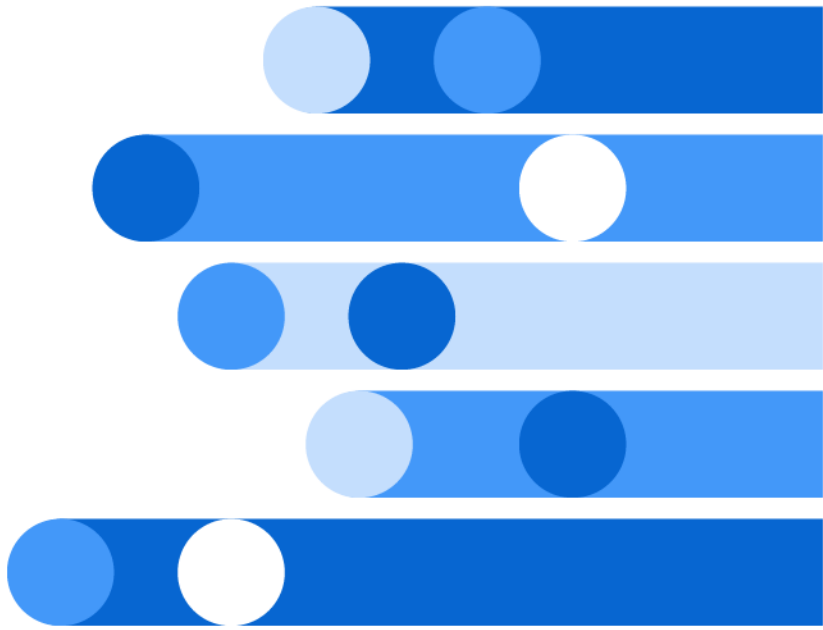
SAS® Quality Knowledge Base: Customization Reference



SAS® Documentation
April 3, 2025

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2025. *SAS® Quality Knowledge Base: Customization Reference*. Cary, NC: SAS Institute Inc.

SAS® Quality Knowledge Base: Customization Reference

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

April 2025

v_001-P1:qkbcstmref

Contents

1

About the Customization Reference

Overview

Overview

This SAS Quality Knowledge Base (QKB): Customization Reference is the same documentation found in the DataFlux Data Management Studio: User's Guide, in earlier releases. This reference is available so that users can quickly locate the information no matter what version of the QKB you use.

SAS QKB users can customize the QKB for your needs.

Note: Contents of the SAS Quality Knowledge Base: Customization Reference originated from the DataFlux Data Management Studio: User's Guide. Go to the DataFlux Data Management Server Learn & Support page > **Documentation** to view the complete user's guide.

What is a QKB?

The SAS QKB is a collection of files that store data and logic that define data management operations such as parsing, standardization, and matching. SAS software products refer to the QKB when performing data management operations, also referred to as data cleansing, on your data.

A QKB supports locales organized by language and country for example, English, United States; English, Canada; and French, Canada.

QKB Definitions

Data and logic in a QKB are organized into a set of objects called definitions. In DataFlux Data Management Studio, these definitions are listed on the **Quality Knowledge Base** tab of the QKB interface.

Each definition defines a single, context-sensitive data management operation. For example, a definition in a QKB might contain data and logic used to parse phone numbers. Another definition might contain data and logic used to determine the gender of individuals by analyzing their names.

When you use SAS data management products to process your data, you specify which definitions the software should invoke. For example, if you are standardizing company names with the **Standardization** node in a data job, you can specify that the node should use the Organization definition when processing data in the **Company** field in your table.

A QKB contains definitions developed for use with common types of data such as names, addresses, and phone numbers. You can use definitions in a QKB as delivered, or you can use DataFlux Data Management Studio to customize the QKB. You can modify definitions or create new definitions for use with your own business data. Since a QKB is used by **DataFlux Data Management Studio**, **SAS Data Quality Server**, and the SAS Data Quality Accelerators, QKB customizations are automatically available to your entire enterprise.

To display the help for QKBs installed at your site, see Accessing Your QKB Documentation on page 8. There are periodic updates to a QKB with new definitions and enhancements to existing definitions. To download updates to a QKB, visit the QKB downloads page.

2

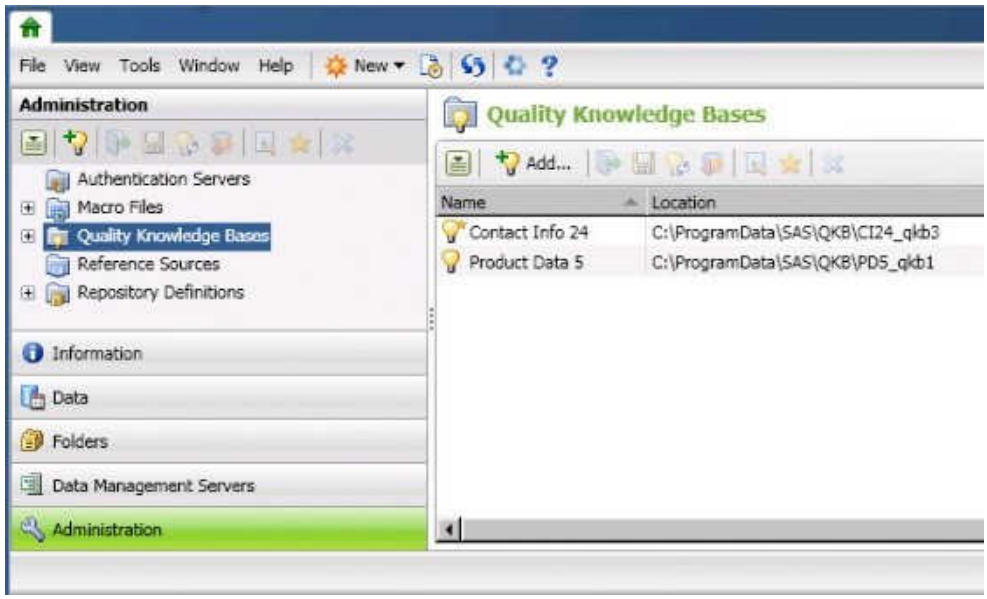
Accessing QKBs and Documentation

Accessing Your QKB

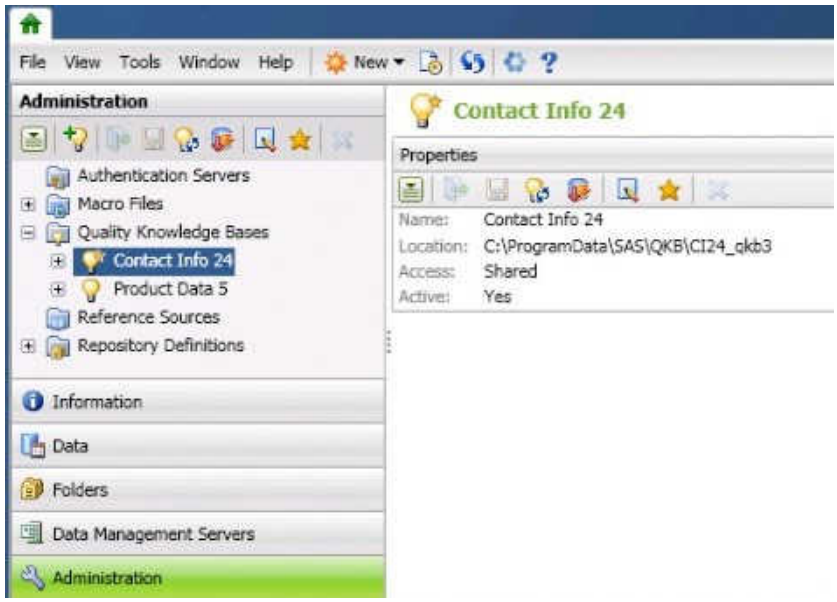
Accessing QKBs from the Administration Riser

If a QKB has been registered on page 148, it can be accessed from the **Administration** riser in the main window of DataFlux Data Management Studio.

Click the **Administration** riser. Select the **Quality Knowledge Bases** folder to see a list of registered QKBs on the right, as shown in the next display.



In the right panel, double-click the QKB that you want to access, such as *Contact Info 24* in the previous display. The top-level folder for that QKB appears in the tree on the left. Expand the folder to display a tree of the locales that are supported locales for the QKB. The **Global** item is the root of the tree, as shown in the next display.



On the left, under **Global**, locales are organized in a hierarchy according to their language. To view the locales associated with a language, expand the tree under that language item. For example, expand **English**. The English-language locales in the QKB will appear, as shown in the next display.

In the example above, **English (Canada)** has a **+** next to the flag. This indicates that there is another language locale available for Canada, in this case **French (Canada)**, as shown in the next display.

If a QKB icon in the tree has a black lock overlay, it means that the QKB is locked and cannot be opened for editing.

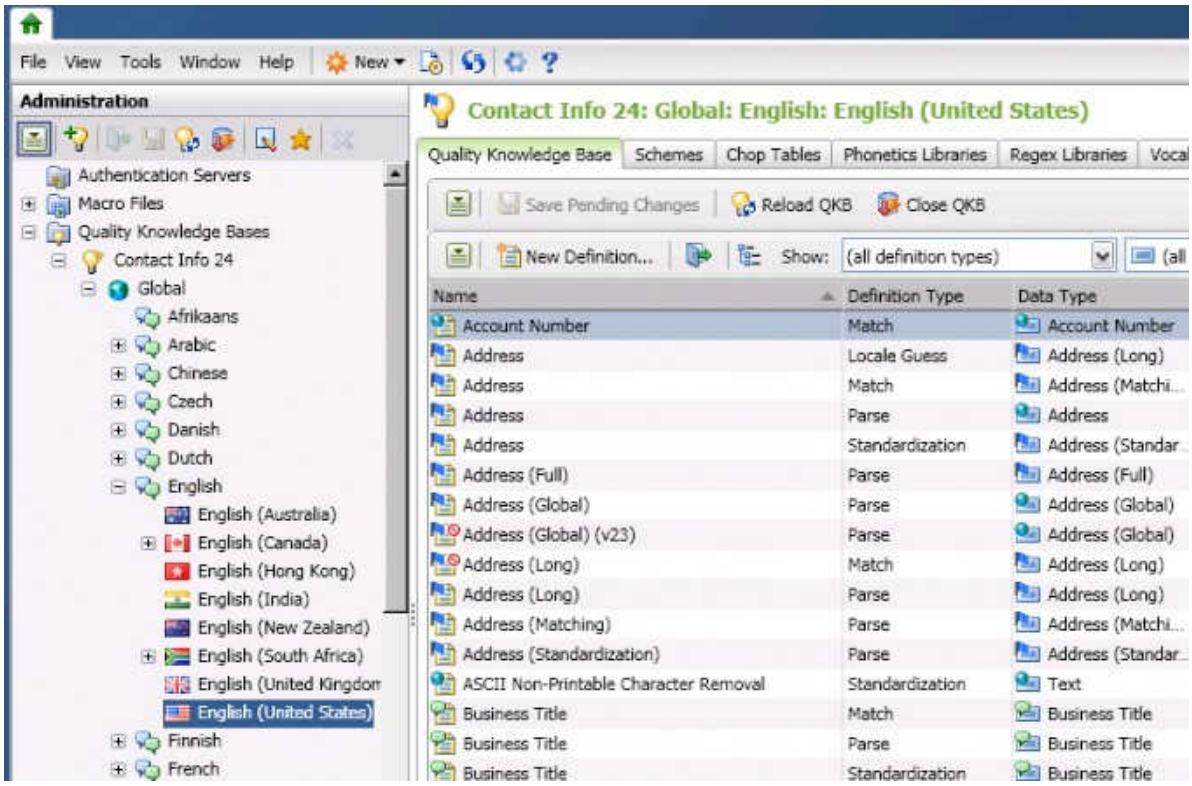
A QKB is locked if it is in a read-only directory or if it is already opened for editing by another user. If you try to open a locked QKB, you see a message indicating that you cannot open the QKB for editing, but you can open the QKB for browsing.

If the QKB is not open, right-click the locale that you want to open, such as **English (United States)** in the previous display. A menu is displayed. The following table describes the options in the menu.

Name	Description
Open QKB	Select to view and edit the definitions and other items in a selected QKB.
Save QKB	Select to save any changes to the current QKB.

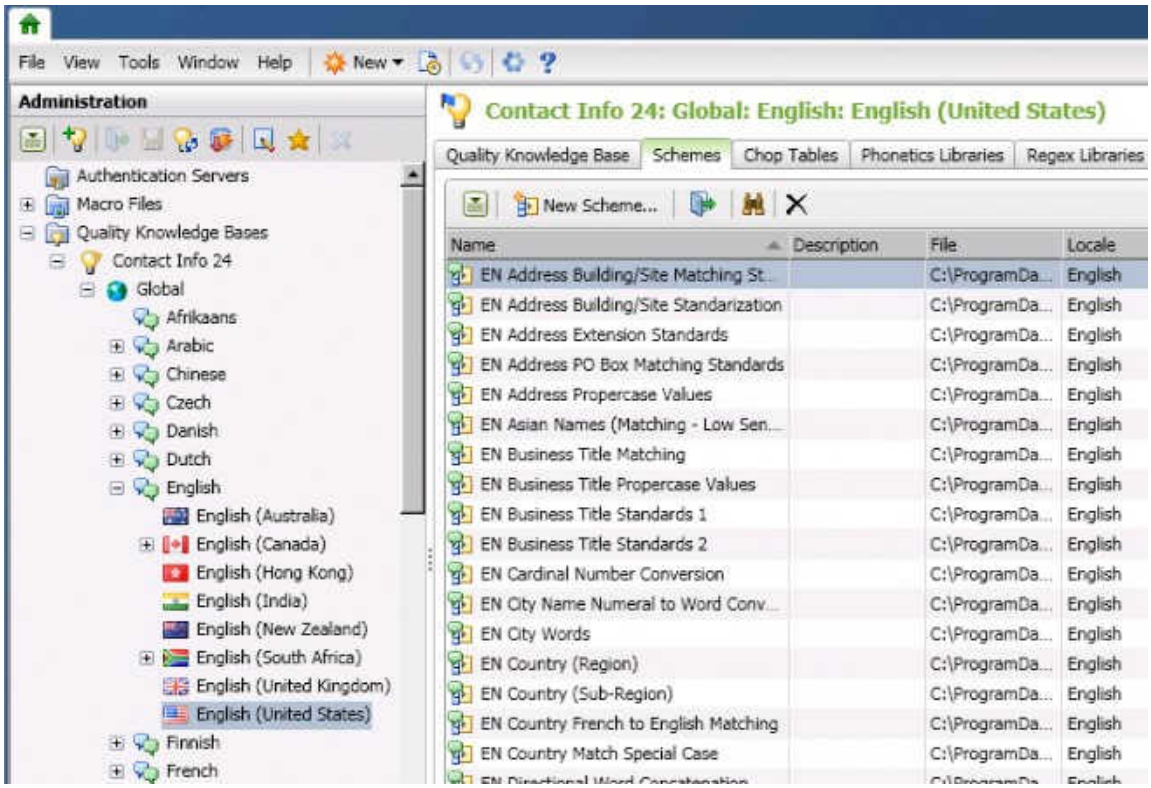
Name	Description
Close QKB	Select to close the QKB that you are viewing. You can have only one QKB open at a time.
Edit QKB Properties	Select to edit the registration record for a QKB. See Registering (Adding) a QKB on page 148.
Make QKB Active	Select to make a QKB active in DataFlux Data Management Studio. There can be only one active QKB at a time. See Making a QKB Active on page 147.
Export Definitions	Select to export definitions from the current QKB to a package file. This option is available for the active QKB only. See Importing and Exporting Definitions on page 150.
Import Definitions	Select to import definitions that were exported from a QKB. This option is available for the active QKB only. See Importing and Exporting Definitions on page 150.
Remove QKB	Select to unregister the selected QKB from DataFlux Data Management Studio. The QKB files are not deleted from the file system. See Removing (Unregistering) a QKB on page 149.

Select **Open QKB** to view the definitions and other items in the selected QKB. When loading is complete, the QKB definitions contained in the selected locale will be listed on the **Quality Knowledge Base** tab, as shown in the next display.



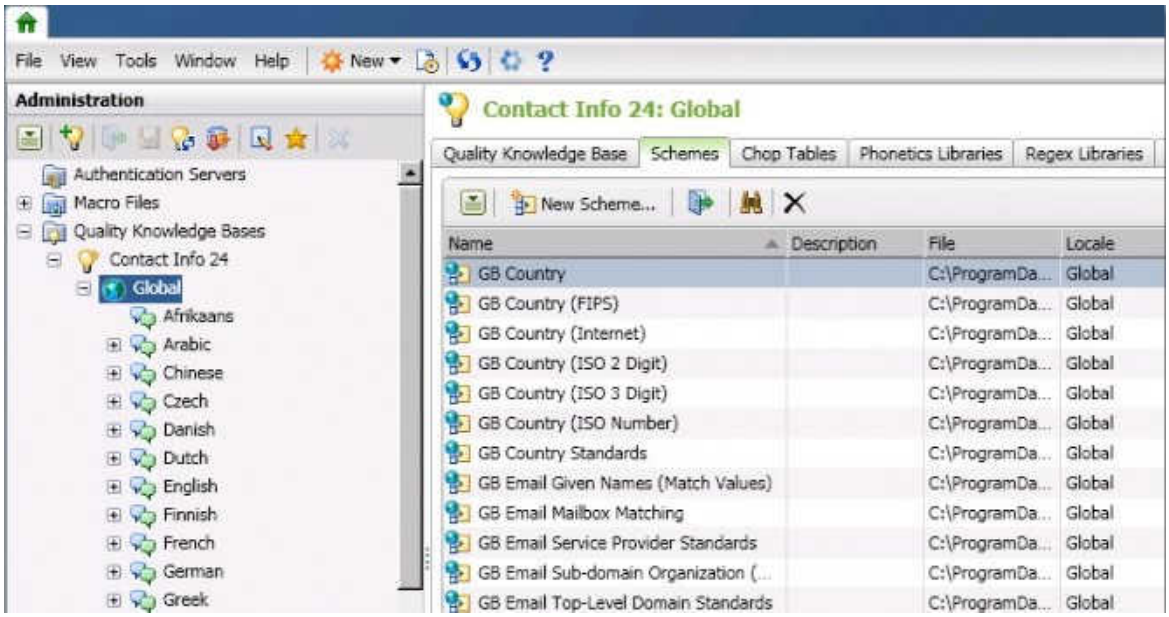
To view errors and warnings associated with opening a QKB, select **View Error Log** from the menu in the first toolbar at the top of the **Quality Knowledge Base** tab. This is the toolbar with the **Close QKB** option.

You can view the definitions and other items that are associated with a language. To do this, select a language in the tree on the left, as shown in the next display.



These definitions and other items on the right are inherited by all locales that appear under the selected language in the tree on the left. For more information about inheritance within a QKB, see Inheritance in a QKB on page 163.

The **Global** item contains definitions and other items that are inherited by all locales in that QKB. Select **Global** in the tree on the left to see definitions and other shared items on the right. The next display shows the schemes that are shared on the **Global** level.



After you have selected a locale, a language, or the **Global** item on the left, you can browse its contents on the right and open individual objects for editing. These objects include definitions and data types, plus libraries that are used to construct definitions. Definitions and data types can be browsed and edited in the Quality Knowledge Base on page 12 tab. The various types of libraries in a QKB have separate tabs: Schemes on page 115, Chop Tables on page 25, Phonetics Libraries on page 55, Regex Libraries on page 89, Vocabularies on page 134, and Grammars on page 34.

Accessing Your QKB Documentation

You can access the QKB documentation using one of the following options.

- From SAS Viya on page 8
- From the Help Menu on page 9
- From the UNIX File System on page 9
- From the Windows File System on page 9

From SAS Viya

The latest QKB documentation is available at SAS Help Center under the SAS Viya Platform documentation. Click **Data** > **Quality Knowledge Base** > **User's Guide**. See the SAS Quality Knowledge Base: User's Guide for more information.

From the Help Menu

The online documentation for a QKB is installed when the software is installed. One way to display help for a QKB is to right-click a registered QKB on the **Administration** tab and then select **Open QKB Help** from the menu. You can also click the QKB and then select **Open QKB Help** from the menu on the toolbar on the left or right.

From the UNIX File System

You can use a web browser to open the documentation at the following default location.

```
<SASHome>/SASQualityKnowledgeBase/<type>/<version>/qkb/doc/html
```

From the Windows File System

You can use a web browser to open the documentation at one of the following default locations.

Under Windows, for QKB CI 22 and later	`drive:\Program Files\SAS\QKB \<type><version>_<unique_identifier>\doc\html \qltykb1000.html`
Under Windows, for CI 2010B to QKB CI 2013A	`drive:\Program Files\SAS\QKB\<type>\<version>\doc\html \qltykb1000.html`
Under Windows, for QKB CI 2010A and earlier	`drive:\Program Files\SAS\QltyKB\<type>\<version>\Help \QKB\qltykb1000.html`

Accessing Logs

The following log files are provided for DataFlux Data Management Studio:

- **Studio log** for DataFlux Data Management Studio events
- **Platform log** for DataFlux Data Management Studio engine events

- **DAC log** for data access events
- **TKTS log** for events related to Base data sets and data sent across the wire to the SAS Federation Server

The **Studio log**, the **Platform log**, and the **DAC log** are enabled by default and are all logged to the platform log file. This log file includes a name prefixed with `platform_`. The **Platform log** file is saved to a user's profile area by default.

Windows XP path for platform_ log file:	`drive:\Documents and Settings\[username]\ApplicationData\SAS\DMStudio\[instance]`
Windows 7 path for platform_ log file:	`drive:\Users\[username]\AppData\Roaming\SAS\DMStudio\[instance]`

Note: Where you see `[instance]` is the instance name, which is chosen at install time and defaults to '`studio1`'.

You can save status history files for the data jobs and process jobs that you run. For information, see Reviewing the Status History of a Job. You can review and delete these status history files in the **Run Status History** section of the **Data Job** pane or the **Process Job** pane for a job when you select the job in the **Folder** riser bar.

