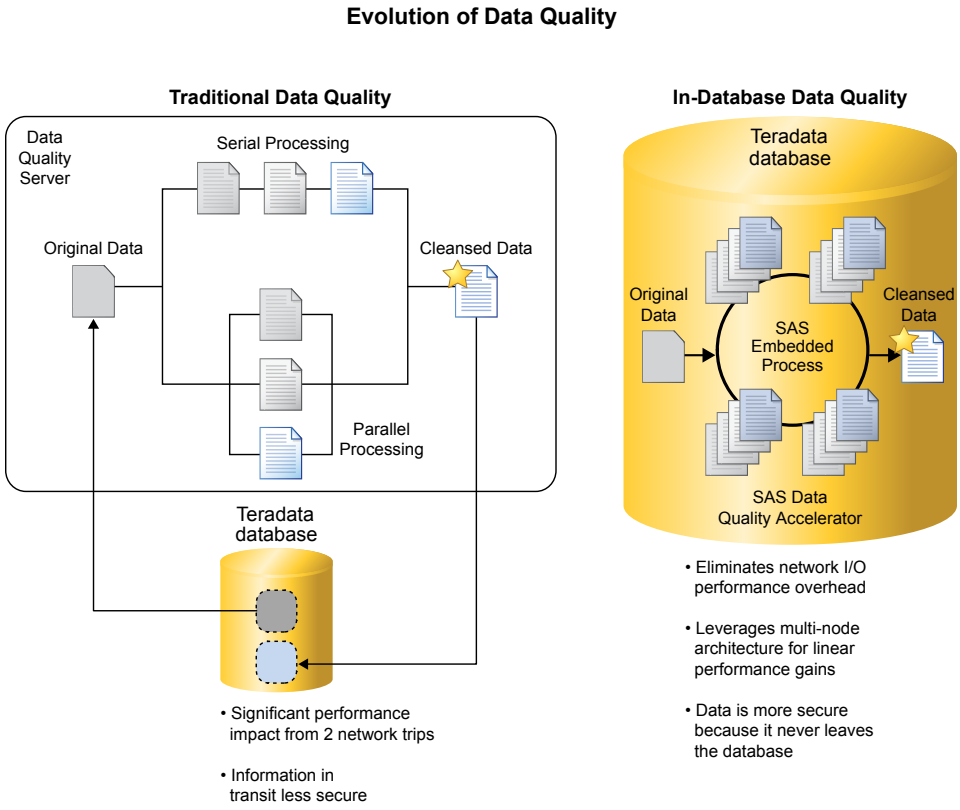
The stored procedures provide the following data quality operations:

- upper-casing, lower-casing, and proper-casing
- attribute extraction
- gender analysis
- identification analysis
- matchcode generation
- parsing
- pattern analysis

- standardization

Benefits of In-Database Processing

The following diagram illustrates the difference between in-database and outside-of-database data quality operations.



Executing data quality operations inside of the database rather than as a separate utility outside of the database provides the following benefits:

- eliminates network I/O performance
- leverages multi-node architectures for linear performance gains
- makes information more secure because it never leaves the database

Benefits of Teradata Stored Procedure Technology

The availability of the in-database data quality operations as Teradata stored procedures offers the following benefits:

- Because they are self-contained, the data quality stored procedures provide a good way to enforce standards for commonly used processes and avoid same-code proliferation.

- Programmatically, the data quality stored procedures can be nested in the definition of other stored procedures.
- The data quality stored procedures can be run anywhere a Teradata stored procedure can be run. They can be executed interactively by a user with a command-line interface such as the Teradata BTEQ utility. Or they can be executed in a client program by any language that supports a Teradata connection or a generic ODBC connection. Examples include the Base SAS SQL procedure; the Teradata Tools and Utilities (TTTU); scripting languages, such as Perl or Python; and third-party ETL tools.

Comparison to SAS Data Quality Server Functions

The data quality stored procedures provide similar data quality operations to those available with the following SAS Data Quality Server functions.

Table 1.1 *Data Quality Stored Procedures and Comparable SAS Data Quality Server Functions*

Stored Procedure Name	Function Name	Description
DQ_EXTRACT()	DQEXTRACT()	Extracts specific entities or attributes from a text string.
DQ_GENDER()	DQGENDER()	Determines the gender of a person from their name or other information.
DQ_IDENTIFY()	DQIDENTIFY()	Determines the type of data that is represented by a text string.
DQ_LOWERCASE()	DQCASE()	Lowercases text.
DQ_MATCH()	DQMATCH()	Generates a matchcode for a text string.
DQ_PARSE()	DQPARSE() DQPARSETOKENGET()	Segments a string into semantically atomic tokens.
DQ_PATTERN()	DQPATTERN()	Returns a simple representation of a text string's character pattern.
DQ_PROPERCASE()	DQCASE()	Applies uppercase and lowercase lettering using context-sensitive rules.
DQ_STANDARDIZE()	DQSTANDARDIZE()	Generates a preferred standard representation of a string.
DQ_UPPERCASE()	DQCASE()	Uppercases text.

Stored Procedure Name	Function Name	Description
DQ_SET_LOCALE()	Not applicable (the locale is set by other means)	Sets the locale for data quality operations in the SQL session.

The data quality stored procedures and SAS Data Quality Server functions differ in the following ways:

- SAS Data Quality Server functions operate outside of the database.
- SAS Data Quality Server functions operate on one data line at a time. The data quality stored procedures operate on the contents of an entire column of an input table.
- SAS Data Quality Server functions are applied dynamically. The data quality stored procedures can return output in a physical table or dynamically, as a *cursor*, which is a control structure that enables traversal over the records in a result set.
- A SAS language is required to use the SAS Data Quality Server functions. The data quality stored procedures can be run by any language that supports the Teradata SQL dialect.

Components of the Accelerator

SAS Data Quality Accelerator for Teradata is made up of the following components:

1. SAS Embedded Process for Teradata. The SAS Embedded Process is a SAS server process that runs within Teradata to read and write data. Currently, the SAS Embedded Process for Teradata can be installed only on a Teradata Linux server.
2. DataFlux Quality Knowledge Base (QKB), a collection of data quality rules provided as part of the SAS Data Quality software offering. The QKB comes equipped with a standard collection of data quality rules, which you can modify for your needs in DataFlux Data Management Studio. The QKB can originate from a Windows or UNIX system.
3. Tools for packaging and deploying the QKB and SAS Embedded Process on the Teradata nodes.
4. Tools for creating and administering the data quality stored procedures in the Teradata database.
5. The data quality stored procedures.

Each component has separate installation and maintenance steps. For more information, see Chapter 2, “Installation,” on page 7.

About This Document

This document describes how to install the data quality stored procedures in the Teradata database. It also describes how to use the data quality stored procedures to improve data quality at your site.

Audience

The audience for this product is IT organizations that want to improve data quality so that their programs run on clean data.

Chapter 2

Installation

Prerequisites

The following products need to be installed along with SAS Data Quality Accelerator for Teradata:

- SAS Foundation
- SAS/ACCESS Interface to Teradata
- SAS Data Quality Standard software offering

SAS/ACCESS Interface to Teradata contains the SAS Embedded Process for Teradata.

The SAS Data Quality Standard software offering represents the minimum level of data quality functionality that must be available to use the SAS Data Quality Accelerator for Teradata. More advanced data quality offerings can be used, but they are not required. The SAS Data Quality Standard software offering includes DataFlux Quality Knowledge Base (QKB) and DataFlux Data Management Studio software. The QKB is required by the accelerator. DataFlux Data Management Studio software enables you to customize the definitions in the DataFlux Quality Knowledge Base.

A Teradata database version of 13.10.02.01 or later is required. Each Teradata node needs approximately 200MB of disk space in the **/opt** file system for the SAS Embedded Process and approximately 8 GB for the QKB.

Post-Installation Steps

After the required products are installed, the SAS Data Quality Accelerator for Teradata requires the following additional setup before it can be used:

1. Customize the DataFlux Quality Knowledge Base (optional).
2. Package the QKB for deployment (required).
3. Install the SAS Embedded Process for Teradata package, SAS Embedded Process support functions, and QKB package on the Teradata nodes.
4. Run accelerator shell scripts to create the stored procedures in the Teradata database and grant users permissions to the stored procedures.

Multi-Node Deployments

The SAS Embedded Process and QKB packages must be deployed on all Teradata nodes. These instructions use the Teradata Parallel Upgrade Tool to deploy the package files. This tool deploys the software on all nodes simultaneously.

Shell scripts that create (or remove) the stored procedures are run once, in the Teradata database, using SQL statements.

Overview of SAS Data Quality Accelerator for Teradata Scripts

The SAS Data Quality Accelerator for Teradata comes with four scripts:

- a QKB packaging script named qkb_pack
- stored procedure installation and uninstallation scripts, named dq_install.sh and dq_uninstall.sh, respectively
- a user authorization script named dq_grant.sh

The scripts are created in the ***<SAS-installation-directory>*/SASTKInDatabaseServer/*<release-number>*/TeradataonLinux/misc/deployment** directory of the SAS/ACCESS installation. The initial release number is 9.4.

All of the shell scripts except qkb_pack need to be run from a system with a UNIX shell and BTEQ available. This system does not necessarily have to be any of the database systems themselves. BTEQ can be configured to access the Teradata database over the network from a non-database system.

The dq_install.sh, dq_uninstall.sh, and dq_grant.sh shell scripts must be run by the Teradata systems administrator.

Customizing the QKB

The standard definitions in the DataFlux Quality Knowledge Base are sufficient for performing most data quality operations. However, you can use the Customize feature of DataFlux Data Management Studio to modify the QKB definitions to meet specific needs. For general instructions about customizing QKBs, see the chapter "Customizing Quality Knowledge Bases" in the Help for DataFlux Data Management Studio.

SAS provides regular updates to the QKB. It is recommended that you update your QKB each time a new one is released. Check the latest release notes to determine whether definitions used by the data quality stored procedures have been updated. Decide whether you need any of the new locale support that might have been added. (If you decide that you need new locale support, please contact SAS to revise your license agreement.) For a listing of the latest enhancements to the QKB, refer to "What's New in DataFlux Quality Knowledge Base". The What's New document is available on the DataFlux Quality Knowledge Base (QKB) product documentation page at **`support.sas.com`**. Either search on the product name or locate it in the product index.

The accelerator supports one QKB in the Teradata database in the initial release. In future releases, there are plans to support multiple QKBs. If you customize your QKB, you will need to evaluate and carry over any customizations when you update the QKB.

Packaging the QKB for Deployment

The qkb_pack script is provided to package the QKB for deployment on the Teradata nodes. The qkb_pack script is run on an existing QKB to create an .rpm file. An .rpm file is a file that is suitable for installation on Linux systems by Linux package management software. Windows and UNIX versions of qkb_pack are available.

Here is the syntax for executing qkb_pack:

Windows:

```
qkb_pack.cmd qkb-dir out-dir
```

UNIX:

```
./qkb_pack.sh qkb-dir out-dir
```

qkb-dir
specifies the path to the QKB. Use the name of the QKB's root directory. Typically, the root directory is found at the following locations:

Windows XP:	C:\Documents and Settings\All Users\Application Data \DataFlux\QKB. Specify the full pathname to the root directory (for example: C:\Documents and Settings\All Users \Application Data\DataFlux\QKB\CI\2013A).
Vista, Windows 7:	C:\ProgramData\DataFlux\QKB (example of full pathname: C:\ProgramData\DataFlux\QKB\CI\2013A).
UNIX:	/opt/dataflux/qkb/share

out-dir
specifies the directory where the package file is created. This argument is required.

The output package file has a name in the following form:

```
sasqkb_product-version-timestamp.noarch.rpm
```

product
is a 2-character product code for the QKB, such as CI (for Contact Information) or PD (for Product Data).

version
is the version number of the QKB.

timestamp
is a UNIX datetime value that indicates when qkb_pack was invoked. A UNIX datetime value is stored as the number of seconds since January 1, 1970.

noarch
indicates the package file is an XML file, which is platform-independent.

Here is an example of an output filename, representing the QKB for Contact Information 22:

Windows:

```
sasqkb_ci-22.0-1367606747659.noarch.rpm
```

UNIX:

```
sasqkb_ci-22.0-1367606747659.rpm
```

Put the sasqkb package file on your Teradata database server in a location where it is both Read and Write accessible. The package file must be readable by the Teradata Parallel Upgrade Tool. You need to move this package file to the server machine in accordance with procedures used at your site.

After the package is on the Teradata database server, install the sasqkb package at the same time that you install the SAS Embedded Process package. For more information, see “Installing the SAS Embedded Process and Support Functions” on page 10.

Installing the SAS Embedded Process and Support Functions

The in-database deployment package for Teradata includes two deployment package files: a SAS Formats Library package and a SAS Embedded Process package. Only the SAS Embedded Process package, tkindbsrv, is required to be deployed for SAS Data Quality Accelerator for Teradata. The accelerator requires tkindbsrv-9.41-1.x86._64.rpm or later.

In addition to tkindbsrv, you must install the SAS Embedded Process support function package. The support function package is provided by Teradata.

The steps for installing the SAS Embedded Process in the Teradata database differ depending on whether an earlier version of the embedded process already exists in the database.

Run this command on the Teradata nodes to check for the current installed version of the SAS Embedded Process.

```
psh "rpm -q -a" | grep tkindbsrv
```

If a previous version is installed, a result similar to this is displayed. The version number might be different.

```
tkindbsrv-9.41-1
```

If the SAS Embedded Process is not installed on the Teradata nodes, no output is displayed.