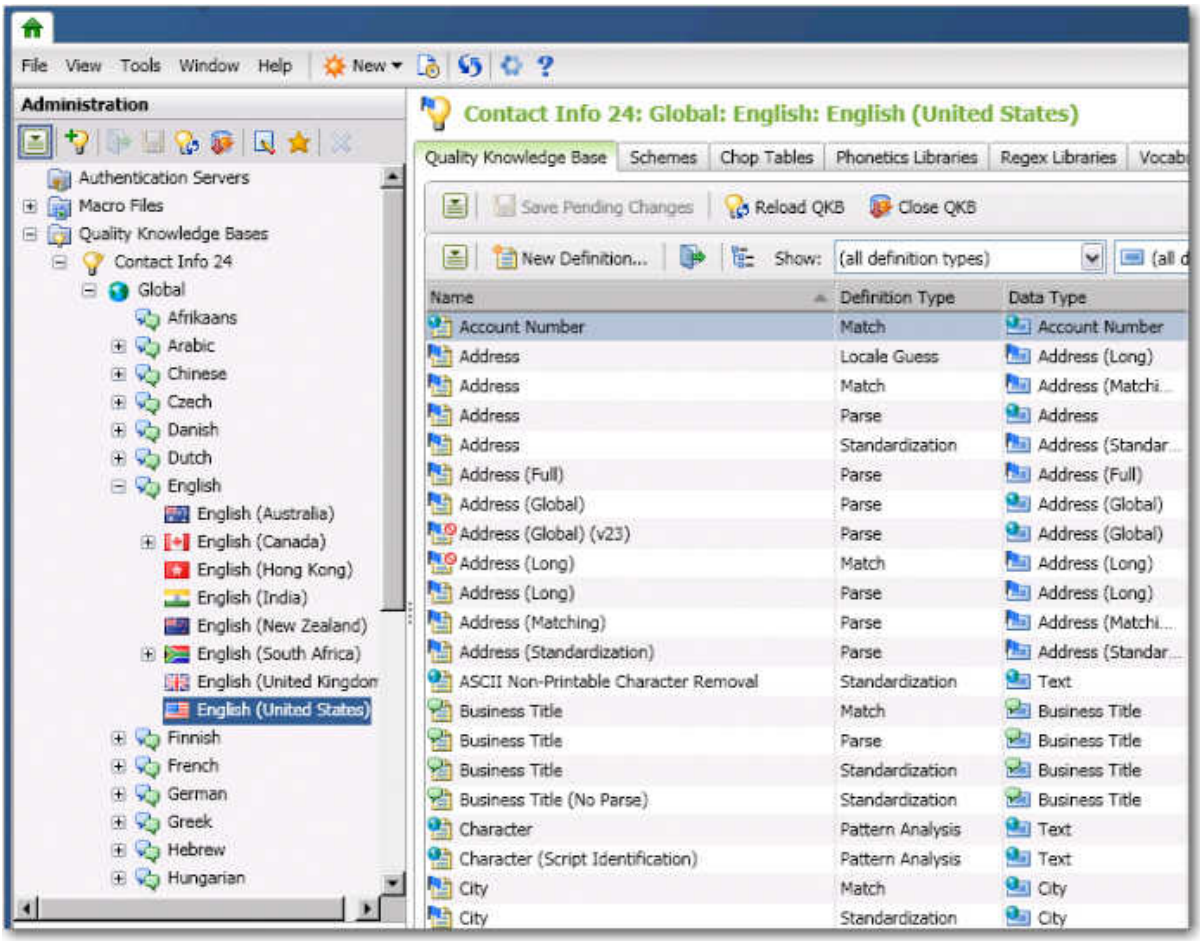
For a description of the logging options that can be set for , see the "Logging Options" topic in the DataFlux Data Management Studio Installation and Configuration Guide.

3

QKB Editors

Navigating the Quality Knowledge Base Tab

When you right-click a locale in the **Administration** riser bar and select **Open QKB**, the definitions for that locale appear on the **Quality Knowledge Base** tab. The next display shows the definitions for the **English (United States)** locale.



You can browse and manage QKB definitions in this tab. You can also manage data types, which are used to organize definitions.

QKB Definitions and Data Types

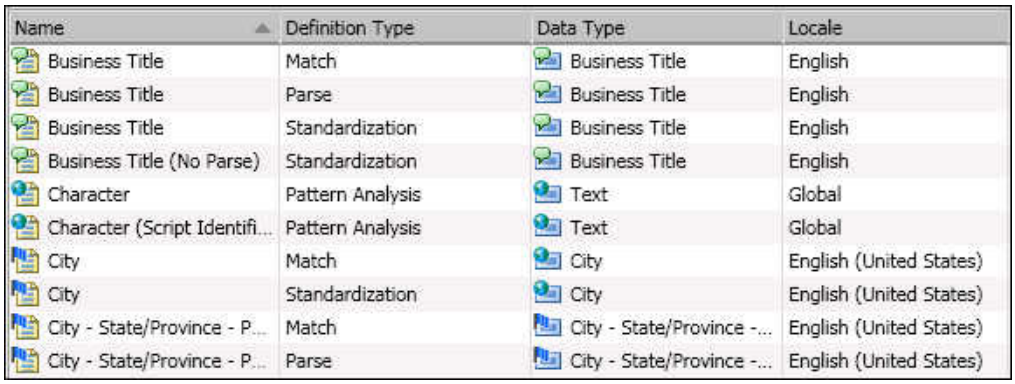
A QKB definition is a callable object that performs a data quality operation such as parsing, standardization, or fuzzy matching. For example, the Parsing node in a DataFlux Data Management Studio data job enables you to select a QKB definition in order to parse your data. You can also invoke definitions in other SAS Data Management products, such as SAS Data Quality Server and SAS Data Quality Accelerators.

Each definition belongs to a data type. A QKB data type is a semantic type of text data. In a QKB, a data type consists of a list of tokens. Tokens represent possible substrings that might be found in a text string associated with a given data type. For example, the tokens for a Name data type might include Given Name, Middle Name, Family Name, and so on. A data type's tokens define the possible inputs and outputs of the data type's definitions.

Data types and definitions are organized by locale. When you select a QKB locale in SAS Data Management software, you have access to all the definitions that are associated with that locale.

Main Columns on the Quality Knowledge Base Tab

The list of definitions in the **Quality Knowledge Base** tab has several columns.



Name	Definition Type	Data Type	Locale
Business Title	Match	Business Title	English
Business Title	Parse	Business Title	English
Business Title	Standardization	Business Title	English
Business Title (No Parse)	Standardization	Business Title	English
Character	Pattern Analysis	Text	Global
Character (Script Identifi...	Pattern Analysis	Text	Global
City	Match	City	English (United States)
City	Standardization	City	English (United States)
City - State/Province - P...	Match	City - State/Province -...	English (United States)
City - State/Province - P...	Parse	City - State/Province -...	English (United States)

The **Name** column shows the name of a definition. The **Definition Type** column shows the type of data quality operation performed by that definition, such as Parse, Standardization, or Match. Note that there might be two definitions with the same name but with different definition types. For example,, there could be an Address Parse definition and an Address Match definition.

The third column, **Data Type**, shows the name of the data type to which the definition belongs. Each definition belongs to exactly one data type, but multiple definitions can belong to the same data type.

The **Locale** column shows the QKB locale with which the definition is associated. Note that a definition might be listed even if it is not associated with the locale that you have selected in the **Quality Knowledge Bases** tree. This is because locales can inherit definitions from ancestors.

Select a locale and look at the values in the **Locale** column. Some definitions are associated directly with the selected locale. Others are associated with the locale language, and still others are associated with the **Global** set of objects. This information is also presented in the icons that are shown for the definitions in the list. As shown in the previous display, if an icon has a flag overlay, the definition is associated directly with a locale. If the icon has a word balloon overlay, it means that the definition is associated with a language. An earth overlay means that the definition belongs to the **Global** set of definitions.

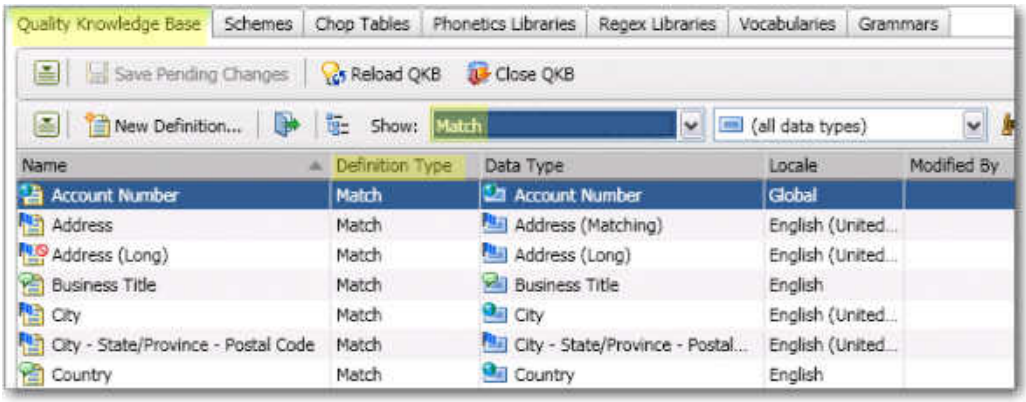
If you want to view only the definitions that are associated with a language, select that language in the tree for your QKB in the **Administration** riser.

If you want to view only the definitions in the **Global** set, click on **Global** in the tree for your QKB.

The **Modified By** column (not shown in the display above) shows the name of the user who last modified the definition. If the definition has not been modified since it was installed, this column is blank.

Filter by Definition Type or Data Type

When browsing a list of definitions, you might wish to view only definitions of a certain type. To do this, select a definition type in the first drop box next to the **Show** label in the lower toolbar.

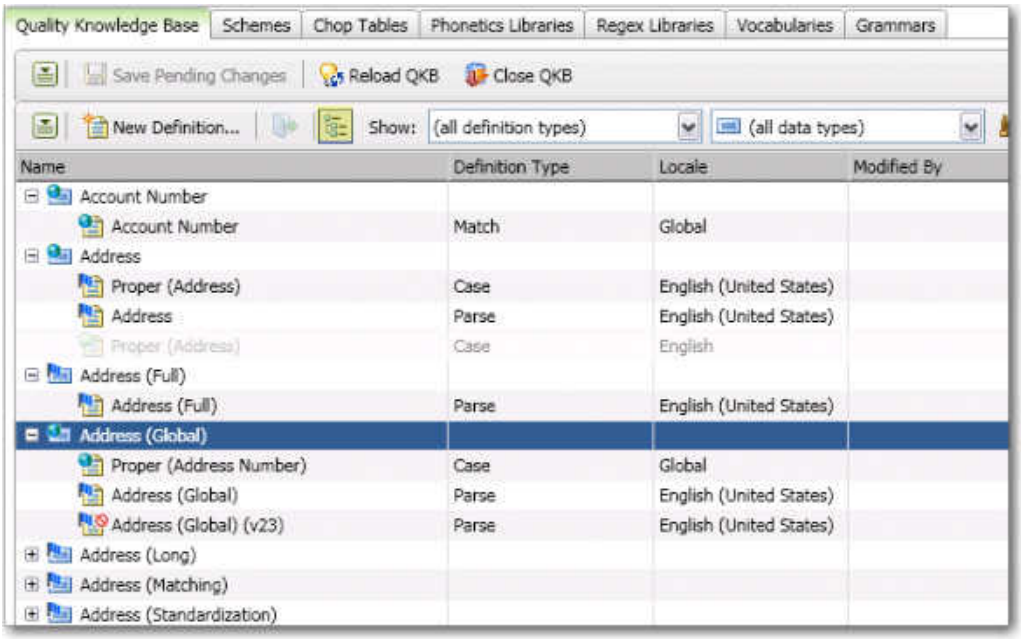


To turn off the filter, select **all definition types** in the drop box.

You might also wish to view only definitions that belong to a certain data type. To do this, select a data type in the second drop box in the lower toolbar. This is the drop box that says **all data types** in the previous display.

Group by Data Type

For an alternate view of the definition list, you can choose to group definitions in a tree according to their data types. Click the tree icon in the lower toolbar, as shown in the next display.



The **Name** column now shows the names of all data types that are available in the selected locale. Expand a data type entry to see the definitions that belong to that data type. To toggle back to the table view, simply click the tree icon in the lower toolbar again.

Find a Definition by Name

To find a definition with a specific name, click the **Find** icon (binocular icon) in the lower toolbar. This toggles on the Find pane. To toggle off the Find pane, click **Find** again.

Rename and Duplicate Definitions

You can rename a definition by right-clicking the name and selecting **Rename** from the pop-up menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a definition by right-clicking the name and selecting **Duplicate** from the pop-up menu. You can also click the name and select **Duplicate** from the menu.

Create a New Definition

To create a new definition, click the **New Definition** button on the lower toolbar, or select **New Definition** from the menu on the lower toolbar. After you have created

the new definition, the definition is displayed in the definition list, and a new tab opens that shows the flow diagram for the definition. See Editing a QKB Definition on page 153 for information about how to edit a definition after you have created it.

Override an Inherited Definition

As described in Overriding a Definition or Data Type on page 202, if a definition already exists in an ancestor, and you create a new definition with the same name in the selected locale, the new definition overrides the existing definition. You can see whether a definition is overridden by looking at the definition list. If a definition is overridden, it appears in a dimmed font.

For example, in the next display, an English (United States) definition has been added for Account Number. The local definition overrides the **Global** definition of the same name. The **Global** definition is automatically overridden and appears in a dimmed font.

Name	Definition T...	Data Type	Locale
Account Number	Match	Account Number	English (United States)
Account Number	Match	Account Number	Global
Address	Locale Guess	Address (Long)	English (United States)

You can view and edit an overridden definition, but you cannot use that definition in a data job. Instead, you use the definition of the same name that is associated directly with your locale.

Delete a Definition

To delete a definition, select the definition and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu or in the menu in the lower toolbar. The definition is removed from the QKB.

Deprecate a Definition

At times you might wish to deprecate the usage of a definition. This is a way of alerting other users that you will delete this definition in the future. When a user executes a data job that uses a deprecated definition, a warning is shown in the data job's log. The warning informs the user that the definition has been deprecated.

You can also specify the name of a definition that should be used in place of the deprecated definition. This helps users to update their data jobs before the deprecated definition is deleted. To deprecate the usage of a definition, select the definition and choose **Deprecate Usage** in the pop-up menu or in the menu in the

lower toolbar. Here, you might optionally enter the name of a definition that should be used in place of the deprecated definition. Click **OK** when done.

When a definition has been deprecated, its icon is displayed with a deprecation overlay. The next display shows the deprecation overlay on the English (United States) definition for Account Number.

Name	Definition T...	Data Type	Locale
Account Number	Match	Account Number	English (United States)
Account Number	Match	Account Number	Global
Address	Locale Guess	Address (Long)	English (United States)

To undo the deprecation of a definition, select the definition and choose **Undo Deprecation** in the pop-up menu or in the menu in the lower toolbar.

Save Pending Changes

If you have made changes to your QKB, save those changes by clicking **Save Pending Changes** in the upper toolbar or by choosing **Save Pending Changes** in the menu in the upper toolbar. Your changes are saved to the QKB.

Reload QKB

To load the last saved version of the QKB into memory within DataFlux Data Management Studio, click **Reload QKB** in the upper toolbar, or choose **Reload QKB** in the menu in the upper toolbar. All saved changes made within the respective QKB editors for QKB Library files is loaded into DataFlux Data Management Studio. Any unsaved changes to definitions are discarded. The reload action is useful for maintaining consistency between the disk and memory copies of the QKB being edited.

Close QKB

To close a QKB, click **Close QKB** in the upper toolbar, or choose **Close QKB** in the menu in the upper toolbar. If you have made edits to the QKB, you are prompted as to whether you would like to save your changes before closing. If you choose **No**, your changes are lost. Note that only one user can have a QKB open at a time. Remember to close the QKB when you are done browsing and editing so that other users can open that QKB.

Open a Definition for Editing

To open a definition for editing, double-click on the definition in the list, or select the definition and click the **Open** icon on the lower toolbar. You might also select the definition and choose **Open** in the pop-up menu or in the menu on the lower toolbar. A new tab opens that shows the flow diagram for the selected definition. See Editing a QKB Definition on page 153 for information about how to edit a definition.

Manage Data Types

Select **Manage Data Types** to view a list of data types, create a new data type, or edit an existing data type.

Error Log

If some error condition occurs when you open a QKB, error messages are written to the **Error Log** window. Display this window by choosing **View Error Log** from the menu on the upper toolbar. For more information, see Viewing the QKB Error Log on page 162. .

Chop Table Editor

Overview

Before using the Chop Table editor, you should define all of the chopping characteristics for each value in the default character table.

A chop table is a collection of character-level rules used to create an ordered word list from an input string. Each character in the default character table has specified both a classification and operation. You should build one chop table per parse definition. The Chop Table Editor enables you to build a chop table.

Table

Unicode Block - The **Unicode Block** drop-down list provides a list of character subsets, see Unicode Block for the complete list.

Character Name - The **Character Name** is the actual name of the character for example, semicolon.

Character - The **Character** represents the actual appearance of the character. For example, the **Character Name** is **semicolon** and the **Character** is **;**.

Classification - The **Classification** drop-down list includes:

- **LETTER/SYMBOL** - a letter or non-separating symbol
- **NUMBER** - a numeric digit (0-9)
- **LEAD SEPARATOR** - a delimiter attached to the beginning of a word (for example, the left parenthesis)
- **TRAIL SEPARATOR** - a delimiter attached at the end of a word (for example, a period)
- **FULL SEPARATOR** - a delimiting character (for example, space, hyphen, and comma)

Operation - The **Operation** drop-down list includes:

- **USE** - use the character as-is in the word list and output tokens
- **TRIM** - omit from word list; trim leading and trailing characters in output tokens
- **SUPPRESS** - omit from the word list and output tokens

Value - This is the ordinal value of the Unicode code point.

Hex Value - The hexadecimal value of the **Value** (above).

Rules

The rules-based chopping algorithm works by matching portions of an input string from left to right using search criteria and states. These are specified by rules, which can be defined from the **Rules** tab. These rules are processed from top to bottom and the search criteria includes:

- A vocabulary of words
- A single regular expression

The system uses the search criteria to first match the input string at the current position. If it fails, it proceeds to the next rule and attempts to match using the criteria for that rule, and so on. If no rules match, the input string position is advanced by one character and the algorithm run again. This process is repeated until the process reaches the end of the input string.

If the system is able to successfully match a substring, it then attempts to validate the state. The system maintains a state of zero or more flags at any time, these are variables that either exist or not. Each rule has the ability to check for the existence of flags in the current state using a simple Boolean syntax. This is called the Prerequisite State Condition. If this validation fails, the rule fails to match and the next rule is checked, and so on. If it succeeds, then the following occurs:

- The input string is advanced to the end of the successful match in the Search criterion.
- A new state (Output State) consisting of zero or more flags is set. The new state replaces the old state.
- The string is chopped before the current match, after the current match, at both points, or not at all.

This part of the **Rules** tab sets up an initial state before any rule processing is performed. This is useful for identifying a pre-existing state in order to force certain rules first.

The **Initial Flags** list has two controls. Click **Add** to open **Add New Flag**. Here, you enter the name of the new initial flag. As you create new flags, it is added alphabetically to the list. To delete an existing flag, select the flag and click **Delete**.

Details

This section displays each rule and its associated components, in order.

Method - The Method is either Vocab or Regex, depending on the type of criterion for the rule. This determines the format for the Search Criterion column.

Search Criterion - This section displays the search criterion. The exact format differs, depending on the method selected for the rule. If the method is Regex, then this field displays a regular expression. If the method is Vocab, then this field displays the name of the vocabulary selected, a comma, and the name of the category in the vocabulary.

Prerequisite State Condition - This displays a logical Boolean expression describing the desired flag configuration of the current state in order for the rule to match. It consists of flag names separated by various operators. The expression is parsed left to right, precedence given to sub-expressions in parentheses .

Value	Description
|	OR operator
&	AND operator
!	NOT operator
()	Parenthetical operators, for precedence

Output State - This field displays a comma-separated list of flag names that become the current state should this rule be matched.

Chop Mode - This option tells the system what to do with the input string when the match is successful. The possibilities include:

Value	Description
None	No chop. This option is helpful if you want to change the state on a certain condition.
Before	Chop the string before the match.
After	Chop the string after the match.
Both	Chop the string before and after the match.
Notes	This is a comment field used to display informational messages about each rule.

Use the buttons on the right to create, edit, and delete rules.

Icon	Description
	Add new rule
	Edit rule
	Delete rule
	Move rule up
	Move rule down

When you click **Add a rule**, **Add Rule** opens.

Matching Method - In the **Matching Method** section, select **Regular Expression** or **Vocabulary**. When you select **Regular Expression** you can enter the regex directly into the accompanying field. If you select **Vocabulary**, you must select one of the available vocabularies from the drop-down list. When you select a vocabulary, the **Category** drop-down list becomes active. All of the possible categories for the selected vocabulary appear. The **All** category is always present in this list. This category allows all words in the vocabulary to be used as possible matches.

Prerequisite State Condition - This is a Boolean text expression.

Output State - The **Output State** list can be populated with flag names just like **Initial Flags**.

Chopping Mode - The **Chopping Mode** drop-down list enables you to select one of the possible modes.

Notes - The **Notes** field enables you to add 128 characters of text. This is used for documentation purposes.

Testing

The test area is used to test input strings against the chopping configuration.

Input String

The Input string field accepts any form of text input.

Go - Click **Go** to run the string through the Chop Table Editor for a result. The result is displayed in the **Result** section.

Clear - Click **Clear** to clear both the Input string and **Result fields**.

Result

The **Result** section includes two columns, **Phrase** and **Source**.

Phrase - The **Phrase** column shows the chopped substring.

Source - The **Source** column displays whether the corresponding chopped phrase originated from the table or the rules by displaying **Table** or **Rules**. The first rows in this table always show **Beginning of Line in the Source** column because this represents some beginning prefix of the input string that was not chopped. All subsequent rows specify an explicit source.

Double-click any row in the **Result** table to see why the substring on that row was chopped.