Instructions for installing the SAS Embedded Process and SAS Embedded Process support functions are provided in the Teradata section of the *SAS In-Database Products: Administrator's Guide*. See "Upgrading from or Reinstalling a Previous Version" or "Installing the SAS Formats Library and the SAS Embedded Process," as appropriate.

When you get to the topic, "Installing the SAS Formats Library and the SAS Embedded Process with the Teradata Parallel Upgrade Tool" in the *SAS In-Database Products: Administrator's Guide*, follow the steps provided to install the tkindbsrv package and the sasqkb package. The SAS Embedded Process should be installed in the **/opt** file system of the Teradata nodes. The QKB will automatically be installed in the **/opt/qkb/default** directory. Then see "Installing the SAS Embedded Process Support Functions" in *SAS In-Database Products: Administrator's Guide* for information about how to install the support functions.

Note: It is not necessary to stop and restart the Teradata database when you install the SAS Embedded Process and QKB. However, if the SAS Embedded Process already exists and is running, you must stop it. It is also necessary to stop and restart the SAS Embedded Process for QKB updates. See "Controlling the SAS Embedded Process" in *SAS In-Database Products: Administrator's Guide* for more information.

You can manually verify that the tkindbsrv and sasqkb installations were successful by running these commands from the shell prompt:

```
psh "rpm -q -a" | grep tkindbsrv
psh "rpm -q -a" | grep sasqkb
```

If the installation was successful, tkindbsrv-9.41-**n** and sasqkb_ci-22.0-**ttttttttttttt** are displayed. **n** is a number that indicates the latest version of the tkindbsrv package file. If this is the initial installation, **n** has a value of 1. Each time you reinstall or upgrade, **n** is incremented by 1. **ttttttttttttt** is the timestamp assigned to the sasqkb package file by the qkb_pack script.

The SAS Data Quality Accelerator for Teradata binaries are installed as part of the tkindbsrv installation process. You do not need to take any further steps to install them.

Creating the Data Quality Stored Procedures

The data quality stored procedures are created in the Teradata database by running the dq_install.sh script. The dq_install.sh script needs to be run from a system with a UNIX shell and the Teradata BTEQ utility available.

The dq_install.sh script requires modification before it can be run. The Teradata administrator must edit the shell script to specify the site-specific Teradata server name and DBC user logon credentials for the DBC_PASS=, DBC_SRVR=, and DBC_USER= variables. Running dq_install.sh puts the data quality stored procedures into the SAS_SYSFNLIB database and enables the accelerator functionality.

Here is the syntax for executing dq_install.sh:

```
./dq_install.sh <-l log-path>
```

log-path
specifies an alternative name and location for the dq_install.sh log. When this parameter is omitted, the script creates a file named dq_install.log in the current directory.

Granting Users Authorization to the Data Quality Stored Procedures

The dq_grant.sh shell script is provided to enable the Teradata system administrator to grant users authorization to the data quality stored procedures. Before running the dq_grant.sh script, the Teradata administrator must edit it to specify the site-specific Teradata server name and DBC user logon credentials for the DBC_SRVR=, DBC_USER=, and DBC_PASS= variables. The user name specified in DBC_USER= and DBC_PASS= must have grant authority in the database.

Here is the syntax for executing dq_grant.sh:

```
./dq_grant.sh <-l log-path> user-name
```

log-path
specifies an alternative name and location for the dq_grant.sh log. When this parameter is omitted, the script creates a file named dq_grant.log in the current directory.

user-name
is the user name to which permission is being granted. The target user account must already exist in the Teradata database.

The authorizations granted by dq_grant.sh augment existing authorizations that the target user account already has in the Teradata database.

After you have installed the tkindsrv and sasqkb package files and run the dq_install.sh and dq_grant.sh scripts, the installation of the SAS Data Quality Accelerator for Teradata is complete.

Verifying the Accelerator Installation

Here is a simple BTEQ program that can be used to verify that the SAS Data Quality Accelerator for Teradata is operational. The example assumes that the SAS Data Quality Accelerator for Teradata is using the QKB for Contact Information and that the "English, United States" QKB locale is installed. Before running the example, substitute a real value for the *output_table* variable throughout the program. Note also that the *output_table* parameter for a SAS Data Quality Accelerator stored procedure takes the name of a non-existent table in a database to which you have Write access.

The CREATE VOLATILE TABLE statement creates a temporary input table named Dqacceltest that lasts for the duration of the SQL session. The DROP TABLE statement deletes the output table that was created by the DQ_GENDER stored procedure.

```
create volatile table dqacceltest (id integer, name varchar(64))
    unique primary index(id)
    on commit preserve rows;

insert into dqacceltest (id, name) values (1, 'John Smith');
```

```
insert into dqacceltest (id, name) values (2, 'Mary Jones');

call sas_sysfnlib.dq_gender_loc('ENUSA', 'Name', 'dqacceltest', 'id', 'name',
'output_table');

select gender from output_table;

drop table output_table;
```

If the request was successful, the SELECT statement produces an output table that contains this:

```
Gender
------
M
F
```

If the "English, United States" locale is not installed, try the following request as an alternative. Specify the name of an installed locale in the *locale* parameter.

```
call sas_sysfnlib.dq_set_locale('locale');
```

This test does not produce a direct output. However, the command should succeed if the installation is correct.

Troubleshooting the Accelerator Installation

Q. I ran the sample code and an output table was not created in my user schema. What now?

A. The stored procedures can fail if one or more of the following are true:

- The request specifies an output location to which the user does not have Write permission. Verify that you have access to the database that is specified in the *output_table* parameter.
- The data quality stored procedures are not installed correctly. Verify that the stored procedures are in the SAS_SYSFNLIB database by executing the following command:

  ```
  select TableName from dbc.tables where databasename='SAS_SYSFNLIB'
  and tablename like 'dq_%';
  ```

  The command should return a list similar to the following:

  ```
  TableName
  ------------------------------
  dq_standardize_loc
  dq_format
  dq_set_qkb
  dqi_invoke_table
  dq_standardize
  dqi_replace_tags
  dq_identify
  dq_extract
  dq_debug
  dq_gender_loc
  dqi_tbl_basename
  dq_parse_loc
  ```

If the procedures are absent, run the dq_install.sh script again, making sure you are logged in as Teradata system administrator.

- Permission to the data quality stored procedures is not granted correctly. Verify that the target user name submitted to the dq_grant.sh script is a valid user account in the Teradata database. Verify that the database server and granter information in the dq_grant.sh shell script is correct.
- The QKB is not in the correct location. Look for subdirectories similar to the following in the **/opt/qkb/default** directory on the Teradata nodes: **chopinfo**, **grammar**, **locale**, **phonetx**, **regexlib**, **scheme**, and **vocab**.
- The database connection does not specify MODE=TERADATA. When you submit the data quality stored procedures in SAS PROC SQL using explicit pass-through, the database connection is made in ANSI mode by default. You must specify the MODE= option to switch to Teradata mode.See "Using Data Quality Stored Procedures in Base SAS" on page 27 for an example of how this option is specified. Consult appropriate documentation for how to set Teradata mode in other client programs.