Character-Level Options

Classifications

Classification	Description
LETTER	A letter or non-separating symbol
NUMBER	A numeric digit (0-9)
FULL SEPARATOR	A delimiting character, for example, spaces, hyphens, and commas
LEAD SEPARATOR	A delimiter attached to the beginning of a word, for example, left parenthesis
TRAIL SEPARATOR	A delimiter attached at the end of a word, for example, a period

Operations

Operation	Description
USE	Use the character as-is in the word list and output tokens.
TRIM	Omit from word list; trim leading and trailing characters in output tokens.
SUPPRESS	Omit from the word list and output tokens.

Implicit Separators

Implicit separators are an optional feature of string chopping. For each parse definition, you can enable or disable implicit separation. If you enable implicit separators, a separator mark is placed wherever the classification of a character differs from the classification of the previous character in a string. To use implicit separation, on the Chop Table Editor screen, choose **Options** > **Implicit Separation**.

For example, using implicit separation, for the phrase `APT124` is separated into two phrases.

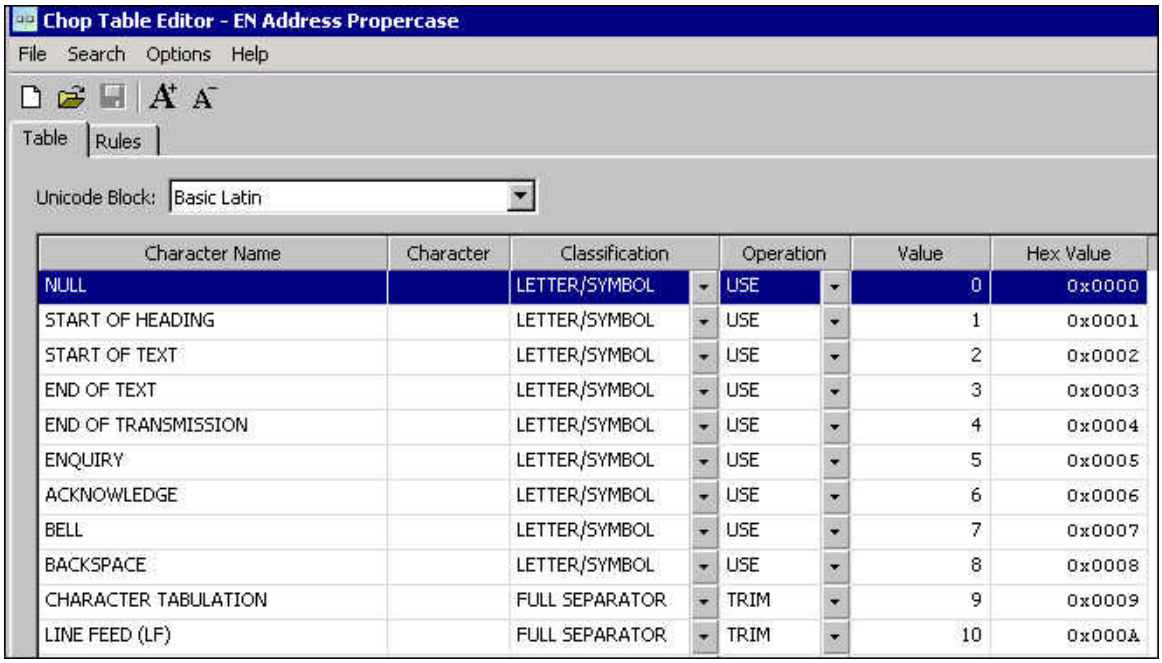
```
APT 124
```

This assumes that the letter "T" is assigned one classification such as "Letter" and the number "1" is assigned a different classification such as "Number." If both characters have the same classification, they remain together. With implicit separation disabled, this phrase could be broken apart only by assigning separator functionality to the "T" or to the "1," but that change would affect every instance of those characters.

Navigating the Chop Table Editor

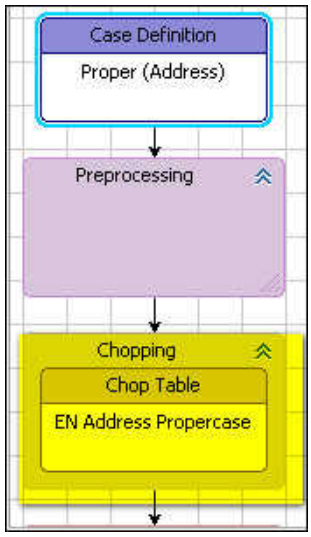
A chop table is a QKB file containing character-level rules that are used to split apart strings of text into an ordered list of words. Each chop table is collection of character-level rules that are used to create an ordered word list out of an input string.

Each character in the default character table has a specified classification and an operation. The next display shows some of the rules for the EN Address Proper Case chop table.



Character Name	Character	Classification	Operation	Value	Hex Value
NULL		LETTER/SYMBOL	USE	0	0x0000
START OF HEADING		LETTER/SYMBOL	USE	1	0x0001
START OF TEXT		LETTER/SYMBOL	USE	2	0x0002
END OF TEXT		LETTER/SYMBOL	USE	3	0x0003
END OF TRANSMISSION		LETTER/SYMBOL	USE	4	0x0004
ENQUIRY		LETTER/SYMBOL	USE	5	0x0005
ACKNOWLEDGE		LETTER/SYMBOL	USE	6	0x0006
BELL		LETTER/SYMBOL	USE	7	0x0007
BACKSPACE		LETTER/SYMBOL	USE	8	0x0008
CHARACTER TABULATION		FULL SEPARATOR	TRIM	9	0x0009
LINE FEED (LF)		FULL SEPARATOR	TRIM	10	0x000A

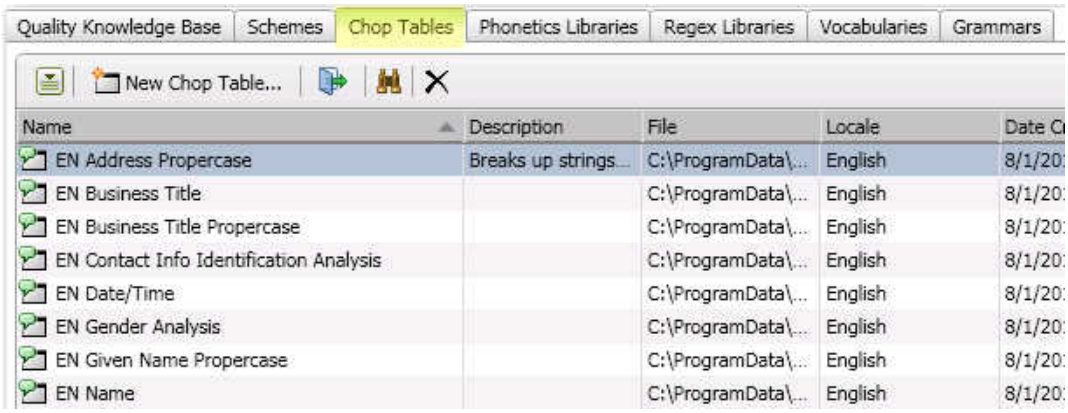
It is recommended that you build one chop table per QKB definition. The next display shows the EN Address Proper Case chop table in the context of the Proper (Address) case definition in the English locale in the SAS Quality Knowledge Base for Contact Information 24.



Chop Tables Tab

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right-hand side of the screen. The **Chop Tables** tab shows a list of chop tables that are contained in the QKB.

The list of chop tables in the **Chop Tables** tab has several columns, as shown in the next display.



The Name column shows the name of a chop table. The Description column contains text about the chop table, such as "Breaks up strings for city/state/ ZIP Propercase." The File column contains the physical path to the chop table file.

The Locale column shows the QKB locale with which the chop table is associated. Note that a chop table might be listed even if it is not associated with the locale that you have selected in the Quality Knowledge Bases tree. This is because inherit chop tables from ancestors on page 163. locales can .

Select a locale and view the values in the chop table Locale column. Some chop tables are associated directly with the selected locale, others are associated with the locale language, or with the Global set of objects. This information is also

presented in the icons that are shown for the chop tables in the list. As shown in the previous display, if an icon has a flag overlay, the chop table is associated directly with a locale. If the icon has a word balloon overlay, it means that the chop table is associated with a language. An earth overlay means that the chop table belongs to the Global set of chop tables.

If you want to view only the chop tables that are associated with a language, select that language in the tree for your QKB in the Administration riser.

If you want to view only the chop tables in the Global set, click **Global** in the tree for your QKB.

Find a Chop Table by Name

To find a chop table with a specific name, click **Find** in the lower toolbar. This toggles on the **Find** pane. To toggle off the **Find** pane, click **Find** again.

Rename and Duplicate Chop Table

You can rename a chop table by right-clicking the name and selecting Rename from the pop-up menu. You can also click the name and select **Rename** from the menu from the lower toolbar.

You can make a copy of a chop table by right-clicking the name and selecting Duplicate from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Chop Table

To create a new chop table, click **New Chop Table** from the lower toolbar, or select **New Chop Table** from the menu on the lower toolbar. After you have created the new chop table, the chop table is shown in the chop table list, and the Chop Table Editor on page 19 appears.

Open a Chop Table for Editing

To open a chop table for editing, double-click on the chop table in the list, or select the chop table and click **Open** from the lower toolbar. You might also select the chop table and choose **Open** in the pop-up menu or in the menu on the lower toolbar. See Chop Table Editor on page 19 for information about how to edit a chop table.

View Definitions That Use a Given Chop Table

To view a list of QKB definitions that use a given chop table, select the chop table and choose **Usage** in the pop-up menu or from the menu on the lower toolbar. A usage list of definitions that use the selected chop table is displayed.

Delete a Chop Table

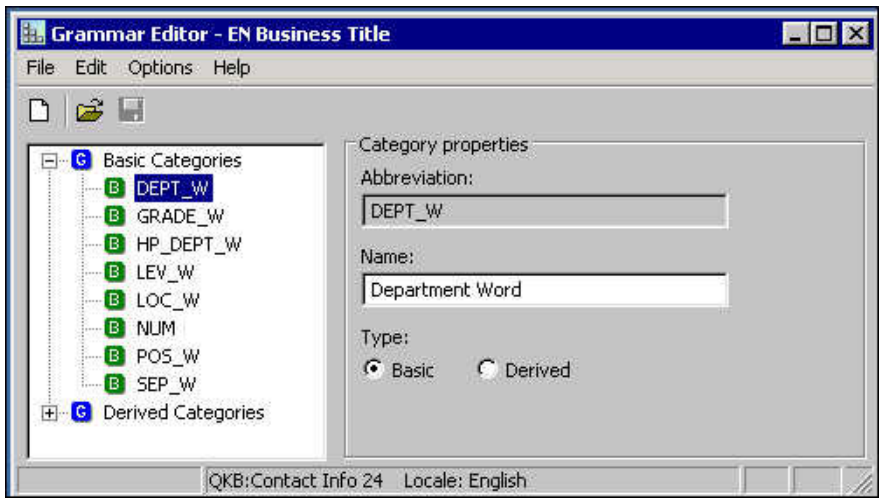
To delete a chop table, select the chop table and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu or from the menu in the lower toolbar. The chop table is removed from the QKB.

Grammar Editor

Overview

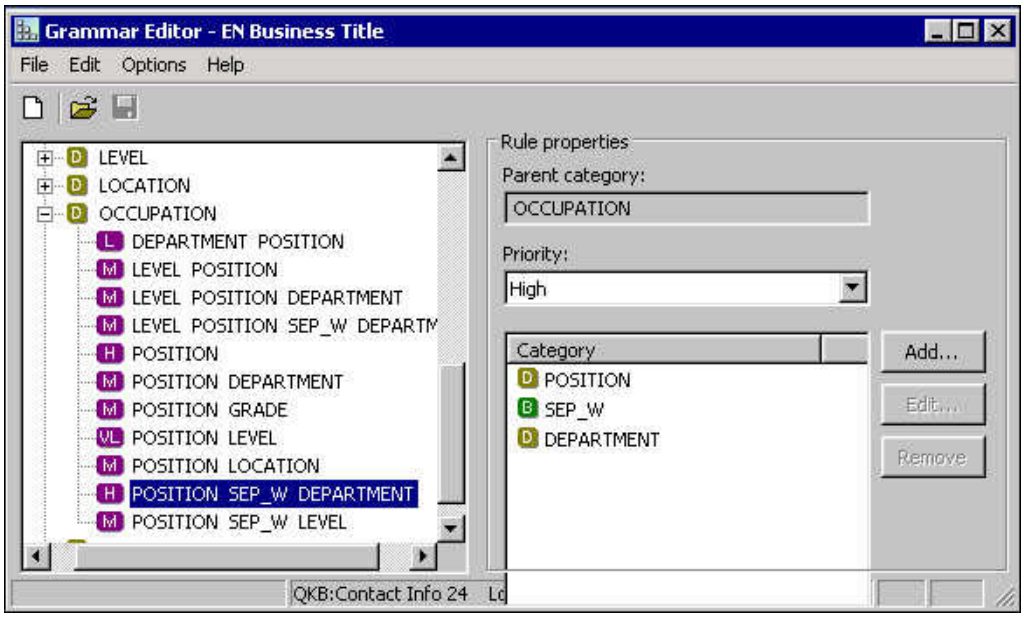
Use the Grammar Editor to maintain grammars in a QKB. A grammar is a file that contains a set of rules that represent expected patterns of words in a given context. The parsing system uses a Vocabulary on page 126 to identify the basic categories for each word. Then the patterns constructed from the categories on those words are compared with rules in the grammar.

Rules in a grammar contain patterns that consist of one or more symbols called categories. There are two types of categories in a grammar: **Basic Categories** and **Derived Categories**. The next display shows some of the Basic Categories listed in the EN Business Title grammar in the English locale in the SAS Quality Knowledge Base.



Basic Categories are used to categorize words in a vocabulary by their semantic type. For example, the Basic Categories in the EN Business Title grammar shown in the previous display are used to categorize words in the EN Business Title vocabulary in the same locale and QKB. Two of the semantic types represented in the previous display are DEPT_W (Department Word) and GRADE_W (Grade Word).

Derived Categories represent sets of one or more rules that specify patterns that define a semantic concept. A pattern in a rule consists of one or more **Basic Categories** or other **Derived Categories**. For example, the next display shows the **Occupation Derived Category**.



The Occupation Derived Category can be derived by any of a number of rules, such as POSITION SEP_W DEPARTMENT. Each rule specifies a pattern containing one or more Basic or Derived Categories, plus a Priority value.

Use the menu or the toolbar to add new grammars, open existing grammars, or maintain the rules and categories in a grammar. The rules tree on the left displays the rules and categories in the current grammar. The **Category Properties** area on the right enables you to add and edit categories The **Rule Properties** area on the right enables you to add and edit rules. The status bar displays the current locale.

Menu

File

New - Enables you to create a new grammar. When you create a new grammar, you must specify a locale with which to associate the grammar. For more information about adding a new grammar, see Adding and Editing Grammars.

Open - Enables you to open a grammar in the current locale.

Close, **Save**, **Save As**, and **Exit** - Enable you to perform these operations on the grammar in the editor.

Edit

These options enable you to maintain the rules and categories in a grammar. The options are inactive until you open a grammar.

Add Category - Add a new category for grammar rules.

Change Category Abbreviation - Changes the abbreviation for a selected category. The abbreviation for a category is displayed in the rules tree.

Duplicate Category - Duplicates the selected category.

Add, **Copy**, and **Paste Rule** - Perform these actions for rules.

Delete Selected Item - Deletes the selected category or rule.

Options

Set QKB - When available, **Select Locale** is available so that you can specify the locale where you want to add or maintain a QKB grammar.

Toolbar

Toolbar buttons enable you to create, open, or save a grammar.

Rules Tree

The rules tree on the left displays the rules and categories in the current grammar. Each rule specifies one or more Basic or Derived Categories and a Priority value.

Category Properties

Abbreviation - Abbreviation of the category name that this displayed in the rules tree.

Name - Full name of the category.

Type - Basic or Derived category.

Rule Properties

Parent Category - Parent category for the selected rule.

Priority - Priority of the selected rule.

Category - One more category included in the selected rule.

Adding and Editing Grammars

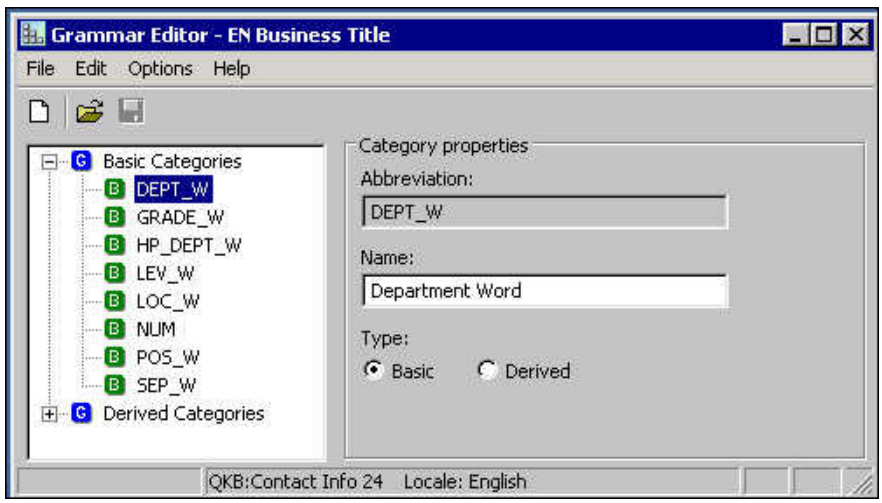
Add a new grammar or update an existing grammar when the categories and rules in existing grammars do not meet your needs. To see existing grammars on the Grammars on page 34 tab. Use these sources to plan the basic categories, derived categories, and rules that you need.

Note: The implementation of a grammar can be a complex task. If you are unfamiliar with grammars, you might want to attend a SAS QKB Customization training course. For information about training courses available from SAS, contact SAS Technical Support.

Add a Grammar and Some Basic Categories

This topic describes how to add a new grammar file to a locale in a QKB and add some basic categories. One reason to do this would be that you are adding a new vocabulary, and you need a new grammar and some basic categories to match the semantic types of the words in the new vocabulary.

The next display shows some of the basic categories listed in the EN Business Title grammar in the English locale in the SAS Quality Knowledge Base for Contact Information 24.



Two of the semantic types represented in the previous display are DEPT_W (Department Word) and GRADE_W (Grade Word).

Perform the following steps to add a new grammar file to a locale in a QKB and add some basic categories. It is assumed that you have opened the QKB from the **Administration Riser**, and the **Grammars** tab is displayed.

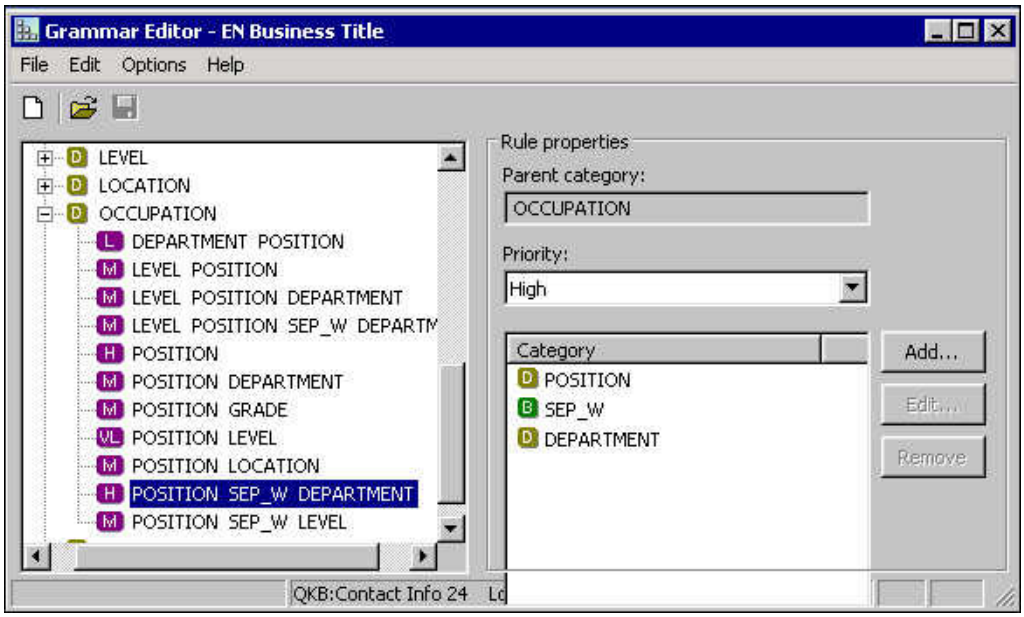
1. Click **New Grammar** on the lower toolbar, or select **New Grammar** from the menu on the lower toolbar. The New Grammar wizard opens.
2. Enter a name and description for the new grammar and click **Next**.
3. Select a locale for the grammar and click **Finish**. The new grammar appears in the grammar list, and the Grammar Editor opens.
4. To add a new basic category, select the **Basic Categories** folder. Then select **Edit** > **Add Category** from the menu.
5. Specify an abbreviation and a name for the category in **Add Categories**.

   The abbreviation is a shorthand name for the category. It is used to optimize readability when the category is displayed in an area with limited space, such as in a tree view. The name of the category is a full descriptive name. Both the abbreviation and the name of a basic category should be appropriate for a semantic type of a word in a vocabulary.
6. Click **OK** to save the new category.
7. Repeat the previous steps to add more basic categories, if desired.
8. If desired, add derived categories and rules as described in the next topic.
9. Select **File** > **Save** to save the new grammar. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.
10. Click **Yes** to reload the QKB.

Add a Derived Category and Some Rules

This topic describes how to add derived category to an existing grammar file. One reason to do this would be that you want to create a rule for a new string pattern such as **[Name]** > **[Given Name] [Family Name]**.

Derived categories represent sets of one or more rules that specify patterns that define a semantic concept. A pattern in a rule consists of one or more basic categories or other derived categories. For example, the next display shows the Occupation derived category.



The Occupation derived category can be derived by any of a number of rules, such as POSITION SEP_W DEPARTMENT. Each rule specifies a pattern containing one or more basic or derived categories, plus a **Priority** value.