Removing the Procedures from the Database

The accelerator provides the dq_uninstall.sh shell script for removing the data quality stored procedures from the Teradata database.

The dq_uninstall.sh shell script requires modification before it can be run. The Teradata administrator must edit the shell script to specify the site-specific Teradata server name and DBC user logon credentials for the DBC_PASS=, DBC_SRVR=, and DBC_USER= variables.

Here is the syntax for executing dq_uninstall.sh:

```
./dq_uninstall.sh <-l log-path>
```

log-path
specifies an alternative name and location for the dq_uninstall.sh log. When this parameter is omitted, the script creates a file named dq_uninstall.log in the current directory.

Running dq_uninstall.sh disables the SAS Data Quality Accelerator for Teradata functionality and removes the data quality stored procedures from the database. The dq_uninstall.sh script does not remove the QKB or the SAS Embedded Process from the Teradata nodes. Follow whatever procedure is appropriate at your site for removing the QKB. See *SAS In-Database Products: Administrator's Guide* for information about how to stop the SAS Embedded Process and remove SAS in-database components from the Teradata database. The dq_grant.sh script also does not remove permissions that were granted by dq_grant.sh. You need to remove the permissions in accordance with the procedures used at your site.

Chapter 3

Reference

Invoking the Stored Procedures

The data quality stored procedures can be invoked with the CALL keyword from any product that supports the Teradata SQL dialect. For example, they might be run from within the Teradata command-line interface, BTEQ, or provided to the database via an ODBC connection in any language that supports the Teradata SQL dialect.

Session-Based and Non-Session Based Variants

You must choose a locale setting in order to run a data quality stored procedure. The stored procedures offer a session-based variant and a non-session-based variant. The session variant means that a locale setting is established for the entire SQL session by calling the DQ_SET_LOCALE() procedure before calling the data quality stored procedure. For more information, see "DQ_SET_LOCALE()" on page 22. The non-session-based variant sets the locale for the duration of the stored procedure's execution only.

The session and non-session names of each stored procedure are listed in Table 3.1 on page 16. The non-session variants have the suffix _LOC attached to the stored procedure name and take an additional parameter, *locale*, which is specified first.

Table 3.1 *Session and Non-Session Names of Data Quality Procedures*

Session Name	Non-Session Name	Description
DQ_EXTRACT()	DQ_EXTRACT_LOC()	Extracts specific entities or attributes from a text string.
DQ_GENDER()	DQ_GENDER_LOC()	Determines the gender of names.
DQ_IDENTIFY()	DQ_IDENTIFY_LOC()	Determines the type of data that is represented by a text string.
DQ_LOWERCASE()	DQ_LOWERCASE_LOC()	Lowercases text.
DQ_MATCH()	DQ_MATCH_LOC()	Generates a matchcode for a text string.
DQ_PARSE()	DQ_PARSE_LOC()	Segments a string into semantically atomic tokens.
DQ_PATTERN()	DQ_PATTERN_LOC()	Returns a simple representation of a text string's character pattern.
DQ_PROPERCASE()	DQ_PROPERCASE_LOC()	Applies uppercase and lowercase lettering using context-sensitive rules.
DQ_STANDARDIZE()	DQ_STANDARDIZE_LOC()	Generates a preferred standard representation of a string.
DQ_UPPERCASE()	DQ_UPPERCASE_LOC()	Uppercases text.
DQ_SET_LOCALE()	Not applicable	Sets the locale for data quality operations in the SQL session.

How the Stored Procedures Work

The stored procedures operate on all of the rows in a specified column of a specified input table. They can return output in an output table or as a dynamic result set via a cursor if you specify NULL as the output table name. A *cursor* is a control structure that enables traversal over the records in a result set. It facilitates subsequent processing, such as retrieval, addition, and removal of database records, by enabling the rows in the result set to be processed sequentially.

The stored procedures have the same general syntax. They all take five parameters, except for DQ_MATCH, which takes a sixth. Four of the parameters are logistical: they specify the input table, input column, optional primary key, and a name for the output table. The fifth parameter, *definition*, is specific to the operation. The *definition* parameter specifies the name of a DataFlux Quality Knowledge Base definition that provides instructions for the operation. The definitions that are available for an operation are locale-specific. For more information about QKB definitions, see "Locating QKB Definition Names" on page 18.

The syntax for the session-based and non-session variants is slightly different. For more information about the stored procedure syntax, see "Session Variant Syntax" on page 19 and "Non-Session Variant Syntax" on page 21. If you are using the session-based variant, also see "DQ_SET_LOCALE()" on page 22.

Stored Procedure Output

The output tables returned by the data quality stored procedures contain a minimum of three or four columns.

- Column _INPUT_ shows what the data in the column specified in the *data_column* parameter of the stored procedure call looked like before the data quality stored procedure was run.
- Column _ERR_ contains a blank value or an error message. A blank value indicates that the stored procedure ran successfully. Error messages are returned for input errors and errors caused by internal processes.

  Examples of error messages:

  ```
  Invalid definition name: 'definition-name'

  Invalid locale specified: 'locale-name' (error code)

  Locale 'locale-name' could not be loaded (error code)

  Out of memory

  Internal error (error code)
  ```

- Column *Result* holds the result of applying the data quality operation to each input found in the column specified in the *data_column* parameter of the stored procedure call.

  The DQ_GENDER, DQ_IDENTIFY, DQ_MATCH, DQ_LOWERCASE, DQ_PATTERN, DQ_PROPERCASE, DQ_STANDARDIZE, and DQ_UPPERCASE stored procedures return a single result column. This column is named after the operation (for example, Gender, Identified, Matchcode, Lowercase, Pattern, Propercase, Standardized, and Uppercase).

  The DQ_EXTRACT and DQ_PARSE stored procedures return a result column for each token present in the QKB definition named in the *definition* parameter of the stored procedure call. The columns are named after the tokens.

- Column _PK_ is optional. It contains the values from the data column that was specified in the *pk_column* parameter of the stored procedure call. If the value of the *pk_column* parameter is NULL in the stored procedure call, then column _PK_ is not created in the output table. When column _PK_ is created, it is created with the PRIMARY KEY constraint, permitting joins with the source table.