Perform the following steps to add a derived category and some rules to an existing grammar file. It is assumed that you have opened the QKB from the **Administration Riser**, and the **Grammars** tab is displayed. It is assumed that you have a general idea what new rules you want to create, such as a rule for a string pattern such as **[Name]** > **[Given Name] [Family Name]**.

1. Double-click the grammar that you want to update. Alternatively, you could right-click a grammar and select **Open**, or select a grammar and then click **Open**. The Grammar Editor displays the selected grammar.

2. Click the plus sign beside the main **Derived Categories** folder to expand it. Note the main derived categories, such as LEVEL, LOCATION, and OCCUPATION in the previous display.

3. Click the plus sign beside a derived category to expand it, such as the OCCUPATION category. Note the rules within that category, such as POSITION SEP_W DEPARTMENT in the previous display.

4. To add a new derived category, right-click the main **Derived Categories** folder or a derived category. Then select **Add Category** from the pop-up menu .

5. Specify an abbreviation and a name for the category under **Add Categories**.

   The abbreviation is a shorthand name for the category. It is used to optimize readability when the category is displayed in an area with limited space, such as in a tree view. The name of the category is a full descriptive name. Both the abbreviation and the name of a basic category should be appropriate for a semantic type of a word in a vocabulary.

6. Click **OK** to save the new derived category.

7 To add a rule within a derived category, right-click the derived category. Then select **Add Rule** from the pop-up menu. The **Rule properties** panel for the current category is displayed on the right.

8 Note the value in the **Priority** field of the panel. This value sets the priority for the rule that you are editing. At run time, if the software finds that an input string can be parsed in more than one way, the solution that uses the rules with the highest priorities is chosen. Use the selection arrow to change the **Priority** value as needed.

9 Click **Add** in the right panel. The **Add Rule Category** is displayed.

10 Use to select a rule category. Click **OK** to save this category as a rule.

11 Repeat steps 9 and 10 until you have added all of the desired categories in your rule.

12 Select **File** > **Save** to save the grammar. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.

13 Click **Yes** to reload the QKB.

Update Categories or Rules

You can update categories and rules from the pop-up menu or from the **Edit** menu under **Edit Grammar**. For example, to add a category to a grammar, right-click a parent folder and select **Add Category**. Alternatively, select a parent folder > **Edit** > **Add Category** from the menu.

To change the abbreviation for a category, right-click a category and select **Change Category Abbreviation**.

To duplicate a category, right-click a category and select **Duplicate Category**. If the category is a derived category, all of the category's rules are also duplicated.

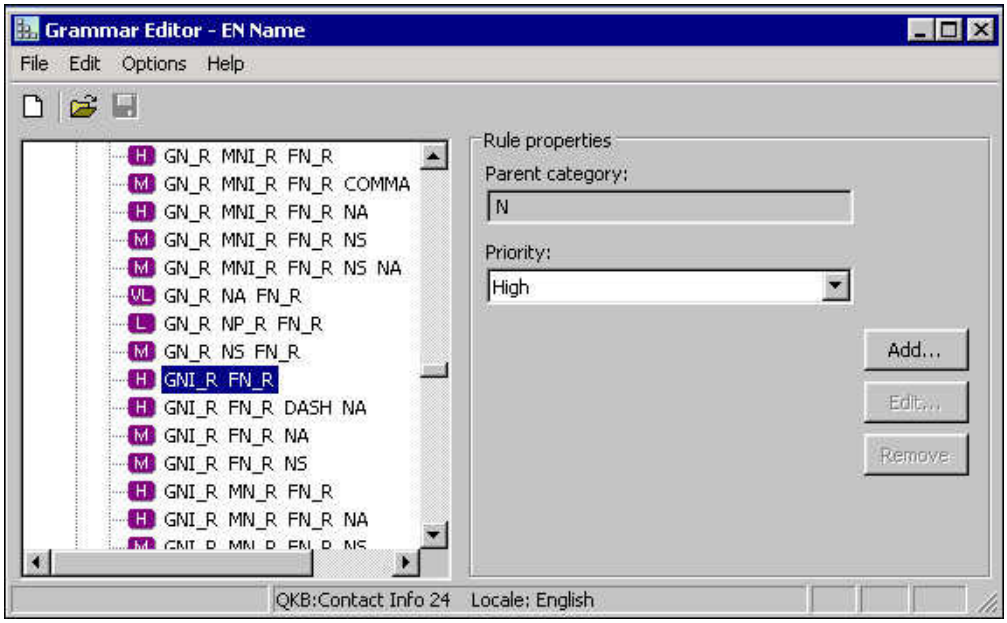
To add a rule, right-click a derived category and select **Add Rule**.

To copy and paste a rule, right-click a rule and select **Copy Rule**. Then select another derived category and select **Paste Rule**.

Navigating the Grammar Editor

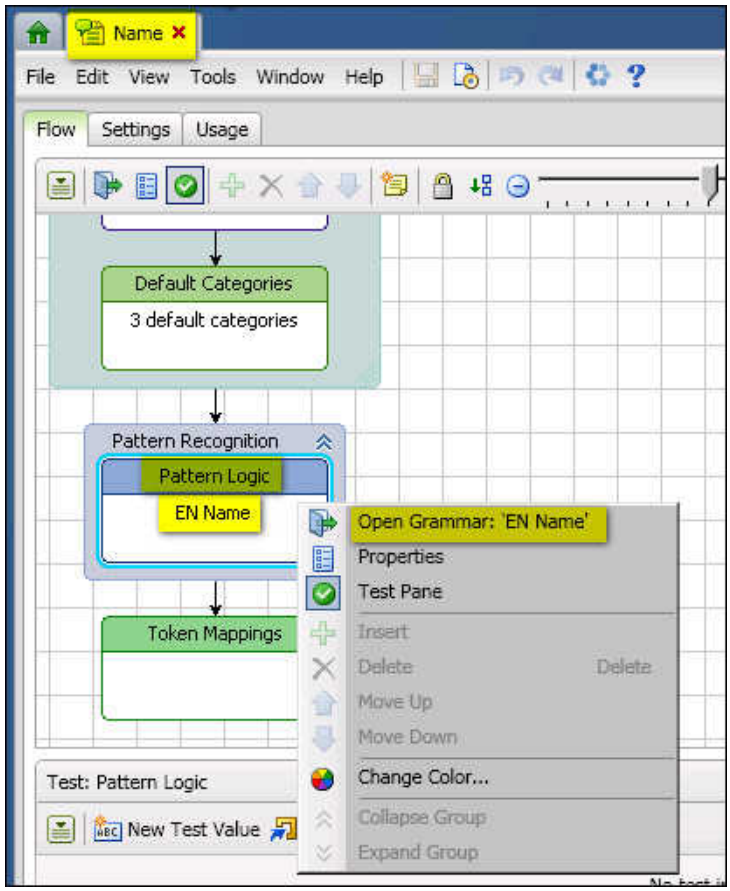
A grammar is a file that contains a set of rules that represent expected patterns of words in a given context. The parsing system uses a Vocabulary on page 126 to identify the basic categories for each word. Then the patterns constructed from the categories on those words are compared with rules in the grammar.

The next display shows some of the rules listed in the EN Name grammar in the English locale in the SAS Quality Knowledge Base for Contact Information 24.



In the previous display, there are many rules that can be used to derive the category N (not pictured), which stands for Name. For example, the GNI_R_FN_R rule represents a pattern containing a given name initial root (GNI_R) followed by a family name root (FN_R). The categories GNI_R and FN_R can then be derived from other rules. For example, the given name initial root (GNI_R) category might be derived from a rule representing a pattern containing a single initial. Grammar rules can be used in Pattern Logic nodes in a QKB definition to discover patterns in an input string.

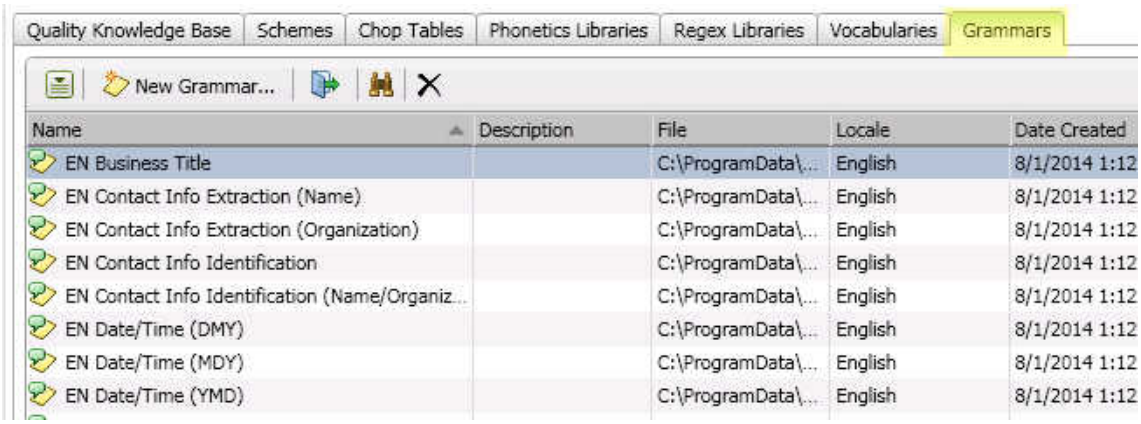
The next display shows the EN Name grammar in the context of the Name parse definition in the English locale. The grammar is specified in a Pattern Logic node.



Grammars Tab

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right-hand side of the screen. The **Grammar Libraries** tab shows a list of vocabularies that are contained in the QKB.

The list of vocabularies in the **Grammar Libraries** tab has several columns, as shown in the next display.



The Name column shows the name of a grammar. The Description column contains text about the grammar. The File column contains the physical path to the grammar file.

The Locale column shows the QKB locale with which the grammar is associated. Note that a grammar might be listed even if it is not associated with the locale that you have selected in the Quality Knowledge Bases tree. This is because locales can inherit grammars from ancestors on page 163. .

If you select a locale and look at the values in the Locale column for a grammar, there are some vocabularies that are associated directly with the selected locale. Others are associated with the locale language, and others are associated with the Global set of objects. This information is also presented in the icons that are shown for the vocabularies in the list. As shown in the previous display, if an icon has a flag overlay, the grammar is associated directly with a locale. If the icon has a word balloon overlay, it means that the grammar is associated with a language. An earth overlay means that the grammar belongs to the Global set of vocabularies.

If you want to view only the vocabularies that are associated with a language, select that language in the tree for your QKB in the Administration riser.

If you want to view only the vocabularies in the Global set, click **Global** in the tree for your QKB.

Find a Grammar by Name

To find a grammar with a specific name, click **Find** in the lower toolbar. This toggles on the **Find** pane. To toggle off the **Find** pane, click **Find** again.

Rename and Duplicate a Grammar

You can rename a grammar by right-clicking the name and selecting Rename from the pop-up menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a grammar by right-clicking the name and selecting Duplicate from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Grammar

To create a new grammar, click **New Grammar** on the lower toolbar, or select **New Grammar** from the menu on the lower toolbar. The New Grammar wizard opens. Specify a name, description, and a locale for the new grammar. After you have saved this information, the new grammar appears in the grammar list, and the Grammar Editor opens. Add categories and rules to the grammar as needed. For details about these tasks, see Adding and Editing Grammars.

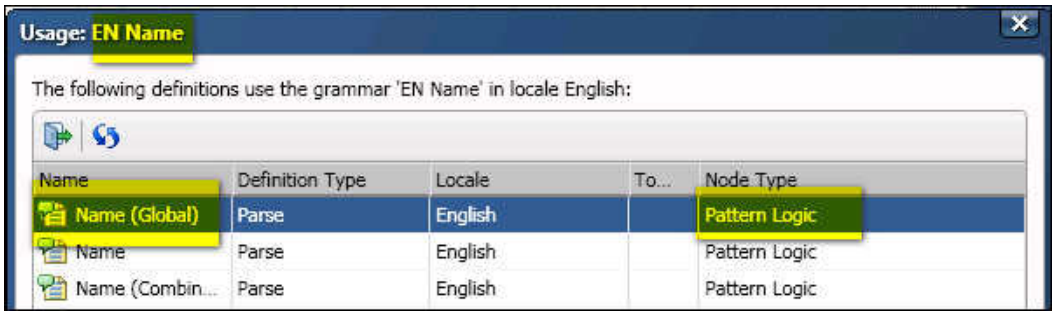
Open a Grammar for Editing

To open a grammar for editing, double-click on the grammar in the list, or select the grammar and click **Open**. You might also select the grammar and choose **Open** in the pop-up menu or from the menu on the lower toolbar. Update categories and

rules in the grammar as needed. For details about these tasks, see Adding and Editing Grammars on page 31.

View Definitions That Use a Given Grammar

To view a list of QKB definitions that use a given grammar, select the grammar and choose **Usage** in the pop-up menu or from the menu on the lower toolbar. Usage displays a list of the definitions that use the selected grammar. The next display illustrates how the Node Type column identifies the type of node that specifies the grammar in a given definition.



Delete a Grammar

To delete a grammar, select the grammar and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu or from the menu in the lower toolbar. The grammar is removed from the QKB.

Parse Definition Quick Editor

Overview

Parse Definition Quick Editor uses your data to automatically build chop tables, vocabularies, and grammars. The framework includes the Main Menu on page 38, toolbar on page 40, and tabs.

Main Menu

File

New - Creates a new Parse Definition Quick Editor job. See New Parse Definition on page 41 for additional information.

Open - Accesses an existing Parse Definition Quick Editor job. See Open Parse Definition on page 42 for additional information.

Save - Saves your Parse Definition Quick Editor job. For more information about the Save and Save As options, see Save or Save As.

Save As - Saves your Parse Definition Quick Editor job to specify a location and file name. For more information about the Save and Save As options, see Save or Save As.

Exit - Closes the Parse Definition Quick Editor.

Edit

Filter List - Creates filters for the data in the Parse Definition Quick Editor.

- **Column** - Click the **Column** drop-down list for the following options: Data Value, Word Count, Categorized Words, Categorization Status, and Root Category Rule.
- **Comparison** - The **Comparison** drop-down list includes the following options: equal to, not equal to, begins with, ends with, contains, and like.
- **Value** - Enter the value that you want to filter in the **Value** field.
- **Match Case** - Select **Match Case** if you want the condition to include the case used in the **Value** field.
- **Add Condition** - Click **Add Condition** once you have selected options from the Column and Comparison fields and entered a Value. Your new condition appears in the Conditions section.

Cancel Filter - Click **Cancel Filter** when you want to remove a filter from the process.

Search

Find - To locate items in your data, click **Search** > **Find**.

Find Next - Once you have the text string that you want to search for, you can click **Find Next** to locate the next instance of that text.

Go To - Enables you to jump to a specific item in the data.

Options

Refresh Parse Results - Click to refresh the parse results from **Categorization and Rule Building** as you build your parse definition. This option updates **Categorization and Rule Building** with the parse results so that you can see how the parse definition appears as you build the parse definition.

Note: If you make changes directly to the grammar or vocabulary, some of the derived category and rule information can be cleared from **Categorization and Rule Building**.

Set QKB - Opens **Select QKB**. Click to select the QKB that you want to use for this particular Parse Definition and click **OK**.

Note: You can have only one instance of Customize open at a time since both access the Quality Knowledge Base (QKB).

Help

Help Topics - Opens the DataFlux Data Management Studio online Help.

About - Displays contact information and the copyright statement.

Toolbar

The **Parse Definition Quick Editor** toolbar provides shortcuts to some of the commonly used options for the Parse Definition Quick Editor.

- Categorization and Rule Building
- Element Analysis
- Chopper
- Normalization Regexib
- Grammar
- Vocabulary

Categorization and Rule Building

Select the **Categorization and Rule Building** tab to view the results after your new parse definition is applied to your data. The table provides:

- **Word Count** - the number of words in the sentence
- **Categorized Words** - the words matched with words in the vocabulary
- **Categorization Status** - the percentage of words from the total number of words found to have a category in the vocabulary
- **Root Category Rule** - whether a particular sentence matched a specified pattern rule or not

Element Analysis

Element Analysis shows how often each word is listed along with the category, based on the parse definition vocabulary.

Chopper

The **Chopper** tab in Parse Definition Quick Editor provides a similar function as the separate Chop Table Editor. You can edit and create chop tables within the parse definition.

Normalization Regexlib

The **Normalization Regexlib** in the Parse Definition Quick Editor provides similar functionality as the separate Regex Editor. You can edit and create new regex libraries within the parse definition.

Grammar

The **Grammar** tab in the Parse Definition Quick Editor provides the same functionality as the separate Grammar Editor. You can edit and create new grammars within the parse definition.

Vocabulary

The **Vocabulary** tab in the Parse Definition Quick Editor provides similar functionality as the separate Vocabulary Editor. You can edit and create new vocabularies for the parse definition.

New Parse Definition

To begin a new parse definition in the **Parse Definition Quick Editor**, click **File** > **New** to open **New Parse Definition**.

Root Category

A Root Category is where all basic and derived rules begin. Here, you begin creating the root category for your parse definition.

Abbreviation - Enter a grammar abbreviation for the Root Category for your new parse definition.

Name - Enter the name of the Root Category for your new parse definition.

Initial Settings

Chop table - Click the **Chop Table** drop-down list to select an existing chop table.

Normalization regexlib - Click the **Normalization Regexlib** drop-down list to select an existing regexlib.

Select a Data Input Type for the Project

Select the type of data input used in the data job.

Select an Input Table

Select a data input table from the list provided (this list is based on your selection in the previous section).

Refresh - Click **Refresh** to update the **Select Table** list after you make changes using the ODBC Data Source Administrator or SAS Connection Administrator. If you select a data source type that prompts you to select an associated file, and then decide that you would rather select another data source, click **Refresh** to clear the file first selected.

View - Click the table that you want to view and then click **View**. The table opens in DBViewer.

Properties - Click to display the **Table Properties**, when a table is selected in the **Select Table** list. This lists useful information about the selected table, such as field name, source, and field lengths. You can print this information or export it as a .txt, .csv, or .xls file.

ODBC Administrator - Click **Administrator** to specify data locations, connect to data sources, and perform other data set configuration activities. To learn more about this tool, from the **ODBC Data Source Administrator**click **Help**.

SAS Connection Administrator - Click **SAS Connection Administrator** to open. To learn more about the SAS Connection Administrator, refer to the SAS Connection Administrator topic.

Select an Input Field

Select an input field from the **Field Name** drop-down list. This list is based on the fields available from your data source. Your results appear in the Parse Definition Quick Editor.

Click the tabs available under the toolbar to view your data.

Open Parse Definition

To open an existing parse definition in the Parse Definition Quick Editor, perform the following steps:

1. Click **File** > **Open** (or from the **Parse Definition Quick Editor** toolbar, click the **Open** icon to open **Open Parse Definition**.
2. Select the Parse Definition that you want to access and click **Next**.
3. Select the data input type for this Parse Definition. Click **Next**.

4 Select the input table that you want to access for this Parse Definition. Click **Next**.

5 Select the field that you want to access in the Parse Definition. Click **Finish**.

Your Parse Definition opens in the Parse Definition Quick Editor. You are ready to edit the Parse Definition.

Save or Save As

To save your new parse definition in the Parse Definition Quick Editor, click **File** > **Save** or **File** > **Save As**.

> **IMPORTANT** Make sure you complete all steps in the Parse Definition Quick Editor prior to saving your parse definition.

Parse Definition

Name - Enter a Name for your new Parse Definition.

Data type - Click the **Data Type** drop-down list to select the data type for this Parse Definition. The options include: City - State/Province - Postal Code, Name/Organization, Organization, Phone, State, ZIP, Organization (Two Organization), Address, Date, Name, Name (Two Name), Name (Multiple Name), Account Number, Date/Time, Business Title, Country, Address (Global), City, City - State/Province - Postal Code (Global), E-mail, Name (Global), Phone (Global), Text, and Website

New - To create a new data type, click **New**.

- **Name** - Enter a name for the New Data Type.
- **Example** - In the **Example** field, create a meaningful example of the new data type.
- **Tokens**
  - **Add** - Click **Add** to add a token to the list.
  - **Delete** - Click **Delete** to delete a token from the list.
  - **Up** - Click **Move Up** to move a token up in the list.
  - **Down** - Click **Move Down** to move a token down in the list.

> **Note:** If you delete a token, extraction schemes or sought categories that reference the token is removed as well. As a result, you could potentially make a definition invalid by removing a token. Customize marks these items with a red "X".

Default Categories

Category - This is the category name from your data source.

Likelihood - This column represents the likelihood that a word belongs in a particular category. The options include: Very Low, Low, Medium, High, and Very High.

Add - Click **Add** to add a category for the selected word.

Edit - If you have multiple categories for a particular word, select the category that you want to edit, and then click **Edit**.

Delete - Select the category that you want to delete and click **Delete**.

Files

Chop Table - Click **Browse** to select the Chop Table where you want this Parse Definition saved.

Note: The Chop Table selected when creating the new Parse Definition automatically appears in this field.

Normalization Regexlib - Click **Browse** to select the Normalization Regexlib where you want this Parse Definition saved.

Note: The Normalization Regexlib selected when creating the new Parse Definition automatically appears in this field.

Grammar - Click **Browse** to select the Grammar where this new Parse Definition is saved.

Vocabulary - Click **Browse** to select the Vocabulary where this new Parse Definition is saved.

Navigating the Parse Definition Quick Editor

The toolbar provides shortcuts to some of the commonly used options for the Parse Definition Quick Editor.

Categorization and Rule Building

The **Categorization and Rule Building** tab shows the results after the new parse definition is applied to your data. The columns include:

- **Word Count** - the number of words in the sentence
- **Categorized Words** - the words matched with words in the vocabulary
- **Categorization Status** - the percentage of words from the total number of words found to have a category in the vocabulary
- **Root Category Rule** - whether a particular sentence matched a specified pattern rule or not

Word/Category

The Word/Category table displays the parsed entry. If there is a grammar rule related to the word, it is displayed in the table. Each step displays how the final definition arrived at the results from the previous definitions and processed components.

You can click an entry to see the Word/Category at the bottom of the screen.

Word

Right-click the Word.