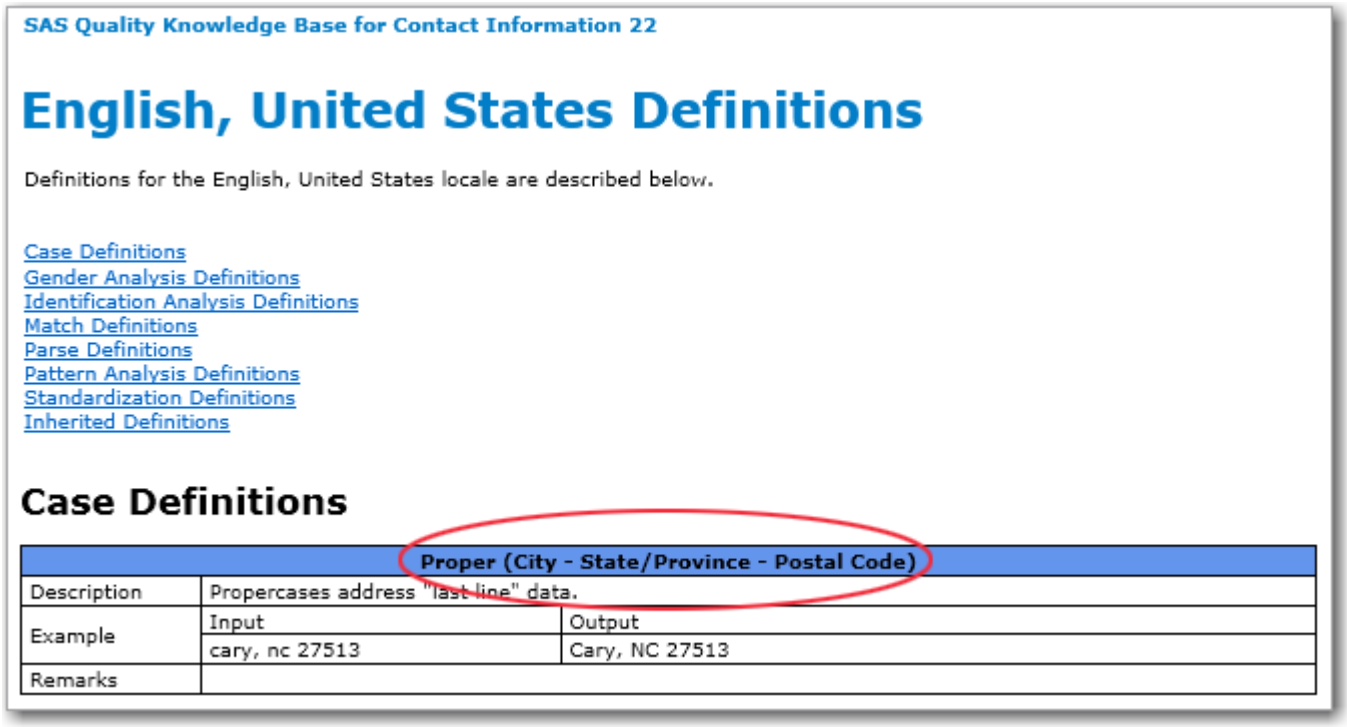
Column _PK_ has the same data type as the column that was specified in the *pk-column* parameter in the stored procedure call. All other output columns have data type VARCHAR(1024), except for the result column in the output table created by DQ_GENDER. The data type for the Gender column is VARCHAR(1).

For an example of the output tables, see "Using Data Quality Stored Procedures in Base SAS" on page 27. This topic contains code that creates a small source table named MyDB.Employees, which contains uncleansed data. Then, it executes data quality stored procedures to cleanse the data and creates a report with the cleansed data. It includes screen captures of the source table, stored procedure output tables, and the cleansed data.

Locating QKB Definition Names

In the initial release of the software, SAS Data Quality Accelerator for Teradata does not provide a programmatic way to retrieve definition names from the QKB. To obtain definition names, you can open the QKB Help in the original QKB installation on Windows or UNIX. Or you can use the Data Management Studio Customize program to view its content. For users that do not have access to the QKB, we recommend that the person installing the QKB provide a list of the installed locales and definitions.

The following is an example of a QKB definition as it appears in the QKB Help. It is a Case definition for the “English, United States” locale.



SAS Quality Knowledge Base for Contact Information 22

English, United States Definitions

Definitions for the English, United States locale are described below.

Case Definitions
Gender Analysis Definitions
Identification Analysis Definitions
Match Definitions
Parse Definitions
Pattern Analysis Definitions
Standardization Definitions
Inherited Definitions

Case Definitions

Proper (City - State/Province - Postal Code)		
Description	Propercases address "last line" data.	
Example	Input	Output
	cary, nc 27513	Cary, NC 27513
Remarks		

The *definition* value that is needed by the stored procedures is shown in the title bar of the table in the documentation. The name must be specified as shown in the table, including all parentheses, dashes, and slashes. Not all names have special characters in them.

Note that the Help includes a description of the definition and a sample input and output string to illustrate the types of changes that the definition will make.

SAS Data Quality Accelerator for Teradata supports the same QKB locales and definitions as SAS Data Quality Server for comparable data quality functions. If you have access to SAS Data Quality Server software, you can use it to generate a list of available definitions for specific data quality operations and locales. For more information, see *SAS Data Quality Server: Reference*.

Session Variant Syntax

The session variant syntax executes the specified data quality operation in the locale established by the DQ_SET_LOCALE stored procedure.

Syntax

DQ_OPERATION ('*definition*', <'*sensitivity*',> '*input-table*', '*pk-column*', '*data-column*', '*output-table*');

Parameters

DQ_OPERATION
specifies the session name of a data quality stored procedure. Preface the stored procedure name with the name of the database in which it is located. The default database name is SAS_SYSFNLIB. See "Session-Based and Non-Session Based Variants" on page 15 for a list of the stored procedure names.

'*definition*'
specifies the name of a DataFlux Quality Knowledge Base definition. The definition must be valid in the session's current locale. The definition name must be specified as shown in the QKB help.

'*sensitivity*'
used by the DQ_MATCH stored procedure only, for which it is required, specifies an integer that represents the degree of similarity that should be applied to consider something a match. Valid values range from 50 to 95. The recommended sensitivity setting is 85. A higher sensitivity value requires greater similarity between input values for a match to be obtained. In general, higher sensitivity leads to fewer matches than lower sensitivity.

'*input-table*'
specifies the name of an existing database table containing data to be processed. Include the database name.

'*pk-column*'
specifies the name of the column in *input-table* that contains the primary key. If a NULL value is passed in this parameter, then no primary key will be transferred to the output table.

'*data-column*'
specifies the name of the column in *input-table* that contains the data to be processed. This column must be a VARCHAR type.

Note: Any column name that is a reserved word in Teradata needs to be quoted when passed into a data quality stored procedure. For example:

```
call sas_sysfnlib.dq_parse('Date (MDY)', 'mydb.mytable', '_PK_',
'"date"', 'mydb.parsed')
```

'*output-table*'
specifies the name of the database table where result rows are written. Include the database name. If the table does not exist, it will be created. We recommend that an existing table not be used. If the table exists, the results are appended to the table, or fail with an error, if the existing table contains a primary key. When the *output-table*

parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table.