Adjust Chopping - Use this option to make changes to the way the separates the item.

Category

Right-click the Category.

Assign Basic Category - A basic category is a category representing a single word. Basic categories are the basic building blocks of Grammar rules.

- **Create Category** - Select **Create Category** to add a basic category.

  Click **OK** and you see the new Category assigned. In this example, ABBOTT is assigned the category of LASTNAME.

- **NUM (Number)** - Use this category when you have a number as a part of the data (for example, in an address).

Unassign Basic Category - When you have a basic category that you want to remove, right-click the word and select **Unassign Basic Category** to return to the default category.

Add Rule to Derived Category - Once a category is created, you can select the category and right-click **Add Rule to Derived Category** to create an additional category.

Note: A rule must be associated with the Derived Category prior to saving the parse definition.

Remove Rule from Derived Category - To delete a derived category, right-click on the category to select **Remove Rule from Derived Category**.

Derive Root Category from Rule - Select this option to assign your derived rule to the root rule used for parsing.

Remove Rule from Root Category - Select the rule that you want to remove from the root category, right-click and select **Remove Rule from Root Category**.

Element Analysis

Element Analysis shows how often each word is listed along with the category, based on the parse definition vocabulary. The columns on the **Element Analysis** tab include:

- **Word** - the parsed word
- **Frequency** - the number of times the word appears in the data source
- **Categories** - the category assigned to each word

Note: The Categories column includes categories from all the vocabularies.

Right-click on the entry to make changes to the category:

Assign Basic Category - Click **Assign Basic Category** for the following drop-down list options:

- **Create Category** - Select **Create Category** to add a basic category.
- **NUM (Number)** - Use this category when you have a number as a part of the data (for example, Address).

Unassign Basic Category - When you have a basic category that you want to remove, right-click the word and select **Unassign Basic Category** to return to the default category.

Chopper

The **Chopper** tab in Parse Definition Quick Editor provides a similar function as the separate Chop Table Editor on page 19. You can edit and create chop tables within the parse definition.

The following unique menu items are available in the **Edit** menu of this tab:

Disable Table - This option enables you to edit and create chop tables within the parse definition.

Disable Rules - This option enables you to update the Chop Rules. See Rules on page 20.

Implicit Separation - Implicit separators are an optional feature of string chopping. For each parse definition, you can enable or disable implicit separators. If implicit separators are enabled, a separator mark is placed wherever the classification of a character differs from the classification of the previous character in a string. See Character Level Options on page 24.

The **Chopper** tab includes the following elements:

- **Chopper used in definition** - Specifies the name of a single chopper that is used by the parse definition. When a new definition is created, a chopper with a placeholder name New Chopper is listed here. You can change the chopper library file or edit the chopper using the buttons next to the field. When you choose a different chopper, you are prompted to either save or discard the existing definition. Then, you can either select an existing chopper with a drop-down menu or create a new chopper in the **Chopper** tab.

  The Table sub-tab includes the following elements:

  - **Unicode block** - the Unicode block drop-down list provides a list of character subsets, see Unicode Block for the complete list.
  - **Character Name** - the Character Name is the actual name of the character, for example, semicolon
  - **Character** - the Character represents the actual appearance of the character for example, the Character Name is semicolon and Character is ;
  - **Classification** - the Classification drop-down list includes:
    - **LETTER/SYMBOL** - a letter or non-separating symbol
    - **NUMBER** - a numeric digit (0-9)
    - **LEAD SEPARATOR** - a delimiter attached to the beginning of a word (for example, the left parenthesis)
    - **TRAIL SEPARATOR** - a delimiter attached at the end of a word (for example, a period)
    - **FULL SEPARATOR** - a delimiting character (for example, space, hyphen, and comma)
- **Operation** - the **Operation** drop-down list includes:
  - **USE** - use the character as-is in the word list and output tokens
  - **TRIM** - omit from word list; trim leading and trailing characters in output tokens
  - **SUPPRESS** - omit from the word list and output tokens
- **Value** - each character is assigned a value
- **Hex Value** - this field represents the character code for the character on that line

The **Rules** sub-tab uses a rules-based chopping algorithm that works by matching portions of an input string from left to right using search criteria and states. These are specified by rules, which can be defined from the **Rules** tab. These rules are processed from top to bottom and the search criteria includes:

- A vocabulary of words
- A single regular expression

The system uses the search criteria to first match the input string at the current position. If it fails, it proceeds to the next rule and attempts to match using the criteria for that rule, and so on. If no rules match, the input string position is advanced by one character and the algorithm run again. This process is repeated until the process reaches the end of the input string.

If the system is able to successfully match a substring, it then attempts to validate the state. At any point in time, the system maintains a state of zero or more flags that are just variables that either exist or not. Each rule has the ability to check for the existence of flags in the current state using a simple Boolean syntax. This is called the Prerequisite State Condition. If this validation fails, the rule fails to match and the next rule is checked, and so on. If it succeeds, then the following occurs:

- The input string is advanced to the end of the successful match in the Search criterion.
- A new state (Output State) consisting of zero or more flags is set. The new state replaces the old state.
- The string is chopped before the current match, after the current match, at both points, or not at all.

This part of the **Rules** tab sets up an initial state before any rule processing is performed. This is useful for identifying a pre-existing state in order to force certain rules first.

The **Initial Flags** list has two controls. Click **Add** to open **Add New Flag**. Here, you enter the name of the new initial flag. As you create new flags, it is added alphabetically to the list. To delete an existing flag, select the flag and click **Delete**.

The Rules sub-tab includes the following elements:

- **Initial Flags** - Displays the initial flags associated with the chopper definition. You can use the neighboring buttons to delete and rearrange the flags.
- **Method** - The Method is either Vocab or Regex, depending on the type of criterion for the rule. This determines the format for the Search Criterion column.
- **Regular Expression** - This field enables you to enter the regex directly into the accompanying field.
- **Vocabulary** - The **Vocabulary** drop-down list enables you to select one of the available vocabularies.
- **Category** - The Category drop-down list displays all of the possible categories for the selected vocabulary.
- **Prerequisite State Condition** - This is a Boolean text expression.
- **Output Flags** - The **Output Flags** list can be populated with flag names just like **Initial Flags**.
- **Chop Mode** - The **Chop Mode** drop-down list enables you to select one of the possible modes.

- **Notes** - The **Notes** field enables you to add 128 characters of text. This is used for documentation purposes.

Normalization Regextlib

The Normalization Regexlib in the **Parse Definition Quick Editor** provides similar functionality as the separate Regex Editor. You can edit and create new regex libraries within the parse definition.

Note: Normalization Regexlib in the **Parse Definition Quick Editor** does not provide the ability to test your regex libraries like the Regex Editor.

The tab includes the following elements:

Normalization regexlibs used in definition - Displays the normalization regexlib used in the parse definition (0 or more). When an existing definition is loaded, all that definition's normalization regexlibs appear in the list view field in logical order, starting at the top. When a new definition is created, no normalization regexlibs exist yet, so the field is empty. Zero or one row of the field can be selected at a time.

Add Expression - To add a new expression to the , right-click on one of the expressions and select **Add Expression**.

- **Regular Expression** - A regular expression is a pattern matched against a subject string from left to right. Most characters stand for themselves in a pattern and match the corresponding characters in the subject. For more information about Regular Expressions and Substitution, refer to Using Regular Expressions.
- **Substitution** - A substitution is a string that is inserted into the output of a regular expression call in place of the pattern matched by the regular expression. For more information about Regular Expressions and Substitution, refer to Using Regular Expressions.
- **Notes** - Use the Notes section to create a note related to the selected expression.

Edit Expression - You can edit an existing expression, right-click on the expression that you want to edit and select **Edit Expression**. Make your changes and click **OK**.

Delete Expression - To delete an existing expression, select the expression, right-click and click **Delete Expression**.

Move Expression - Move a selected expression up or down in the list.

Save Expression - Saves the expression.

Grammar

The **Grammar** tab in the **Parse Definition Quick Editor** provides the same functionality as the separate **Customize Grammar Editor**. You can edit and create new grammars within the parse definition.

Grammar used in definition - Specifies the grammars used for the definition. You can add grammars and change, edit, delete, and reorder the existing grammars by using the buttons next to the field. When you choose a different grammar, you are prompted to either save or discard the existing definition. Then, you can either select an existing grammar with a drop-down menu or create a new grammar in the **Grammar** tab.

The left navigation section displays the **Basic Categories** and **Derived Categories** set in earlier steps of the Parse Definition Quick Editor. Right-click on any of the items in this section and the following options are available:

- **Add Category** - Click **Add Category** to add a Basic or Derived Category to the selected item.
- **Change Category Abbreviation** - Click **Change Category Abbreviation** for the item selected.
- **Add Rule** - Select the item where you want to add a rule, right-click and select **Add Rule**. The Rule Properties on page 50 section on the right is where you create your rule.
- **Delete Selected Item** - To delete an item, select the item from the list, right-click and select **Delete Selected Item**.

Category Properties

- **Abbreviation** - The abbreviation for the selected category.
- **Name** - The name of the selected category.
- **Type** - Select **Basic** or **Derived**.

Rule Properties

- **Parent category** - The **Parent** category is the basic or derived category the rule is related to.
- **Priority** - Click the **Priority** drop-down list for the priority options available. The options include: Very Low, Low, Medium, High, or Very High.
- **Category** - The **Category** field shows you a list of categories related to the rule. You can add, edit, and remove categories in this section.
  - **Add** - Click **Add** to add another category to your rule.
  - **Edit** - Select the item that you want to edit and click **Edit**.
  - **Remove** - Select the rule category that you want to remove from the rule.

Vocabulary

The **Vocabulary** tab in the Parse Definition Quick Editor provides similar functionality as the separate Vocabulary Editor. You can edit and create new vocabularies for the parse definition.

Note: The Vocabulary option in the Parse Definition Quick Editor does not provide the option to import vocabularies like the Vocabulary Editor.

Vocabularies used in definition - Specifies the one or vocabularies that are used in the parse definition. When an existing definition is loaded, all of that definition's vocabularies appear in the list in logical order, starting at the top. When a new definition is created, it contains one vocabulary with the placeholder name "New Vocabulary." Note that there must always be at least one vocabulary present (unlike regexlibs). Note that you can add vocabularies and change, edit, delete, and reorder the existing vocabularies by using the buttons next to the field.

Editing the selected vocabulary - Displays the words contained in whichever vocabulary is currently selected in the **Vocabularies used in definition** field. If there are no items in the **Vocabularies used in definition** field or none are selected, the Word ListView is empty. The words are arranged alphabetically.

Word - The word that you have selected in the **Editing the selected vocabulary** field appears in the **Word** field.

Word Properties - Categories

Category - This is the category name for the word selected from your data source.

Likelihood - This column represents the likelihood that a word belongs in a particular category. The options include: Very Low, Low, Medium, High, and Very High.

Add - Click **Add** to add a category for the selected word.

Edit - If you have multiple categories for a particular word, select the category that you want to edit, and then click **Edit**.

Delete - Select the category that you want to delete and click **Delete**.

Note: Because you can add any existing vocabularies to the definition, it is possible that an existing vocabulary can contain categories that are not contained in the definition's grammar. These categories are retained when the vocabulary is saved, but they are ignored by the definition's processing unless they are added to the grammar. You cannot add or edit categories that are not defined in the grammar.

Phonetics Editor

DataFlux Data Management Studio software uses phonetic analysis (Phonetics) during the process of generating match codes. Phonetics processing involves applying a set of Phonetics rules to an input string. The goal is to create Phonetics rules that produce the same output string for input strings that have similar pronunciations or spellings. For example, the following two strings are phonetically equivalent in English.

- "SCHMIDT"
- "SHMITT"

Phonetics rules would reduce both of these to the string "SHMIT." To create Phonetics rules, use the Customize Phonetics Editor.

Creating Phonetics Files

1. Open the Phonetics Editor. On the Customize main screen, select **Tools** > **Phonetics Editor**.
2. Set Your Locale. Select **Options** > **Set Locale**.
3. Select the appropriate locale and click **OK**. The locale setting is saved from session to session, so you do not need to specify it again unless you need to build a Phonetics file for a different locale.
4. Create a New Phonetics File. Select **File** > **New**.
5. Create Phonetics Rules. Select **Edit** > **Add Rule**.
6. Specify **Rule Text** and **Replacement Text**, and then click **OK**.
7. Set Priority and any flags. (For more information about Phonetics rules, see Components of a Rule on page 53).

   To add another rule, select **Edit** > **Add Rule** again. Be aware that the new row is added after the current row. This enables you to place the new rule in the desired location sequentially within the library. Repeat this process until you have defined all desired rules.
8. Save Your Phonetics File. Select **File** > **Save**. Because this is a new Phonetics file, the Phonetics Editor prompts you for a file name.
9. Test Your Phonetics File. At the bottom of the Phonetics Editor is a Test Area. This area enables you to supply sample input strings to test your Phonetics rules. Enter an input string and observe the result. If the result is not what you intended, you can modify your rules and re-test. The Test Area also displays the previous string and result so that you can make comparisons.

Components of a Rule

A phonetics rule specifies a pattern that you want to locate, the replacement text (if any), and the action that you want taken after the replacement is made.

Rule Test

Rule text identifies the pattern that you want to locate. Rule text can consist of literal characters and a small set of meta-characters.

- **Dot meta-character** (".") - A single period (dot) in the rule text matches any single character in the input string. For example, the rule of "CA." would match any phrase that has a "C" followed by an "A" followed by any other character.
- **Character class ("[" and "]")** - Enclose a set of literal characters in square brackets to match any one of the specified characters. For example, the rule text "CA[TB]" would match "CAT" and "CAB," but would not match "CAP."

  If the first character within a character class is a circumflex ("^"), it matches only characters that are not listed in the character class. For example, the rule text "CA[^TB]" would not match "CAT" and "CAB," but it would match "CAP," "CAN," or any other phrase that has a "CA" followed by any character except a "T" or "B."
- **Beginning of word ("^")** - Use a circumflex to indicate that the following pattern must be found at the beginning of a word. For example, the rule text "^SAND" would match "SANDWICH," but not "STREISAND."
- **End of word ("$")** - Use a dollar sign to indicate that the preceding pattern must be found at the end of a word. For example, the rule text "SAND$" would match "STREISAND" as well as the word "SAND" (because it is at the end of the word) but not "SANDWICH."
- **Replace the first n characters ("/")** - Use an embedded forward slash to search on an entire pattern but then replace only the characters prior to the slash. For example, the rule text "SCH/OOL" with the replacement text "SK" matches the word "SCHOOL" and produce an output string of "SKOOL."

  This differs from the rule "SCH," which matches the pattern "SCH" anywhere within a string. It is also different from "^SCH," which matches the pattern "SCH" if it is at the beginning of any word, regardless of what follows it.
- **Literal characters** - Specify all literal characters in uppercase. If you want to use a literal that is also used as a meta-character (the dollar sign, period (dot), circumflex, forward slash, backslash, or square bracket), escape that character by preceding it with a backslash. For example, to look for a dollar sign, use the rule text "\$." To match a backslash, use "\\."
- **To match against a space** - You cannot match against a space unless the space is the only thing in the rule. For example, to replace a space with an underscore,

use the rule text "\ " with the replacement text "_." Other than this one purpose, you should not use white space in Phonetics rules.

Replacement Text

Replacement text replaces the entire pattern found by the rule text, except in the case of a forward slash. If you use the forward slash in your rule text, the replacement text replaces the pattern up to the slash.

Replacement text can contain only literal text, no meta-characters.

Priority

There are two factors that determine the order in which Phonetics rules are processed: priority and order.

Phonetics rules are processed in decreasing priority. In the case of multiple rules with the same priority, rules are processed in the order in which they appear (top to bottom). Priority must be a numeric value greater than 0 and less than 100.

For more information, see Changing Rule Order on page 55.

Reset flag

When you select the Reset flag, the character following the replacement is considered the beginning of a word, potentially allowing it to match rules using the "^" meta-character.

For example, for the input string of "MCKNIGHT", if the rule text is "MC" and the **Reset flag** is selected, then after any replacement has been made, the "K" is considered the beginning of a word. This means that "KN" is replaced by "N" due to the "KN" rule.

Rewind flag

Phonetics maintains a search pointer that references a position in the input string. When processing begins, the search pointer points to the beginning of the string. When a Phonetics rule is applied to the string, the search pointer moves forward through the string to point to the position just after the text that was just replaced.

For example, suppose the input string is "ISAACS", and your Phonetics library has rule "C" with replacement text "K". After this rule is applied, the string would look like "ISAAKS," and the search pointer would point to the letter "S."

But suppose you also have the rule "KS" and the replacement string "X". If the search pointer points to the letter "S" in "ISAAKS," there is no way to match the text "KS" and make the replacement. This is where the Rewind flag is helpful. If you use the rewind flag on the first rule, then after "C" is replaced with "K," the search pointer will be rewound to point to the letter "K" in "ISAAKS," Now your rule "KS" will be applied, and the string will be changed to "ISAAX."

Ignore Replace flag

If you select the **Ignore Replace flag**, when a rule is matched, DataFlux Data Management Studio does not make any replacements, even if replacement text is

specified. DataFlux Data Management Studio continues processing rules from the end of the matched pattern forward.

This is useful when you want to make sure that a specific pattern is never replaced by other Phonetics rules. For example, suppose you have a rule "Y" with replacement text "I," but you do not want to change "Y" to "I" if the "Y" is followed by another vowel. You could use a rule such as "Y[AEIOU]" and set the Ignore Replace flag. As long as this rule has a higher priority than the first rule, the "Y" is preserved when it is followed by another vowel. There is no need to write any replacement text for the "Y[AEIOU]" rule; the replacement text field is ignored when the Ignore Replace flag is set.

Note: It does not make sense to use the Ignore Replace flag and the Rewind flag in the same rule. These two flags are mutually exclusive.

Changing Rule Order

After using the Phonetics Editor's Test Area to test your Phonetics rules, you might see some unintended effects because of the order in which the rules are processed. If this happens, you can alter the order in which the rules are processed to change the effects.

Two factors control the order in which Phonetics rules are processed. Highest precedence is given to Priority, so you can simply increase the priority to process the rule earlier, or decrease the priority to process the rule later.

Within a certain priority, rules are processed in the order in which they appear in the Phonetics Editor (top to bottom). To reorder:

1. In the Phonetics Editor, select the rule or row that you want to move.
2. Drag the row to the desired location, and then release the mouse button. The row appears in its new position.

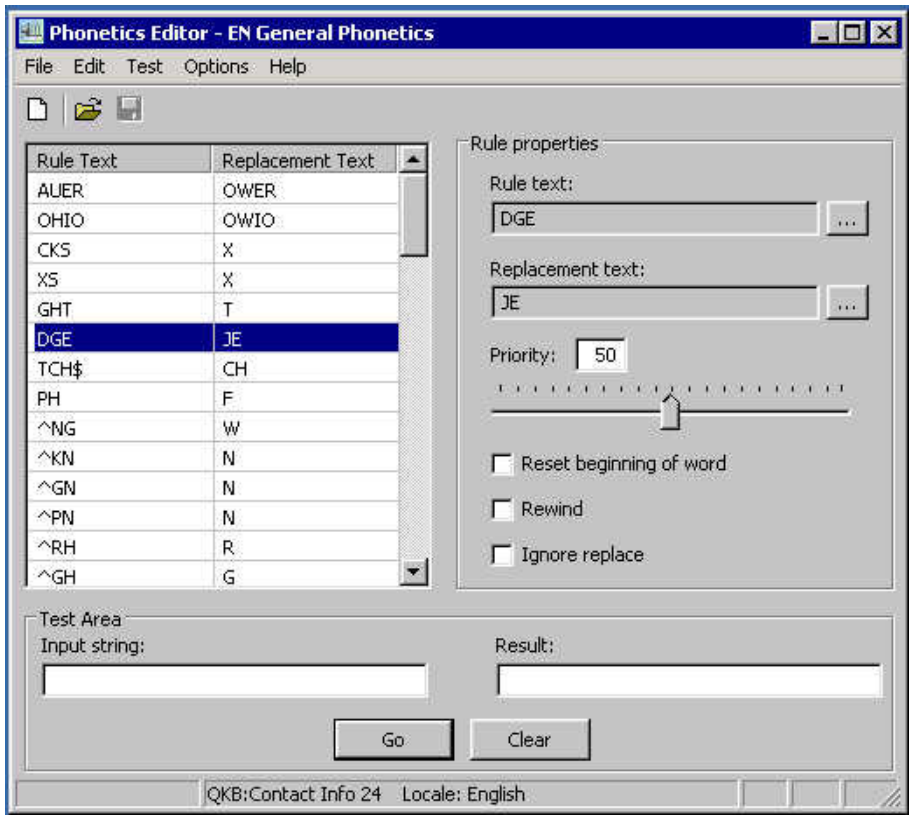
Navigating Phonetics Editor

Overview

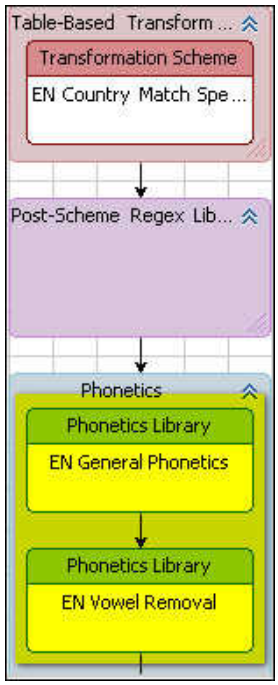
Phonetic reduction, when combined with other methods, can identify table records, which differ only because different spellings were mistakenly used for the same person or entity. A phonetic library is a type of QKB file that is used in phonetic reduction. Each phonetic library contains rules, which facilitate the matching of words with similar pronunciations or spellings. Based on the settings in the library,

silent or unnecessary letters are eliminated. Complex groups of letters are changed to simpler spellings that represent similar sounds.

For example, the next display shows some of the rules listed in the EN General Phonetics library in the English locale in SAS Quality Knowledge Base for Contact Information.



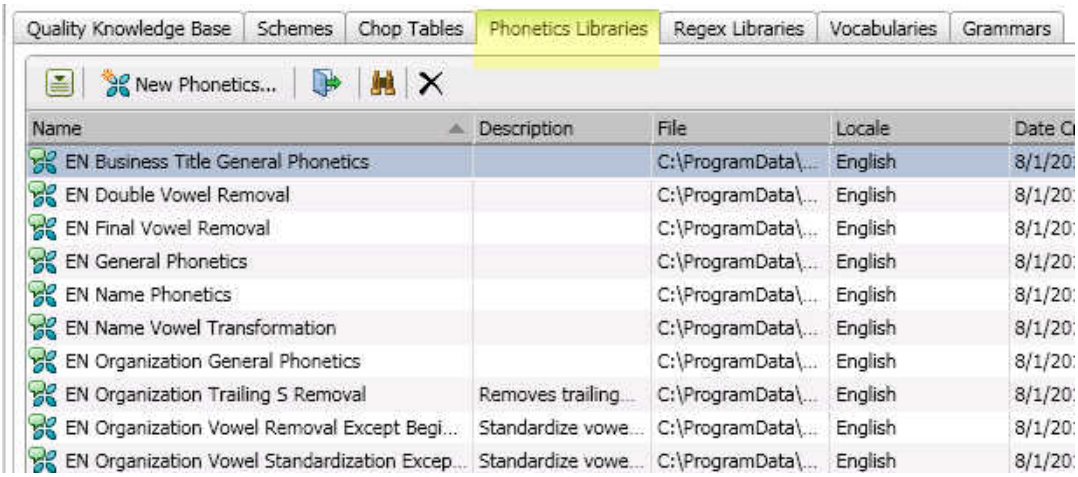
In the previous display, the string DGE is simplified to JE. This information can be used in subsequent nodes in a QKB definition. The next display shows the EN General Phonetics library in the context of the Country match definition in the English locale.



Phonetics Editor Tab

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right-hand side of the screen. The **Phonetics Libraries** tab shows a list of phonetic libraries that are contained in the QKB.

The list of libraries in the **Phonetics Libraries** tab has several columns, as shown in the next display.



The Name column shows the name of a phonetic library. The Description column contains text about the library. The File column contains the physical path to the library file.

The Locale column shows the QKB locale with which the library is associated. Note that a phonetic library might be listed even if it is not associated with the locale that you have selected in the Quality Knowledge Bases tree. This is because locales can inherit phonetic libraries from ancestors on page 163.

Select a locale and look at the values in the Locale column for a phonetic library. You see that some libraries are associated with the selected locale. Others are associated with the locale language, and still others are associated with the Global set of objects. This information is also presented in the icons that are shown for the libraries in the list. As shown in the previous display, if an icon has a flag overlay, the library is associated directly with a locale. If the icon has a word balloon overlay, it means that the library is associated with a language. An earth overlay means that the library belongs to the Global set of phonetic libraries.

If you want to view only the phonetic libraries in the Global set, click **Global** in the tree for your QKB.

Find a Phonetic Library by Name

To find a phonetic library with a specific name, click **Find** in the lower toolbar. This toggles on the **Find** pane. Click **Find** again to toggle off of **Find**.

Rename and Duplicate Phonetic Libraries

You can rename a phonetic library by right-clicking the name and selecting Rename from the pop-up menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a phonetic library by right-clicking the name and selecting **Duplicate** from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Phonetic Library

To create a new phonetic library, click **New Phonetics Library** on the lower toolbar, or select **New Phonetics Library** from the menu. After you have created the new library, it appears in the phonetic library list, and the Phonetics Editor on page 52 appears.

Open a Phonetic Library for Editing

To open a phonetic library for editing, double-click on the library in the list, or select the library and click **Open** on the lower toolbar. You might also select the library and choose **Open** in the pop-up menu or in the menu on the lower toolbar. A

new tab opens that displays the flow diagram for the selected library. See Phonetics Editor on page 52 for information about how to edit a phonetic library.

View Definitions That Use a Given Phonetic Library

To view a list of QKB definitions that use a given phonetic library, select the library and choose **Usage** in the pop-up menu or in the menu on the lower toolbar. **Usage** displays a list of the definitions that use the selected library.

Delete a Phonetic Library

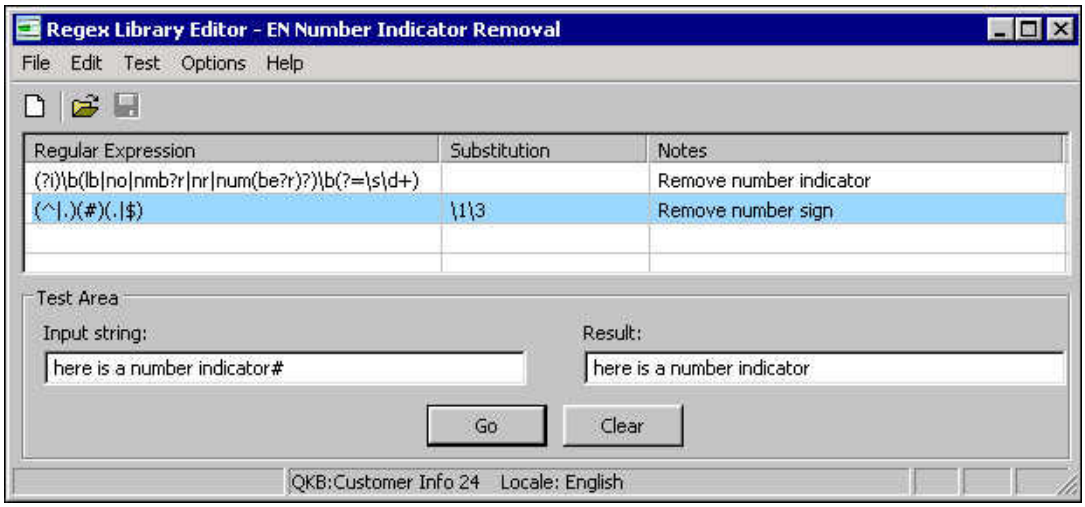
To delete a phonetic library, select the library and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu or in the menu in the lower toolbar. The library is removed from the QKB.

Regex Library Editor

Overview

Use the Regex Library Editor to maintain regular expression libraries (regex libraries). A regular expression is a pattern that is matched against a subject string from left to right. A regex library is a type of QKB file that contains regular expressions that follow Perl syntax. These expressions enable you to express how a pattern in a string should be detected and, optionally, replaced.

The next display shows the regular expressions in the EN Number Indicator Removal regex library in the English locale in the SAS Quality Knowledge Base for Contact Information 24.



Use the menu or the toolbar to add new libraries, open existing libraries, or maintain the regular expressions in a library. The expression table displays any expressions in the current library. The Test Area enables you to see the results of applying the regex library to a text string. For example, in the previous display, the expression highlighted in light blue matched the Input String in the Test Area. The **Result** field shows how the input string was changed by the matching expression. The current locale appears in the status bar.

Menu

File

New	Enables you to create a new library. When you create a new library, you must specify a locale with which to associate the library. After you have specified a locale, you will be prompted to add at least one expression to the new library. For more information about adding a new library, see Adding and Editing Regex Libraries.
Open	Enables you to open a regex library in the current locale.
Close, Save, Save As, Exit	Enable you to perform these operations on the library in the editor.

Edit

These options enable you to maintain the regular expressions in a library. The options are inactive until you open a library.

Add Expression	Enables you to specify a new expression. Use Perl syntax for the expression and any substitution options. For more information about expression syntax, see Regular Expression Syntax in the Regex Library Editor.
Edit, Delete, Copy, Paste Expression	Enable you to perform these options on a selected expression.

Test

Go	Applies a selected expression to a string in the **Input string** field in the Test Area.
Clear	Removes current text from the Test Area.

Options

Set QKB	Displays **Select Locale** so that you can specify the locale where you want to add or maintain a QKB library.

Toolbar

Toolbar buttons enable you to create, open, or save a regex library.

Expression Table

The expression table displays any expressions in the current library.

Regular Expression	Displays a regular expression in Perl syntax. If you select an expression, it is highlighted in dark blue. If an expression matches an input string in the Test Area, the expression is applied to the string, and the expression is highlighted in light blue. If multiple expressions match an input string in the Test Area, they are all highlighted in light blue.
Substitution	Displays a string that replaces the portion of the input string that is matched by the expression, if any. Note that as in Perl, a substitution string can contain back references.

Notes	Displays any notes about the expression.

For more information about maintaining expressions, see Adding and Editing Regex Libraries.

Test Area

The Test Area enables you to see the results of applying the regex library to a text string. Specify an input string and click **Go** to see the result.

Input string	Enables you to enter a string that tests the regex library.
Result	Displays the result of applying the regex Library to the input string. All expressions that matched the string and were applied to the sting are highlighted in light blue in the expression table.
Go	Applies the expressions in the regex library to a string in the **Input string** field in the Test Area.
Clear	Removes current text from the Test Area.

Adding and Editing Regex Libraries

Add a new regex library or update an existing library when the current libraries do not produce the output that you desire. To understand regex libraries, see the help for the Regex Library Editor on page 59. See also the existing libraries on the **Regex Library** tab. For details about expression syntax, see Regular Expression Syntax in the Regex Library Editor on page 64.

Regular expressions are applied sequentially in the order in which they appear in the regex library. It is possible that a value could be modified by one regular expression. Then the result of that modification could be modified again by a regular expression that appears farther down in the library. This could occur several times as DataFlux Data Management Studio applies each regular expression from top to bottom. Be careful not to subvert the effects of one regular expression with another.

After creating or updating a regex library, you can use the Regex Library Editor on page 59 to test your expressions. The Test Area at the bottom of the editor enables you to enter sample input strings to verify that you have written your regular expressions correctly. Enter an input string and observe the result. If the result is not what you intended, you can modify your regular expressions or their order and re-test. When you are satisfied with the results, save your changes.

Add a New Regex Library

Perform these steps to add a new regex library to a locale in a QKB. It is assumed that you have opened the QKB from the **Administration Riser**, and the **Regex Libraries** tab is displayed.

1. Click **New Regex Library** on the lower toolbar, or select **New Regex Library** from the menu on the lower toolbar. The **New Regex Library** wizard opens.
2. Enter a name and description for the new library and click **Next**.
3. Select a locale for the library and click **Finish**. The new library appears in the regex library list, and the **Regex Library Editor** opens.
4. Add one or more expressions to the library.
   a. To add a new expression, select **Edit** > **Add Expression**.
   b. To edit, delete, or copy an expression in the editor, select an expression. Then select the appropriate option from the **Edit** menu. To paste a copied expression, select **Edit** > **Paste Expression**.
   c. To reorder expressions in the editor, select an expression and drag it to a new location.
5. Use the Test Area of the editor to see the results of applying all expressions in the current regex library to a text string. Specify an input string and click **Go** to see the result.
6. When you are satisfied with the results of your testing, save the library. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.
7. Click **Yes** to reload the QKB.

Edit a Regex Library

It is usually best to copy an existing regex library and modify the copy, rather than modify the original. If you modify an existing library, you might break a QKB definition that uses that library.

Perform these steps to edit a regex library in a locale in a QKB. It is assumed that you have opened the QKB and locale from the **Administration Riser**, and the **Regex Libraries** tab is displayed.

1. To open a regex library for editing, double-click on the library in the list, or select the library and click **Open** on the lower toolbar. You might also select the library and choose **Open** in the pop-up menu or in the menu on the lower toolbar. The library opens in the **Regex Library Editor**.

2 To add a new expression, select **Edit** > **Add Expression**

   a To edit, delete, or copy an expression in the editor, select an expression. Then select the appropriate option from the **Edit** menu. To paste a copied expression, select **Edit** > **Paste Expression**.

   b To reorder expressions in the editor, select an expression and drag it to a new location.

3 Use the Test Area of the editor to see the results of applying all expressions in the current regex library to a text string. Specify an input string and click **Go** to see the result.

4 When you are satisfied with the results of your testing, save the library. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.

5 Click **Yes** to reload the QKB.

Regular Expression Syntax in the Regex Library Editor

Introduction

The Regex Library Editor supports Perl-compatible syntax for regular expressions. A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject. For example, the following pattern matches a portion of a subject string identical to itself.

```
The quick brown fox
```

The power of regular expressions lies in their ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of meta-characters, which do not stand for themselves but instead are interpreted in some special way.

A substitution is a string inserted into the output of a regular expression call in place of the pattern matched by the regular expression. To further the example, suppose the following regular expression and substitution are applied to the input string:

Regular Expression	Substitution
brown	blue
Results: The quick blue fox	

In a substitution string, all characters stand for themselves, except the backslash-escape digit is interpreted as a back reference to a captured subpattern. See information about capturing subpatterns with regular expressions. Also, casing operators \U, \u, \L, \l, and \E can be used in substitution strings just like Perl substitutions.

Regular expressions that you create in the Regex Library Editor must comply with the syntax defined for Perl regular expressions. The sections below describe some typical syntax for expressions in a SAS regex library. For a description of the differences in the ways that SAS and Perl handle regular expressions, see Differences From Perl. For general information about writing Perl regular expressions, see "Mastering Regular Expressions" by Jeffrey E.F. Friedl or other readily available reference material on the subject.

Meta-Characters

There are two different sets of meta-characters: those recognized anywhere in the pattern except within square brackets ([ ]), and those recognized in square brackets. Outside square brackets, the meta-characters are:

Character	Description
\	General escape character with several uses
^	Assert start of subject (or line, in multi-line mode)
$	Assert end of subject (or line, in multi-line mode)
.	Match any character except new line (by default)
[	Start character class definition
|	Start of alternative branch
(	Start subpattern
)	End subpattern
?	Extend the meaning of (; 0 or 1 quantifier; quantifier minimizer
*	0 or more quantifier
+	1 or more quantifier; also possessive quantifier
{	Start minimum or maximum quantifier

The part of a pattern in square brackets is called a character class. In a character class, the only meta-characters are:

Character	Description
\	General escape character
^	Negate the class, but only the first character
-	Indicate character range
[	POSIX character class (only if followed by POSIX syntax)
]	Terminate the character class

The following sections describe the use of each of the meta-characters.

Backslash

The backslash character (\) has several uses. First, if it is followed by a non-alphanumeric character, it takes away any special meaning the character has. This use of backslash as an escape character applies both inside and outside character classes. For example, if you want to match a * character, you write * in the pattern. This applies whether the following character is interpreted as a meta-character, so it is always safe to precede a non-alphanumeric character with \ to specify that it stands for itself. For example, if you want to match a backslash, type \\.

If you want to remove the special meaning from a sequence of characters, you can do so by putting them between \Q and \E. This is different from Perl in that $ and @ are handled as literals in \Q...\E sequences in SAS, whereas in Perl, $ and @ cause variable interpolation. Note the following examples.

Pattern	SAS Matches	Perl Matches
\Qabc$xyz\E	abc$xyz	abc followed by the contents of $xyz
\Qabc\$xyz\E	abc\$xyz	abc\$xyz
\Qabc\E\$\Qxyz\E	abc$xyz	abc$xyz

The \Q...\E sequence is recognized both inside and outside character classes.

Non-printing Characters

Second, the backslash provides a way of encoding non-printing characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern. However,

when you are preparing a pattern by text editing, it is usually easier to use one of the following escape sequences than the binary character that it represents:

Character	Description
\a	Alarm; that is, the BEL character (hex 07)
\cx	Control-x, where x is any character
\e	Escape (hex 1B)
\f	Form feed (hex 0C)
\n	New line (hex 0A)
\r	Carriage return (hex 0D)
\t	Tab (hex 09)
\ddd	Character with octal code ddd, or back reference
\xhh	Character with hex code HH
\x{hhh..}	Character with hex code hhh.

The precise effect of \cx is as follows: if x is a lowercase character, it is converted to uppercase. Then, bit 6 of the character (hex 40) is inverted. Thus, \cz becomes hex 1A, but \c{ becomes hex 3B, while \c; becomes hex 7B.

After \x, up to two hexadecimal characters are read. These digits can be in upper or lowercase. Any number of hexadecimal characters can appear between \x{ and }. However, the value of the character code must be less than 2**31 (that is, the maximum hexadecimal value is 7FFFFFFF). If characters other than hexadecimal characters appear between \x{ and }, or if there is no terminating }, this form of escape is not recognized. Instead, the initial \x is interpreted as a basic hexadecimal escape, with no following digits, giving a character whose value is zero.

Characters whose value is less than 256 can be defined by either of the two syntaxes for \x. There is no difference in the way they are handled. For example, \xdc is exactly the same as \x{dc}.

After \0 up to two further octal digits are read. If there are fewer than two digits, just those that are present are used. Thus, the sequence \0\x\07 specifies two binary zeros followed by a BEL character (code value 7). Make sure you supply two digits after the initial zero if the pattern character that follows is itself an octal digit.

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, SAS reads it and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing a left parentheses in the expression, the entire sequence is taken as a back reference. See a description of how this works under Back Reference.

Inside a character class, or if the decimal number is greater than 9 and there have not been many capturing subpatterns, Data Quality re-reads up to three octal digits following the backslash. It then generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves.

Character	Description
\040	Another way of writing a space
\40	The same, provided there are fewer than 40 previous capturing subpatterns
\7	Always a back reference
\11	Might be a back reference, or another way of writing a tab
\011	Always a tab
\0113	A tab followed by the character 3
\113	The character with octal code 113, because there can be no more than 99 back references
\377	A byte consisting entirely of 1 bit
\81	Either a back reference or a binary zero followed by the two characters, 8 and 1

Note: You must not introduce octal values of 100 or greater because no more than three octal digits are ever read.

You can use all the sequences that define a single-byte value both inside and outside character classes. In addition, inside a character class, the sequence \b is interpreted as the backspace character (hex 08). The sequence \x is interpreted as the character X. Outside a character class, these sequences have different meanings, see Generic Character Types.

Generic Character Types

The third use of backslash is to specify generic character types:

Character	Description
\d	Any decimal digit
\D	Any character that is not a decimal digit

Character	Description
\s	Any whitespace character
\S	Any character that is not a whitespace character
\w	Any word character
\W	Any non-word character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

These character type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

For compatibility with Perl, \s does not match the VT character (code 11). This makes it different from the POSIX space class. The \s characters are HT (9), LF (10), FF (12), CR (13), and space (32). (If "use locale;" is included in a Perl script, \s can match the VT character. In SAS, it never does.)

A "word" character is an underscore or any character that is a letter or digit. The definition of letters and digits is controlled by the SAS Unicode character tables.

Unicode Character Properties

Three additional escape sequences to match Unicode character properties are available. They are:

Character	Description
\p{xx}	A character with the xx property
\P{xx}	A character without the xx property
\X	An extended Unicode sequence

The property names represented by xx above are limited to the Unicode script names, the general category properties, and "Any", which matches any character (including newline). Other properties such as "InMusicalSymbols" are not currently supported by SAS. Note that \P{Any} does not match any characters, so always causes a match failure.

Sets of Unicode characters are defined as belonging to certain scripts. A character from one of these sets can be matched using a script name. For example:

```
\p{Greek}
```

```
\P{Han}
```

Those that are not part of an identified script are lumped together as "Common". The current list of scripts includes:

Arabic, Armenian, Bengali, Bopomofo, Braille, Buginese, Buhid, Canadian_Aboriginal, Cherokee, Common, Coptic, Cypriot, Cyrillic, Deseret, Devanagari, Ethiopic, Georgian, Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul, Hanunoo, Hebrew, Hiragana, Inherited, Kannada, Katakana, Kharoshthi, Khmer, Lao, Latin, Limbu, Linear_B, Malayalam, Mongolian, Myanmar, New_Tai_Lue, Ogham, Old_Italic, Old_Persian, Oriya, Osmanya, Runic, Shavian, Sinhala, Syloti_Nagri, Syriac, Tagalog, Tagbanwa, Tai_Le, Tamil, Telugu, Thaana, Thai, Tibetan, Tifinagh, Ugaritic, and Yi

Each character has exactly one general category property, specified by a two-letter abbreviation. For compatibility with Perl, negation can be specified by including a circumflex between the opening brace and the property name. For example, \p{^Lu} is the same as \P{Lu}.

If only one letter is specified with \p or \P, it includes all the general category properties starting with that letter. In this case, in the absence of negation, the braces in the escape sequence are optional; these two examples have the same effect:

```
\p{L}
\pL
```

These general category property codes are supported:

Property Code	Description
C	Other
Cc	Control
Cf	Format
Cn	Unassigned
Co	Private use
Cs	Surrogate
L	Letter
Ll	Lowercase letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter

Property Code	Description
Lu	Uppercase letter
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
Nl	Letter number
No	Other number
P	Punctuation
Pc	Connector punctuation
Pd	Hyphen punctuation
Pe	Close punctuation
Pf	Final punctuation
Pi	Initial punctuation
Po	Other punctuation
Ps	Open punctuation
S	Symbol
Sc	Currency symbol
Sk	Modifier symbol
Sm	Mathematical symbol
So	Other symbol
Z	Separator

Property Code	Description
Zl	Line separator
Zp	Paragraph separator
Zs	Space separator

The special property **L&** is also supported; it matches a character that has the Lu, Ll, or Lt property (a letter that is not classified as a modifier or "other").

The long synonyms for these properties that Perl supports (such as, **\p{Letter}**) are not supported by SAS, and it is not permitted to prefix any of these properties with **Is**.

No character that is in the Unicode table has the **Cn** (unassigned) property. Instead, this property is assumed for any code point that is not in the Unicode table.

Specifying caseless matching does not affect these escape sequences. For example, **\p{Lu}** always matches only uppercase letters.

The **\X** escape matches any number of Unicode characters that form an extended Unicode sequence. The **\X** is equivalent to the following example.

```
(?>\PM\pM*)
```

That is, it matches a character without the "mark" property, followed by zero or more characters with the "mark" property, and treats the sequence as an atomic group. Characters with the "mark" property are typically accents that affect the preceding character.

Matching characters by Unicode property is not fast because SAS must search a structure that contains data for over fifteen thousand characters.

Simple Assertions

The fourth use for backslash is certain simple assertions. An assertion specifies a condition that must be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The backslashed assertions include:

Character	Description
\b	Matches at a word boundary
\B	Matches when not at a word boundary
\A	Matches at start of subject
\Z	Matches at end of subject or before newline at end

Character	Description
\z	Matches at end of subject

You cannot use these assertions in character classes, but note that **\b** has a different meaning, the backspace character, inside a character class.