Details

The strings passed to the stored procedures need to be encoded in the ISO-8859-1 (Latin-1) encoding or some compatible encoding. This means that any table names or column names containing non-Western European characters are not supported. This restriction does not apply to the actual data in the columns. Unicode is supported within the columns, just not in the table metadata.

All parameters, except *sensitivity*, accept a string of up to 256 characters.

Example 1: Example of a One-Column Operation

This example uses DQ_STANDARDIZE to standardize the values of a column named Address in a table named Places. The output is stored in a table named Standardized.

```
call sas_sysfnlib.dq_standardize('Address', 'mydb.Places', 'Id',
'Address', 'mydb.Standardized');
   select * from mydb.Standardized;
```

This is the content of the Places table:

```
Id    Address
1    6512 SIX Forks Road 404B
2    1503 del norte drive
```

This is the content of the Standardized table:

```
_PK_    Result
1    6512 Six Forks Rd 404B
2    1503 Del Norte Dr
```

Example 2: Example of a Two- or More- Column Operation

This example uses the DQ_PARSE procedure to parse the values of a column named Contact from a table named People. The output is stored in a table named Parsed.

```
call sas_sysfnlib.dq_parse('Name', 'mydb.People', 'Id', 'Contact', 'mydb.Parsed');
   select * from mydb.Parsed;
```

This is the content of the People table:

```
Id    Contact
1     JAMES K WRIGHT
2     Mr. Willie White
```

This is the content of the Parsed table. The data in column Contact is split into columns Prefix, Given Name, Middle Name, and Family Name.

```
_PK_   Prefix   Given Name   Middle Name   Family Name
 1              JAMES        K             WRIGHT
 2     Mr.      Willie                     White
```

Non-Session Variant Syntax

The non-session variant syntax executes the specified data quality operation in the specified locale. If a session locale has already been set, this type of call overrides it for the duration of the call only, after which the session returns to the pre-set locale.

Syntax

DQ_OPERATION_LOC('*locale*', '*definition*', <'*sensitivity*',> '*input-table*', '*pk-column*', '*data-column*', '*output-table*');

Parameters

Except for *DQ_OPERATION_LOC* and *locale*, the parameters are identical to the ones described for "Session Variant Syntax" on page 19.

DQ_OPERATION_LOC
specifies the non-session name of a data quality stored procedure. Preface the stored procedure name with the name of the database in which it is located. The default database name is SAS_SYSFNLIB. See "Session-Based and Non-Session Based Variants" on page 15 for a list of the names.

'***locale***'
specifies an uppercase string specifying the 5-letter ISO code name of the QKB locale to use for the session (for example, ENUSA).

The QKB locales that are available to the data quality stored procedures were set when the SAS Data Quality Accelerator for Teradata software was installed. Contact the administrator of the SAS Data Quality Accelerator for Teradata software at your installation to get the ISO code name (or ISO code names) that are available for your use.

Examples

The following example shows how the non-session variant would be used in the DQ_STANDARIZE example from "Example 1: Example of a One-Column Operation" on page 20.

```
call sas_sysfnlib.dq_standardize_loc('ENUSA', 'Address', 'mydb.Places',
'Id', 'Address', 'mydb.Standardized');
```

This example shows how the non-session variant would be used in the DQ_PARSE example from "Example 2: Example of a Two- or More- Column Operation" on page 20.

```
call sas_sysfnlib.dq_parse_loc('ENUSA', 'Name', 'mydb.People', 'Id',
'Contact', 'mydb.Parsed');
```

Dictionary

DQ_SET_LOCALE()

Establishes the locale for an in-database data quality stored procedure session.

Syntax

DQ_SET_LOCALE('*locale*');

Parameters

'*locale*'
is an uppercase string specifying the 5-letter ISO code name of the QKB locale to use for the session (for example, ENUSA).

Details

DQ_SET_LOCALE is a session-based stored procedure that establishes the locale for the entire SQL session, or until it is called again with a different locale. Follow this call with a session-based data quality stored procedure call.

The QKB locales that are available were set when the DataFlux Quality Knowledge Base was installed. Contact the administrator of the SAS Data Quality Accelerator for Teradata software at your installation to get the ISO code name (or ISO code names) that are available for your use.

Non-session data quality stored procedure calls can still be made using other locales without affecting the value set by this call. Only one session-based locale can be set at a time.

Example

```
call sas_sysfnlib.dq_set_locale('ENUSA');
```

Chapter 4

Usage and Examples

Overview of Using SAS Data Quality Accelerator

The traditional data quality process involves the following steps:

1. Extract uncleansed source data from the DBMS to the SAS Data Quality Server.
2. Standardize the data.
3. Publish the cleansed data to the DBMS.

With SAS Data Quality Accelerator for Teradata, the extraction step is removed, and the process consists only of connecting to the DBMS and calling stored procedures to standardize the uncleansed source data.

Example of Data Quality Operations

The following are examples of the data quality operations that can be performed with the data quality stored procedures.

Casing

Casing applies context-sensitive case rules to text. It operates on character content, such as names, organizations, and addresses. Casing is applied with the DQ_LOWERCASE(), DQ_PROPERCASE(), and DQ_UPPERCASE() stored procedures.

Table 4.1 *Examples of Case Stored Procedure Inputs and Outputs*

Input	Procedure	Output
SAS INSTITUTE	DQ_LOWERCASE	sas institute
	DQ_UPPERCASE	SAS INSTITUTE
	DQ_PROPERCASE	SAS Institute

Extraction

Extraction returns one or more extracted text values, or tokens, as output. Extraction is performed with the DQ_EXTRACT() stored procedure.

Table 4.2 *Example of Extraction*

Input	Output	
Blue men’s long-sleeved button-down collar denim shirt	Color:	Blue
	Material:	Denim
	Item:	Shirt

Gender Analysis

Gender analysis 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 stored procedure returns a value that indicates that the gender is unknown. The exact return value is determined by the specified gender analysis definition and locale. Gender analysis is performed with the DQ_GENDER() stored procedure.

Table 4.3 *Example of Gender Analysis*

Input	Output
Jane Smith	F

Input	Output
Sam Adams	M
P. Jones	U

Identification Analysis

Identification analysis returns a value that indicates the category of the content in an input character string. The available categories and return values depend on your choice of identification definition and locale. Identification analysis is performed with the DQ_IDENTIFY() stored procedure.