A word boundary is a position in the subject string where the current character and the previous character do not both match **\w** or **\W** (one matches **\w** and the other matches **\W**) or the start or end of the string if the first or last character matches **\w**, respectively.

The **\A**, **\Z**, and **\z** assertions differ from the traditional circumflex and dollar in that they match at the beginning and end of the subject string. The difference between **\Z** and **\z** is that **\Z** matches before a newline at the end of the string as well as at the very end, whereas **\z** matches only at the end.

Circumflex and Dollar

Outside a character class, in the default matching mode, the circumflex character (^) is an assertion that is true only if the current matching point is at the beginning of the subject string. Inside a character class, circumflex has an entirely different meaning.

Circumflex need not be the first character of the pattern if a number of alternatives are involved, but it should be the first character in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex that is, if the pattern is constrained to match only at the start of the subject is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character ($) is an assertion that is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar does not need to be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

Full Stop (Period, Dot)

Outside a character class, a dot (.) in the pattern matches any one character in the subject string except (by default) a character that signifies the end of a line. The matched character can be more than one byte long. When a line ending is defined as a single character (CR or LF), dot never matches that character. When the two-character sequence CRLF is used, dot does not match CR if it is immediately followed by LF. Otherwise, it matches all characters (including isolated CRs and LFs).

The handling of dot is entirely independent of the handling of circumflex and dollar. The only relationship is that they all involve newline characters. Dot has no special meaning in a character class.

Matching a Single Byte

Outside a character class, the escape sequence \C matches any one byte. Unlike a dot, it always matches CR and LF. The feature is provided in Perl in order to match individual bytes in UTF-8 mode. Since it breaks up UTF-8 characters into individual bytes, what remains in the string can be a malformed UTF-8 string. For this reason, the \C escape sequence is best avoided.

SAS does not allow \C to appear in look behind assertions, because this would make it impossible to calculate the length of the look behind for UTF-8 characters.

Square Brackets and Character Classes

An opening square bracket ([) introduces a character class, terminated by a closing square bracket (]). A closing square bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject. The character can occupy more than one byte. A matched character must be in the set of characters defined by the class, unless the first character in the class definition is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class **[aeiou]** matches any lowercase vowel, but **[^aeiou]** matches any character that is not a lowercase vowel. Note that a circumflex is just a convenient notation for specifying the characters that are in the class by enumerating those that are not. It is not an assertion; it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

Characters with values greater than 255 can be included in a class as a literal string of bytes, or by using the \x{ escaping mechanism.

When caseless matching is set, any letters in a class represent both their upper and lowercase versions. So, for example, a caseless **[aeiou]** matches **A** as well as a, and a caseless **[^aeiou]** does not match **A**, whereas a caseful version would.

Characters that might indicate line breaks (CR and LF) are never treated in any special way when matching character classes.

You can use the minus or hyphen (-) character to specify a range of characters in a character class. For example, [d-m] matches any letter between d and m, inclusive. If a minus character is required in a class, it must be escaped with a backslash or

appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class.

It is not possible to have the literal character **]** as the end character of a range. A pattern such as **[W-]46]** is interpreted as a class of two characters (**W** and **-**) followed by a literal string **46]**, so it would match **W46]** or **-46]**. However, if the **]** is escaped with a backslash, it is interpreted as the end of range, so **[W-\]46]** is interpreted as a single class containing a range followed by two separate characters. You can also use the octal or hexadecimal representation of **]** to end a range.

Ranges operate in the collating sequence of character values. They can also be used for characters specified numerically for example, **[\000-\037]**. Ranges can include characters whose values are greater than 255 for example, **[\x{100}-\x{2ff}]**. If a range that includes letters is used when caseless matching is set, it matches the letters in either case. For example, **[W-c]** is equivalent to **[][\ \^_`wxyzabc]**, matched without case.

The character types **\d**, **\D**, **\p**, **\P**, **\s**, **\S**, **\w**, and **\W** can also appear in a character class, and add the characters that they match to the class. For example, **[\dABCDEF]** matches any hexadecimal character. A circumflex can conveniently be used with the uppercase character types to specify a more restricted set of characters than the matching lowercase type. For example, the class **[^\W_]** matches any letter or digit, but not underscore.

The recognized meta-characters in character classes are backslash, hyphen (when interpreted in a specified range), circumflex (at the start), opening square bracket (when interpreted with a POSIX class name), and the terminating closing square bracket. However, escaping other non-alphanumeric characters does no harm. See POSIX Character Classes on page 75.

POSIX Character Classes

Perl supports the POSIX notation for character classes. This uses names enclosed by **[:** and **:]** within the enclosing square brackets. SAS also supports this notation. For example,

```
[01[:alpha:]%}
```

matches "0", "1", any alphabetic character, or %. The supported class names include:

Class Name	Description
alnum	letters and digits
alpha	letters
ascii	character codes 0 - 127
blank	space or tab only
cntrl	control characters

Class Name	Description
digit	decimal digits (same as \d)
graph	printing characters, excluding space
lower	lowercase letters
print	printing characters, including space
punct	printing characters, excluding letters and digits
space	white space (not quite the same as \s)
upper	uppercase letters
word	"word" characters (same as \w)
xdigit	Hexadecimal characters

The **space** characters are HT (9), LF (10), VT (11), FF (12), CR (13), and space (32). Notice that this list includes the VT character (code 11). This makes **space** different from **\s**, which does not include VT (for Perl compatibility).

The name **word** is a Perl extension, and **blank** is a GNU extension from Perl version 5.8. Another Perl extension is negation, which is indicated by a **^** character after the colon. For example,

```
[12[:^digit:]]
```

matches "1", "2", or any non-digit. SAS (and Perl) also recognize the POSIX syntax [.ch.] and [=ch=] where **ch** is a collating element, but these are not supported, and an error is given if they are encountered.

Vertical Bar

You can use the vertical bar character (|) to separate alternative patterns. For example, the following pattern matches either "gilbert" or "sullivan."

```
gilbert|sullivan
```

Any number of alternatives can appear, and an empty alternative is permitted, matching the empty string. The matching process tries each alternative in turn, from left to right, and the first alternative that succeeds is used. If the alternatives are within a subpattern (defined below), **succeeds** means matching the rest of the main pattern as well as the alternative in the subpattern.

Internal Option Settings

Some matching options can be changed from within the pattern by a sequence of Perl option letters enclosed between **(?** and **)**. The option letters are:

Option Letters	Description
i	for CASELESS
m	for MULTILINE
s	for DOTALL

For example, (?im) sets caseless, multi-line matching.

When an option change occurs at top level (that is, not inside subpattern parentheses), the change applies to the remainder of the pattern that follows. If the change is placed right at the start of a pattern, SAS extracts it into the global options for that expression.

An option change within a subpattern affects only that part of the current pattern that follows, so

```
(a(?i)b)c
```

matches abc and aBc and no other strings. By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example,

```
(a(?i)b|c)
```

matches "ab", "aB", "c", and "C", even though when matching C the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time (there would be some strange behavior otherwise).

Subpatterns

Subpatterns are delimited by parentheses **(** and **)**, which you can nest. Marking part of a pattern as a subpattern does two things:

It localizes a set of alternatives. For example, the following pattern matches one of the words "cat," "cataract," or "caterpillar."

```
cat(aract|erpillar|)
```

Without the parentheses, it would match "cataract," "erpillar," or the empty string.

It also sets up the subpattern as a capturing subpattern. When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller through the ovector argument **pcre_exec()**. Opening parentheses

are counted from left to right (starting from 1) to obtain numbers for the capturing subpatterns.

For example, if the string "the red king" is matched against the pattern:

```
the ((red|white) (king|queen))
```

the captured substrings are "red king," "red," and "king," and are numbered 1, 2, and 3, respectively.

The fact that parentheses fulfill two functions is not always helpful. There are often when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by a question mark and a colon **(?:**, the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string "the white queen" is matched against the pattern:

```
the ((?:red|white) (king|queen))
```

the captured substrings are "white queen" and "queen", and are numbered 1 and 2. The maximum number of capturing subpatterns is 65535, and the maximum depth of nesting of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters can appear between the "?" and the ":." Thus, the following two patterns match exactly the same set of strings.

```
(?i:saturday|sunday) (?:(?i)saturday|sunday)
```

Here, the alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached. An option setting in one branch affects subsequent branches, so the above patterns match "SUNDAY" as well as "Saturday."

Repetition

Repetition is specified by quantifiers, which can follow any of the following items:

- a literal data character
- the . meta-character
- the \C escape sequence
- the \X escape sequence
- an escape such as \d that matches a single character
- a character class
- a back reference (see Atomic Grouping and Possessive Quantifiers)
- an enclosed in parentheses subpattern, unless it is an assertion

The general repetition quantifier specifies a minimum and maximum number of permitted matches by giving the two numbers in braces ({ and }), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example:

```
z{2,4}
```

matches **zz**, **zzz**, or **zzzz**. A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit. If the second number and the comma are both omitted, the quantifier specifies an exact number of required matches. Thus, the following example matches at least three successive vowels.

```
[aeiou]{3,}
```

However, the example can match many more matches with exactly eight digits.

```
\d{8}
```

An opening brace that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, {,6} is not a quantifier, but a literal string of four characters.

Quantifiers apply to UTF-8 characters rather than to individual bytes. For example,`\x{100}{2}` matches two UTF-8 characters. Each match is represented by a two-byte sequence. Similarly, when Unicode property support is available, `\X{3}` matches three Unicode extended sequences. Each match can be several bytes long (and they might be of different lengths).

The quantifier {0} is permitted, causing the expression to behave as if the previous item and the quantifier were not present.

For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

- * is equivalent to {0,}
- + is equivalent to {1,}
- ? is equivalent to {0,1}

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit. For example:

```
(a?)*
```

Earlier versions of Perl and SAS used to give an error for such patterns. However, because there are cases where this can be useful, such patterns are now accepted, but if any repetition of the subpattern does in fact does not match any characters, the loop is forcibly broken.

By default, the quantifiers match as much as possible (up to the maximum number of permitted times), without causing the rest of the pattern to fail. The classic example of where this causes problems is in trying to match comments in C programs. These appear between **/*** and ***/** and within the comment, individual * and **/** characters can appear. An attempt to match C comments by applying the pattern `/\*.*\*/` to the following string fails, because it matches the entire string because of the .* item.

```
/* first comment */ not comment /* second comment */
```

However, if a quantifier is followed by a question mark, it does not match the maximum number, and instead matches the minimum number of times possible, so the pattern `/\*.*?\*/` does the right thing with the C comments.

The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled. For example, `\d??\d` matches one digit by preference, but can match two if that is the only way the rest of the pattern matches.

These quantifiers try to match the maximum number by default, but individual ones can be made to match a minimum by following them with a question mark. In other words, it inverts the default behavior.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after the string `(tweedle[dume]{3}\s*)+` has matched "tweedledum tweedledee" the value of the captured substring is "tweedledee".

However, if there are nested capturing subpatterns, the corresponding captured values can have been set in previous iterations. For example, after `/(a|(b))+/` matches "aba" the value of the second captured substring is "b".

Atomic Grouping and Possessive Quantifiers

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to change the nature of the match, or to cause it to fail earlier, when the author of the pattern knows there is no point in carrying on. For example, the pattern `\d+one` when applied to the subject line results returns the following string.
`123456two`

After matching all 6 digits and then failing to match "one", the normal action of the match is to try again with only 5 digits matching the `\d+` item, and then 4, and so on, before failing. "Atomic grouping" (a term used in "Mastering Regular Expressions" by Jeffrey Friedl) provides the means for specifying that once a subpattern has matched, it is not to be re-evaluated in this way.

If atomic grouping is used for the previous example, the match fails to match "one" the first time. The notation is a type of special parenthesis, starting with `(?>` as in the following example `(?>\d+)one`.

This type of parenthesis "locks up" the part of the pattern that it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern matches, if anchored at the current point in the subject string.

Atomic grouping subpatterns are not capturing subpatterns. Simple cases such as the above example can be thought of as a maximizing repeat that must swallow everything it can.

Both `\d+` and `\d+?` are prepared to adjust the number of digits they match in order to make the rest of the pattern match, `(?>\d+)` can match an entire sequence of digits.

Atomic groups in general can contain arbitrarily complicated subpatterns, and can be nested. However, when the subpattern for an atomic group is just a single repeated item, as in the example above, a simpler notation, called a "possessive quantifier" can be used. This consists of an additional + character following a quantifier. Using this notation, the previous example can be rewritten as, `\d++one`.

Possessive quantifiers are always greedy. They are a convenient notation for the simpler forms of atomic group. However, there is no difference in the meaning or processing of a possessive quantifier and the equivalent atomic group. The possessive quantifier syntax is an extension to the Perl syntax.

When a pattern contains an unlimited repeat inside a subpattern that can itself be repeated an unlimited number of times, the use of an atomic group is the only way to avoid some failing matches taking a very long time. The pattern:

```
(\D+|<\d+>)*[!?]
```

matches an unlimited number of substrings that either consist of non-digits, or digits enclosed in **<>**, followed by either **!** or **?**. When it matches, it runs quickly. However, if it is applied to the following example, then it takes a long time before reporting failure.

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

The string can be divided between the internal `\D+` repeat and the external * repeat in a large number of ways, and all have to be tried. (The example uses `[!?]` rather than a single character at the end, because both SAS and Perl have an optimization that allows for fast failure when a single character is used. They remember the last single character that is required for a match, and fail early if it is not present in the string.) If the pattern is changed so that it uses an atomic group, like this:

```
((?>\D+)|<\d+>)*[!?]
```

sequences of non-digits cannot be broken, and failure happens quickly.

Back References

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. A "forward back reference" of this type can make sense when a repetition is involved and the subpattern to the right has participated in an earlier iteration. See Backslash on page 66 for more information about the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. (See Subpatterns as Subroutines on page 87 for more information. Therefore, the pattern `(sens|respons)e and \1ibility` matches "sense and sensibility" and "response and responsibility," but not "sense and responsibility."

If caseful matching is in force at the time of the back reference, the case of letters is relevant. For example, `((?i)rah)\s+\1` matches "rah rah" and "RAH RAH," but not "RAH rah," even though the original capturing subpattern is matched without case.

There can be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, any back references to it always

fail. For example, the pattern `(a|(bc))\2` always fails if it starts to match a rather than bc. Because there can be many capturing parentheses in a pattern, all digits following the backslash are taken as part of a potential back reference number.

If the pattern continues with a digit character, some delimiter must be used to terminate the back reference. Here an empty comment (see Comments) can be used.

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used. So, for example, (a\1) never matches. However, such references can be useful inside repeated subpatterns. For example, the pattern `(a|b\1)+` matches any number of **a**'s and also **aba**, **ababbaa**, and so on.

At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. For this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of 0.

Assertions

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as **\b**, **\B**, **\A**, **\Z**, **\z**, **^**, and **$** are described above. More complicated assertions are coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except it does not cause the current matching position to be changed. An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed.

Assertion subpatterns are not capturing subpatterns, and cannot be repeated, because it makes no sense to assert the same thing several times. If any type of assertion contains capturing subpatterns within it, these are counted for the purposes of numbering the capturing subpatterns in the whole pattern. However, substring capturing is carried out only for positive assertions, because it does not make sense for negative assertions.

Look Ahead Assertions

Look ahead assertions start with (?= for positive assertions and (?! for negative assertions. For example,

```
\w+(?=;)
```

matches a word followed by a semicolon, but does not include the semicolon in the match, and:

```
one(?!two)
```

matches any occurrence of "one" that is not followed by "two." Note that the similar pattern `(?!one)two` does not find an occurrence of "two" that is preceded by something other than "one". Instead, it finds any occurrence of "two", because the

assertion **(?!one)** is always true when the next three characters are "two." A look behind assertion is needed to achieve this effect.

If you want to force a matching failure at some point in a pattern, the most convenient way to do it is with **(?!)** because an empty string always matches, so an assertion that requires there not to be an empty string must always fail.

Look Behind Assertions

Look behind assertions start with **(?<=** for positive assertions and **(?<!** negative assertions. For example:

```
(?<!one)two
```

does find an occurrence of "two" that is not preceded by "one." The contents of a look behind assertion are restricted so that all the strings it matches must have a fixed length. However, if there are several alternatives, they do not all have to have the same fixed length. For example:

```
(?<=leopard|donkey)
```

is permitted, but:

```
(?<!dogs?|cats?)
```

causes an error at compile time. Branches that match different length strings are permitted only at the top level of a look behind assertion. This is an extension compared with Perl (for version 5.8), which requires all branches to match the same length of string. An assertion such as `(?<=ab(c|de))` is not permitted because its single top-level branch can match two different lengths, but it is acceptable if rewritten to use two top-level branches.

```
(?<=abc|abde)
```

The implementation of look behind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail.

SAS does not allow the **\C** escape to appear in look behind assertions, because it makes it impossible to calculate the length of the look behind for UTF-8 characters. The **\X** escape, which can match different numbers of bytes, is also not permitted.

Atomic groups can be used in conjunction with look behind assertions to specify efficient matching at the end of the subject string. Consider a simple pattern such as `abcd$`, when applied to a long string that does not match. Because matching proceeds from left to right, SAS looks for each **a** in the subject and then sees what follows matches the rest of the pattern. If the pattern is specified as `^.*abcd$` the initial **.*** matches the entire string at first, but when this fails (because there is no following **a**), it backtracks to match all but the last character, and then all but the last two characters, and so on. Once again the search for **a** covers the entire string, from right to left, so the results are no better. However, if the pattern is written as `^(?>.*)(?<=abcd)` or, equivalently, using the possessive quantifier syntax, `^.*+(?<=abcd)` there can be no backtracking for the .* item. It can match only the entire string. The subsequent look behind assertion does a single test on the last four characters. If it fails, the match fails immediately. For long strings, this approach makes a significant difference to the processing time.

Using Multiple Assertions

Several assertions (of any sort) can occur in succession. For example:
`(?<=\d{3})(?<!999)one`

matches "one" preceded by three digits that are not **999**. Notice that each of the assertions is applied independently at the same point in the subject string. First, there is a check that the previous three characters are all digits, and then there is a check that the same three characters are not **999**. This pattern does not match "one" preceded by six characters, the first of which are digits and the last three of which are not **999**. For example, it does not match `123abcone`. However, a pattern to do that does match is `(?<=\d{3}...)(?<!999)one`. This time, the first assertion looks at the preceding six characters, checking that the first three are digits, and then the second assertion checks the preceding three characters are not 999.

Assertions can be nested in any combination. For example, `(?<=(?<!one)two)cat` matches an occurrence of "cat" that is preceded by "two". This, in turn is not preceded by "one," while, `(?<=\d{3}(?!999)...) one` is another pattern, which matches "one" preceded by three digits and any three characters that are not **999**.

Conditional Subpatterns

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern shown here.
`(?(condition)yes-pattern)`
`(?(condition)yes-pattern|no-pattern)`

If the condition is satisfied, the yes-pattern is used. Otherwise, the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are three types of conditions. If the text between the parentheses consists of a sequence of digits, or a sequence of alphanumeric characters and underscores, the condition is satisfied, if the capturing subpattern of that number or name previously matched. There is a possible ambiguity here, because subpattern names can consist entirely of digits. First, it looks for a named subpattern. If it cannot find one and the text consists entirely of digits, it looks for a subpattern of that number, which must be greater than zero. Using subpattern names that consist entirely of digits is not recommended.

Consider the following pattern, which contains non-significant white space (to make it more readable and to divide it into three parts for ease of discussion):
`( \( )? [^()]+ (?(1) \) )`

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern

that tests whether the first set of parentheses matched or not. If they did, if the subject started with an opening parenthesis, the condition is true, and the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses. Rewriting it to use a named subpattern gives this:

```
(?P \( )? [^()]+ (?(OPEN) \) )
```

If the condition is the string (R), and there is no subpattern with the name R, the condition is satisfied if a recursive call to the pattern or subpattern has been made. At "top level", the condition is false. This is a SAS extension. Recursive patterns are described in the next section.

If the condition is not a sequence of digits or (R), it must be an assertion. This can be a positive or negative look ahead or look behind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line:

```
(?(?=[^a-z]*[a-z])
\d{2}-[a-z]{3}-\d{2} | \d{2}-\d{2}-\d{2} )
```

The condition is a positive look ahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms dd-aaa-dd or dd-dd-dd, where aaa are letters and dd are digits.

Comments

The sequence (?# marks the start of a comment that continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching.

Recursive Patterns

Consider the problem of matching a string in parentheses, allowing for unlimited nested parentheses. Without the use of recursion, the best that can be done is to use a pattern that matches up to some fixed depth of nesting. It is not possible to handle an arbitrary nesting depth. Perl provides a facility that allows regular expressions to recurse (among other things). It does this by interpolating Perl code in the expression at run time, and the code can refer to the expression itself. A Perl pattern to solve the parentheses problem can be created like this:

```
$re = qr{\( (?: (?>[^()]+) | (?p{$re}) )* \)}x;
```

The **(?p{...})** item interpolates Perl code at run time, and in this case refers recursively to the pattern in which it appears. SAS cannot support the interpolation of Perl code. Instead, it supports special syntax for recursion of the entire pattern, and for individual subpattern recursion.

The special item that consists of **(?** followed by a number greater than zero and a closing parenthesis is a recursive call of the subpattern of the given number, provided that it occurs inside that subpattern. (If not, it is a "subroutine" call, which is described in Subpatterns as Subroutines). The special item **(?R)** is a recursive call of the entire regular expression.

A recursive subpattern call is always treated as an atomic group. That is, once it has matched some of the subject string, it is never re-entered, even if it contains untried alternatives and there is a subsequent matching failure.

This pattern solves the nested parentheses problem:

```
\( ( (?>[^()]+) | (?R) )* \)
```

First it matches an opening parenthesis. Then it matches any number of substrings that can either be a sequence of non-parentheses, or a recursive match of the pattern itself (that is, a correctly enclosed in parentheses substring). Finally, there is a closing parenthesis.