Table 4.4 *Example of Identification Analysis*

Input	Output
John Smith	NAME
SAS Institute	ORGANIZATION

Matching

Matching analyzes the input text string and generates a matchcode for the string. The matchcode represents a condensed version of the character value. Similar strings get identical matchcodes. You can specify a sensitivity value that indicates the degree of similarity that should be applied to consider something a match. For higher sensitivities, two values must be very similar to produce the same matchcode. At lower sensitivities, two values might produce the same matchcode despite considerable dissimilarities. Records can be clustered by sorting by matchcodes. Fuzzy lookups can be performed via matchcode searches. Matching is performed with the DQ_MATCH() stored procedure.

Data quality stored procedures currently do not perform clustering. You need to retrieve matchcodes and matchcode index keys from the DBMS and use a product such as SAS Data Quality Server to cluster data to determine possible matches and to determine which record from each cluster will be the survivor. After the survivors have been determined, issue SQL statements to select the cleansed data of index keys of relevant survivors and write them to the DBMS.

Table 4.5 *Example of Matching*

Input	Output
Gidley, Scott A	XYZ$$$
Scotty Gidleigh	XYZ$$$
Mr Scott Gidlee Jr.	XYZ$$$

Input	Output
Mr Robert J Brauer	ABC$$$
Bob Brauer	ABC$$$

Parsing

Parsing segments a string into semantically atomic tokens. Parsing is performed with the DQ_PARSE() stored procedure.

Table 4.6 *Example of Parsing*

Input	Output	
Mr. Roy G. Biv Jr	Prefix:	Mr.
	Given Name:	Roy
	Middle Name:	G.
	Family Name:	Biv
	Suffix:	Jr

Pattern Analysis

Pattern analysis returns a simple representation of a text string's character pattern, which can be used for pattern frequency analysis in profiling jobs. Pattern analysis identifies words or characters in the input data column as numeric, alphabetic, non-alphanumeric, or mixed. The choice of pattern analysis definition determines the nature of the analysis. Pattern analysis is performed with the DQ_PATTERN() stored procedure.

Table 4.7 *Example of Pattern Analysis*

Input	Output
919-677-8000	999-999-999
NC	AA

Standardization

Standardization generates a preferred standard representation of a string. Standardization definitions are provided for character content such as dates, names, and postal codes. The available standardization definitions vary from one locale to the next. The return

values are provided in the appropriate case, and insignificant blank spaces and punctuation are removed. The order of the elements in the return values might differ from the order of the elements in the input character values. Standardization is performed with the DQ_STANDARDIZE() stored procedure.

Table 4.8 *Example of Standardization*

Input	Output
N car	NC
919.6778000	(919) 677–8000
Smith, Mister James	Mr. James Smith

Examples of Using Data Quality Stored Procedures to Clean Data

Overview of Examples

This section shows two ways that you can connect to a Teradata database and execute the data quality stored procedures. The first example uses the Base SAS SQL procedure to execute the data quality stored procedures. The second example uses the Teradata BTEQ utility to execute the data quality stored procedures.

The examples are provided to show how the data quality stored procedures can be used to clean data. In both examples, the sample code creates a small source table named MyDB.Employees, which contains uncleansed data. Then, it executes data quality stored procedures to cleanse the data and creates a report with the cleansed data. In the examples:

- The DQ_GENDER_LOC stored procedure determines the gender of the employees in the source table.
- The DQ_PARSE_LOC stored procedure splits the employee’s first, middle, and last names.
- The DQ_STANDARDIZE_LOC stored procedure standardizes the employees' addresses.

The non-session variants of the stored procedures are used, which means the stored procedures specify the QKB locale in each procedure call. The examples use the “English, United States” locale.

The Base SAS sample program contains extra code to print the stored procedure output tables.

Using Data Quality Stored Procedures in Base SAS

In a Base SAS session, you must use the SQL procedure to execute the data quality stored procedures. The stored procedures are executed through the SQL pass-through facility.

```
proc sql;
  /* Establish the connection to Teradata. */
  connect to teradata
    (server=teraserver
     user=myid
     password=mypwd
     database=mydb
     mode=teradata); 1

  /* Create an example source table. */
  execute (
    create table mydb.employees
      (empid      integer primary key not null,
       name       varchar(50),
       address    varchar(100),
       start_date varchar(9),
       birth_date varchar(9))
  ) by teradata;

  /* Populate the source table with some data. */
  execute (
    insert into mydb.employees values
        (203,
         'Ken Patrick Burke',
         '445 main street',
         '27JAN1994',
         '14JAN1950')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (485,
         'Stella Mae Santorini',
         '160-a NORTH 6TH ST # 3',
         '18DEC2005',
         '17JUL1980')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (1003,
         'Dale Michael Hayes',
         '1575 Jordon St, Suite 3 2nd floor',
         '01AUG2009',
         '09OCT1985')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (460,
         'Angela Sarah Koehlepp',
         'Number 704 Martin Luther King Blvd',
         '14FEB2005',
         '12MAY1978')
  ) by teradata;
```

```
  execute (
    insert into mydb.employees values
        (255,
         'Ross Maxwell Petersen',
         'POBOX 4 SMITH STREET',
         '23NOV1996',
         '12MAR1968')
  ) by teradata;

select * from connection to teradata (select * from mydb.employees);

  /* Determine gender of employees in the source table. */ 2
  execute (
    call sas_sysfnlib.dq_gender_loc(
      'ENUSA',
      'Name',
      'mydb.employees',
      'empid',
      'name',
      'mydb.employees_gend')
  ) by teradata;

/* print output table */
select * from connection to teradata (select * from mydb.employees_gend);

  /* Run parse to split the first, middle, and last name. */ 3
  execute (
    call sas_sysfnlib.dq_parse_loc(
      'ENUSA',
      'Name',
      'mydb.employees',
      'empid',
      'name',
      'mydb.employees_pars')
  ) by teradata;

/* print output table */
select * from connection to teradata (select * from mydb.employees_pars);

  /* Standardize the employees' addresses. */ 4
  execute (
    call sas_sysfnlib.dq_standardize_loc(
      'ENUSA',
      'Address',
      'mydb.employees',
      'empid',
      'address',
      'mydb.employees_stdz')
  ) by teradata;

/* print output table */
select * from connection to teradata (select * from mydb.employees_stdz);

  /* Aggregate the processed data into a report. */ 5
  select * from connection to teradata (
    select e.empid,
```

```
            p."Given Name",
            p."Middle Name",
            p."Family Name",
            e.birth_date,
            e.start_date,
            g.Gender,
            s.Standardized
       from mydb.employees      e,
            mydb.employees_gend g,
            mydb.employees_pars p,
            mydb.employees_stdz s
      where e.empid=g._PK_ and
            e.empid=p._PK_ and
            e.empid=s._PK_
  );

  /* Clean up. */
  execute (drop table employees)      by teradata;
  execute (drop table employees_gend) by teradata;
  execute (drop table employees_pars) by teradata;
  execute (drop table employees_stdz) by teradata;

quit;
```

1 Note the use of the LIBNAME option MODE=TERADATA in the database connection statement. The data quality stored procedures require client programs to connect to the Teradata database with a database connection of session-mode Teradata. If this option is not specified, PROC SQL will connect in ANSI mode.

2 The DQ_GENDER_LOC stored procedure call specifies to apply the QKB definition "Name" to data column Name. It specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Gend.

3 The DQ_PARSE_LOC stored procedure call specifies to apply the QKB definition "Name" to data column Name as well. Although this definition has the same name as the definition specified in DQ_GENDER_LOC, the content of the definitions is not the same. The definitions contain different instructions. The stored procedure call specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Pars.

4 The DQ_STANDARDIZE_LOC stored procedure specifies to apply the QKB definition "Address" to data column Address. It specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Stdz.

5 In the final step, a pass-through query is submitted to the Teradata database that specifies which columns from the source and stored procedure output tables to display and performs an equijoin on the primary key EmpID.

The following display shows the source table:

The SAS System

empid	name	address	start_date	birth_date
460	Angela Sarah Koehlepp	Number 704 Martin Luther King Blvd	14FEB2005	12MAY1978
203	Ken Patrick Burke	445 main street	27JAN1994	14JAN1950
255	Ross Maxwell Petersen	POBOX 4 SMITH STREET	23NOV1996	12MAR1968
485	Stella Mae Santorini	160-a NORTH 6TH ST # 3	18DEC2005	17JUL1980
1003	Dale Michael Hayes	1575 Jordon St, Suite 3 2nd floor	01AUG2009	09OCT1985

The following display shows the MyDB.Employees_Gend table:

The SAS System

INPUT	_ERR_	Gender	_PK_
Angela Sarah Koehlepp		F	460
Ken Patrick Burke		M	203
Ross Maxwell Petersen		M	255
Stella Mae Santorini		F	485
Dale Michael Hayes		M	1003

The following display shows the MyDB.Employees_Pars table:

The SAS System

ERR	_INPUT_	_PK_	Prefix	Given Name	Middle Name	Family Name	Suffix	Title/Additional Info
	Angela Sarah Koehlepp	460		Angela	Sarah	Koehlepp		
	Ken Patrick Burke	203		Ken	Patrick	Burke		
	Ross Maxwell Petersen	255		Ross	Maxwell	Petersen		
	Stella Mae Santorini	485		Stella	Mae	Santorini		
	Dale Michael Hayes	1003		Dale	Michael	Hayes		

The following display shows the MyDB.Employees_Stdz table:

The SAS System

INPUT	_ERR_	Standardized	_PK_
Number 704 Martin Luther King Blvd		Martin Luther King Blvd Num 704	460
445 main street		445 Main St	203
POBOX 4 SMITH STREET		Smith St PO Box 4	255
160-a NORTH 6TH ST # 3		160-A N 6th St # 3	485
1575 Jordon St, Suite 3 2nd floor		1575 Jordon St Suite 3 Fl 2nd	1003

The following display shows the content of the report after the data was cleaned with the data quality stored procedures:

The SAS System

empid	Given Name	Middle Name	Family Name	birth_date	start_date	Gender	Standardized
460	Angela	Sarah	Koehlepp	12MAY1978	14FEB2005	F	Martin Luther King Blvd Num 704
1003	Dale	Michael	Hayes	09OCT1985	01AUG2009	M	1575 Jordon St Suite 3 Fl 2nd
255	Ross	Maxwell	Petersen	12MAR1968	23NOV1996	M	Smith St PO Box 4
203	Ken	Patrick	Burke	14JAN1950	27JAN1994	M	445 Main St
485	Stella	Mae	Santorini	17JUL1980	18DEC2005	F	160-A N 6th St # 3

Using Data Quality Stored Procedures in BTEQ

The following is BTEQ code that can be used to create and clean the content of the MyDB.Employees source table and create a report. BTEQ is a utility from the Teradata Utility Pack that allows users on a workstation to submit SQL statements to the Teradata database. This utility allows users to query, import, and export data from one or more Teradata RDBMS, and to produce reports. BTEQ stands for Basic Teradata Query. BTEQ connects to the Teradata database in Teradata mode by default.

```
.logon teraserver/myid,mypwd
.width 160

/* Create an example source table. */
create table mydb.employees
  (empid      integer primary key not null,
   name       varchar(50),
   address    varchar(100),
   start_date varchar(9),
   birth_date varchar(9));

/* Populate the source table with some data. */
insert into mydb.employees values
  (203,
   'Ken Patrick Burke',
   '445 main street',
   '27JAN1994',
   '14JAN1950');

insert into mydb.employees values
  (485,
   'Stella Mae Santorini',
   '160-a NORTH 6TH ST # 3',
   '18DEC2005',
   '17JUL1980');

insert into mydb.employees values
  (1003,
   'Dale Michael Hayes',
```

```
   '1575 Jordon St, Suite 3 2nd floor',
   '01AUG2009',
   '09OCT1985');

insert into mydb.employees values
  (460,
   'Angela Sarah Koehlepp',
   'Number 704 Martin Luther King Blvd',
   '14FEB2005',
   '12MAY1978');

insert into mydb.employees values
  (255,
   'Ross Maxwell Petersen',
   'POBOX 4 SMITH STREET',
   '23NOV1996',
   '12MAR1968');

/* Determine gender of employees in the source table. */
call sas_sysfnlib.dq_gender_loc(
  'ENUSA',
  'Name',
  'mydb.employees',
  'empid',
  'name',
  'mydb.employees_gend');

/* Run parse to split the first, middle, and last name. */
call sas_sysfnlib.dq_parse_loc(
  'ENUSA',
  'Name',
  'mydb.employees',
  'empid',
  'name',
  'mydb.employees_pars');

/* Standardize the employees' addresses. */
call sas_sysfnlib.dq_standardize_loc(
  'ENUSA',
  'Address',
  'mydb.employees',
  'empid',
  'address',
  'mydb.employees_stdz');

/* Aggregate the processed data into a report. */
select e.empid                                as "Employee ID",
       cast (p."Given Name"  as varchar(15)) as "Given Name",
       cast (p."Middle Name" as varchar(15)) as "Middle Name",
       cast (p."Family Name" as varchar(15)) as "Family Name",
       e.birth_date                           as "Birth Date",
       e.start_date                           as "Start Date",
       g.Gender                               as "Gender",
       cast (s.Standardized as varchar(30))  as "Address"
  from mydb.employees      e,
       mydb.employees_gend g,
```

```
       mydb.employees_pars p,
       mydb.employees_stdz s
where e.empid=g._PK_ and
       e.empid=p._PK_ and
       e.empid=s._PK_;

/* Clean up. */
drop table mydb.employees;
drop table mydb.employees_gend;
drop table mydb.employees_pars;
drop table mydb.employees_stdz;

.logoff
.quit
```

Index