If this were part of a larger pattern, you would not want to recurse the entire pattern, so instead you could use this:

```
( \( ( (?>[^()]+) | (?1) )* \) )
```

The pattern is in parentheses, and causes the recursion to refer to them instead of the whole pattern. In a larger pattern, keeping track of parenthesis numbers can be tricky. It can be more convenient to use named parentheses instead. For this, `(?P>name)` is used (an extension to the Python syntax that SAS uses for named parentheses. Note that Perl does not provide named parentheses. You could rewrite the above example like the following, `(?P \( ( (?>[^()]+) | (?P>pn) )* \) )`

This particular example pattern contains nested unlimited repeats. Therefore, the use of atomic grouping for matching strings of non-parentheses is important when applying the pattern to strings that do not match. For example, when this pattern is applied to the following string that yields "no match" quickly. However, if atomic grouping is not used, the match runs for a very long time because there are so many different ways the + and * repeats can separate the subject, and all have to be tested before failure can be reported.

```
(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa()
```

At the end of a match, the values set for any capturing subpatterns are those from the outermost level of the recursion at which the subpattern value is set. If you want to obtain intermediate values, a `callout` function can be used. (See Subpatterns as Subroutines on page 77 and the precallout documentation.) If the pattern above is matched against `(ab(cd)ef)`, then the value for the capturing parentheses is `ef`, which is the last value taken on at the top level. If additional parentheses are added, giving `\( ( ( (?>[^()]+) | (?R) )* ) \)` the string that they capture is `ab(cd)ef`, the contents of the top level parentheses. If there are more than 15 capturing parentheses in a pattern, SAS has to obtain extra memory to store data during a recursion. If no memory can be obtained, the match fails with an error.

Do not confuse the **(?R)** item with the condition **(R)**, which tests for recursion. Consider this pattern, which matches text in angle brackets, allowing for arbitrary nesting. Only digits are allowed in nested brackets during recursion, whereas any characters are permitted at the outer level.

```
< (?: (?(R) \d++ | [^<>]*+) | (?R)) * >
```

In this pattern, (?(R) is the start of a conditional subpattern, with two different alternatives for the recursive and non-recursive cases. The (?R) item is the actual recursive call.

Subpatterns as Subroutines

If the syntax for a recursive subpattern reference (either by number or by name) is used outside the parentheses to which it refers, it operates like a subroutine in a programming language. An earlier example demonstrated that the pattern:

```
(sens|respons)e and \1ibility
```

matches "sense and sensibility" and "response and responsibility", but not "sense and responsibility". If the pattern:

```
(sens|respons)e and (?1)ibility
```

is used, it matches "sense and responsibility" as well as the other two strings. Such references, if given numerically, must follow the subpattern to which they refer. However, named references can refer to later subpatterns.

Like recursive subpatterns, a "subroutine" call is always treated as an atomic group. That is, once it has matched some of the subject string, it is never re-entered, even if it contains untried alternatives and there is a subsequent matching failure.

Unicode Support

Special information about support of Unicode characters is given below:

1. An unbraced hexadecimal escape sequence (such as \xb3) matches a two-byte UTF-8 character if the value is greater than 127.
2. Octal numbers up to \777 are recognized, and match two-byte UTF-8 characters for values greater than \177.
3. Repeat quantifiers apply to complete UTF-8 characters, not to individual bytes, for example: \x{100}{3}.
4. The dot (.) meta-character matches one UTF-8 character instead of a single byte.
5. The escape sequence \C can be used to match a single byte in UTF-8 mode, but its use can lead to some strange results.
6. Similarly, characters that match the POSIX named character classes are all low-valued characters.

Differences from Perl

This section describes the differences in the ways that SAS and Perl handle regular expressions. The differences described here compare SAS with Perl version 5.8.

1. SAS has only a subset of the Perl UTF-8 and Unicode support. Details of what it does have are given above.
2. SAS does not allow repeat quantifiers on look ahead assertions. Perl permits them, but they do not mean what you might think. For example, (?!a){3} does not assert that the next three characters are not "a". It asserts that the next character is not "a" three times.
3. Capturing subpatterns that occur inside negative look ahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative look ahead assertion contains just one branch.
4. Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence \0 can be used in the pattern to represent a binary zero.
5. The properties that can be tested with `\p` and `\P` are limited to the general category properties. For example, the properties `Lu` and `Nd`, script names like `Greek` or `Han`, and the derived properties `Any` and `L&`.
6. SAS does support the `\Q...\E` escape for quoting substrings. Characters in between are treated as literals. This is slightly different from Perl in that `$` and `@` are also handled as literals inside the quotation marks. In Perl, they cause variable interpolation (but SAS does not have variables). Note the following table with examples.

Note: The `\Q...\E` sequence is recognized both inside and outside character classes.

Pattern	SAS Matches	Perl Matches
\Qabc$xyz\E	abc$xyz	abc followed by the contents of $xyz
\Qabc\$xyz\E	abc\$xyz	abc\$xyz
\Qabc\E\$\Qxyz\E	abc$xyz	abc$xyz

7 SAS does not support the `(?{code})` and `(?p{code})` constructions. However, there is support for recursive patterns using the non-Perl items `(?R)`, `(?number)`, and `(?P>name)`.

8 There are some differences that are concerned with the settings of captured strings when part of a pattern is repeated. For example, matching "`aba`" against the pattern `/^(a(b)?)+$/` in Perl leaves `$2` unset, but in SAS it is set to "`b`".

9 SAS provides some extensions to the Perl regular expression facilities:

   a Although look behind assertions must match fixed length strings, each alternative branch of a look behind assertion can match a different length of string. Perl requires them all to have the same length.

   b Where a backslash is followed by a letter with no special meaning, the backslash is always ignored (Perl can be made to issue a warning).

   c The greediness of the repetition quantifiers is inverted; that is, by default they are not greedy, but if followed by a question mark that they are .

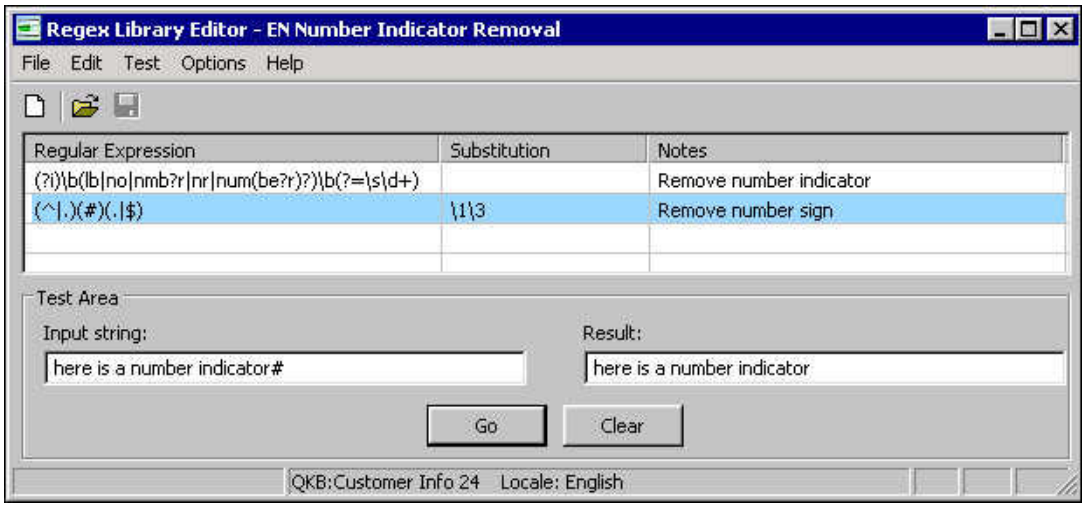
   d The `(?R)`, `(?number)`, and `(?P>name)` constructs allow for recursive pattern matching (Perl can do this using the `(?p{code})` construct, which PCRE cannot support.)

   e SAS supports the possessive quantifier "++" syntax, taken from the Sun© Java© package.

Navigating the Regex Library Editor

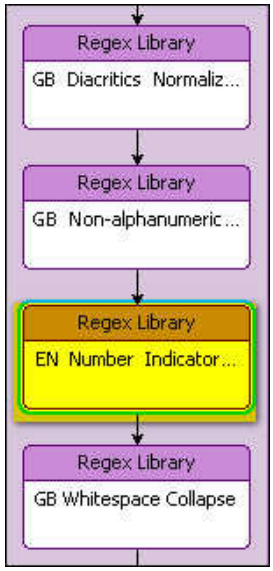
A regular expression is a pattern that is matched against a subject string from left to right. A regex library is a type of QKB file that contains regular expressions that follow Perl syntax. These expressions enable you to express how a pattern in a string should be detected and, optionally, replaced. Regex libraries have many uses in a QKB definition. They can be used to conduct pattern-based searches in strings and for purposes of categorization. They can be used to perform conditional pattern-based transformations of strings. They can also be used to normalize words for lookup in schemes and vocabularies.

The next display shows the regular expressions in the **EN Number Indicator Removal** library in the English locale in the SAS Quality Knowledge Base for Contact Information 24.



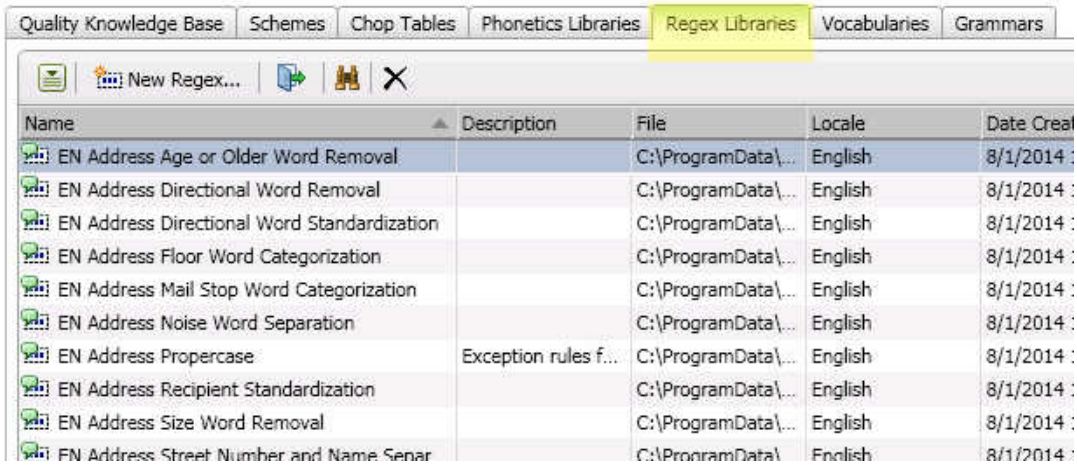
Regular expressions that you create in the Regex Library Editor must comply with the syntax defined for Perl regular expressions. For details about this syntax, see Regular Expression Syntax in the Regex Library Editor on page 92.

The next display shows the **EN Number Indicator Removal** library in the context of the Address match definition in the English (United States) locale in the SAS Quality Knowledge Base for Contact Information 24.



Regex Library Editor Tab

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right side of the screen. The **Regex Libraries** tab shows a list of regex libraries that are contained in the QKB.



The **Name** column shows the name of a regex library. The **Description** column contains text about the library. The **File** column contains the physical path to the library file.

The **Locale** column shows the QKB locale with which the library is associated. Note that a regex library might be listed even if it is not associated with the locale that you have selected in the **Quality Knowledge Bases** tree. This is because locales can inherit regex libraries from ancestors on page 163.

Select a locale and look at the values in the **Locale** column for a regex library. You see that some libraries are associated directly with the selected locale. Other libraries are associated with the locale's language or are associated with the **Global** set of objects. This information is also presented in the icons that are shown for the libraries in the list. As shown in the previous display, if an icon has a flag overlay, the library is associated directly with a locale. If the icon has a word balloon overlay, it means that the library is associated with a language. An earth overlay means that the library belongs to the **Global** set of regex libraries.

To view only the regex libraries that are associated with a language, select that language in the tree for your QKB in the **Administration** riser.

To view only the regex libraries in the **Global** set, click on **Global** in the tree for your QKB.

Find a Regex Library by Name

To find a regex library with a specific name, click **Find** in the lower toolbar. This toggles on the **Find** pane. To toggle off the **Find** pane, click **Find**.

Rename and Duplicate Regex Libraries

You can rename a regex library by right-clicking the name and selecting **Rename** from the menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a regex library by right-clicking the name and selecting **Duplicate** from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Regex Library

To create a new regex library, click **New Regex Library** on the lower toolbar, or select **New Regex Library** from the menu on the lower toolbar. The New Regex Library wizard opens. Specify a name, description, and a locale for the new library. After you have saved this information, the new library appears in the regex library list, and the Regex Library Editor on page 59 opens. Enter one or more expressions to the library. For details about this task, see Adding and Editing Regex Libraries on page 62.

Open a Regex Library for Editing

To open a regex library for editing, double-click on the library in the list, or select the library and click **Open** on the lower toolbar. You might also select the library and choose **Open** in the pop-up menu on the lower toolbar. Add or update expressions in the library. For details about this task, see Adding and Editing Regex Libraries on page 62.

View Definitions That Use a Given Regex Library

To view a list of QKB definitions that use a given regex library, select the library and choose **Usage** in the pop-up menu on the lower toolbar.

Delete a Regex Library

To delete a regex library, select the library and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu in the lower toolbar. The library is removed from the QKB.

Regex Library Editor - Syntax

Introduction

The Regex Library Editor supports Perl-compatible syntax for regular expressions. A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject. For example, the pattern `The quick brown fox` matches a portion of a subject string identical to itself. The power of regular expressions lies in their ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of meta-characters, which do not stand for themselves but instead are interpreted in some special way.

A substitution is a string inserted into the output of a regular expression call in place of the pattern matched by the regular expression. To further the example, suppose the following regular expression and substitution are applied to the input string.

Regular Expression	Substitution
brown	blue

Result: `The quick blue fox`

In a substitution string, all characters stand for themselves, except the backslash-escape digit is interpreted as a back reference to a captured subpattern. See information about capturing subpatterns with regular expressions. Also, casing operators `\U`, `\u`, `\L`, `\l`, and `\E` can be used in substitution strings just like Perl substitutions.

Regular expressions that you create in the Regex Library Editor must comply with the syntax defined for Perl regular expressions. The sections below describe some typical syntax for expressions in a SAS regex library. For a description of the differences in the ways that SAS and Perl handle regular expressions, see Differences From Perl on page 113. For general information about writing Perl regular expressions, see "Mastering Regular Expressions" by Jeffrey E.F. Friedl or other readily available reference material on the subject.

Meta-Characters

There are two different sets of meta-characters: those recognized anywhere in the pattern except within square brackets ([ ]), and those recognized in square brackets. Outside square brackets, the meta-characters are:

Character	Description
\	General escape character with several uses
^	Assert start of subject (or line, in multi-line mode)
$	Assert end of subject (or line, in multi-line mode)
.	Match any character except new line (by default)
[	Start character class definition
|	Start of alternative branch
(	Start subpattern
)	End subpattern
?	Extend the meaning of (; 0 or 1 quantifier; quantifier minimizer
*	0 or more quantifier

Character	Description
+	1 or more quantifier; also possessive quantifier
{	Start minimum or maximum quantifier

The part of a pattern in square brackets is called a character class. In a character class, the meta-characters are the following.

Character	Description
\	General escape character
^	Negate the class, but only the first character
-	Indicate character range
[	POSIX character class (only if followed by POSIX syntax)
]	Terminate the character class

The following sections describe the use of each of the meta-characters.

Backslash

The backslash character (\) has several uses. First, if it is followed by a non-alphanumeric character, it takes away any special meaning the character has. This use of backslash as an escape character applies both inside and outside character classes. For example, if you want to match a * character, you write * in the pattern. This applies whether the following character is interpreted as a meta-character. Therefore, it is always safe to precede a non-alphanumeric character with \ to specify that it stands for itself. For example, if you want to match a backslash, type \\.

If you want to remove the special meaning from a sequence of characters, you can do so by putting them between \Q and \E. This is different from Perl in that $ and @ are handled as literals in \Q...\E sequences in SAS, whereas in Perl, $ and @ cause variable interpolation. Note the following examples:

Pattern	SAS Matches	Perl Matches
\Qabc$xyz\E	abc$xyz	abc followed by the contents of $xyz
\Qabc\$xyz\E	abc\$xyz	abc\$xyz
\Qabc\E\$\Qxyz\E	abc$xyz	abc$xyz

The \Q...\E sequence is recognized both inside and outside character classes.

Non-printing Characters

Second, the backslash provides a way of encoding non-printing characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern. However, when you are preparing a pattern by text editing, it is usually easier to use one of the following escape sequences than the binary character that it represents .

Character	Description
\a	Alarm; that is, the BEL character (hex 07)
\cx	Control-x, where x is any character
\e	Escape (hex 1B)
\f	Form feed (hex 0C)
\n	New line (hex 0A)
\r	Carriage return (hex 0D)
\t	Tab (hex 09)
\ddd	Character with octal code ddd, or back reference
\xhh	Character with hex code HH
\x{hhh..}	Character with hex code hhh..

The precise effect of `\cx` is as follows: if `x` is a lowercase character, it is converted to uppercase. Then, bit 6 of the character (hex 40) is inverted. Thus, `\cz` becomes hex 1A, but `\c{` becomes hex 3B, and `\c;` becomes hex 7B.

After `\x`, up to two hexadecimal characters are read. These digits can be in upper or lowercase. Any number of hexadecimal characters can appear between `\x{` and `}`. However, the value of the character code must be less than `2**31` (that is, the maximum hexadecimal value is 7FFFFFFF). If characters other than hexadecimal characters appear between `\x{` and `}`, or if there is no terminating `}`, this form of escape is not recognized. Instead, the initial `\x` is interpreted as a basic hexadecimal escape, with no following digits, giving a character whose value is zero.

Characters whose value is less than 256 can be defined by either of the two syntaxes for `\x`. There is no difference in the way they are handled. For example, `\xdc` is exactly the same as `\x{dc}`.

After `\0` up to two further octal digits are read. If there are fewer than two digits, just those that are present are used. Thus, the sequence `\0\x\07` specifies two binary zeros followed by a BEL character (code value 7). Make sure you supply two digits after the initial zero if the pattern character that follows is itself an octal digit.

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, SAS reads it and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing left parentheses in the expression, the entire sequence is taken as a back reference. See a description of how this works under Back Reference on page 108.

Inside a character class, if the decimal number is greater than 9, and there are not many captured subpatterns, Data Quality re-reads up to three octal digits following the backslash. Then it generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves. For example:

Character	Description
\040	Another way of writing a space
\40	The same, provided there are fewer than 40 previous capturing subpatterns
\7	Always a back reference
\11	Might be a back reference, or another way of writing a tab
\011	Always a tab
\0113	A tab followed by the character 3
\113	The character with octal code 113, because there can be no more than 99 back references
\377	A byte consisting entirely of 1 bit
\81	Either a back reference or a binary zero followed by the two characters, 8 and 1

Note: You must not introduce octal values of 100 or greater because no more than three octal digits are ever read.

You can use all the sequences that define a single-byte value both inside and outside character classes. In addition, inside a character class, the sequence `\b` is interpreted as the backspace character (hex 08). The sequence `\x` is interpreted as the character `x`. Outside a character class, these sequences have different meanings, see the next section, Generic Character Types.

Generic Character Types

The third use of backslash is to specify generic character types:

Character	Description
\d	Any decimal digit

Character	Description
\D	Any character that is not a decimal digit
\s	Any whitespace character
\S	Any character that is not a whitespace character
\w	Any word character
\W	Any non-word character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

These character type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

For compatibility with Perl, `\s` does not match the VT character (code 11). This makes it different from the POSIX `space` class. The `\s` characters are HT (9), LF (10), FF (12), CR (13), and space (32). (If "use locale;" is included in a Perl script, `\s` can match the VT character. In SAS, it never does.)

A "`word`" character is an underscore or any character that is a letter or digit. The definition of letters and digits is controlled by the SAS Unicode character tables.

Unicode Character Properties

Three additional escape sequences to match Unicode character properties are available.

Character	Description
\p{xx}	A character with the xx property
\P{xx}	A character without the xx property
\X	An extended Unicode sequence

The property names represented by xx above are limited to the Unicode script names, the general category properties, and "Any", which matches any character (including newline). Other properties such as "InMusicalSymbols" are not currently supported by SAS. Note that `\P{Any}` does not match any characters, so always causes a match failure.

Sets of Unicode characters are defined as belonging to certain scripts. A character from one of these sets can be matched using a script name. For example,

```
\p{Greek}
```

```
\P{Han}
```

Those that are not part of an identified script are lumped together as "Common". The current list of scripts includes the following languages.

Arabic, Armenian, Bengali, Bopomofo, Braille, Buginese, Buhid, Canadian_Aboriginal, Cherokee, Common, Coptic, Cypriot, Cyrillic, Deseret, Devanagari, Ethiopic, Georgian, Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul, Hanunoo, Hebrew, Hiragana, Inherited, Kannada, Katakana, Kharoshthi, Khmer, Lao, Latin, Limbu, Linear_B, Malayalam, Mongolian, Myanmar, New_Tai_Lue, Ogham, Old_Italic, Old_Persian, Oriya, Osmanya, Runic, Shavian, Sinhala, Syloti_Nagri, Syriac, Tagalog, Tagbanwa, Tai_Le, Tamil, Telugu, Thaana, Thai, Tibetan, Tifinagh, Ugaritic, and Yi

Each character has exactly one general category property, specified by a two-letter abbreviation. For compatibility with Perl, negation can be specified by including a circumflex between the opening brace and the property name. For example, `\p{^Lu}` is the same as `\P{Lu}`.

If only one letter is specified with `\p` or `\P`, it includes all the general category properties starting with that letter. In this case, in the absence of negation, the braces in the escape sequence are optional; these two examples have the same effect.

```
\p{L}
```

```
\pL
```

These general category property codes are supported:

Property Code	Description
C	Other
Cc	Control
Cf	Format
Cn	Unassigned
Co	Private use
Cs	Surrogate
L	Letter
Ll	Lowercase letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter

Property Code	Description
Lu	Uppercase letter
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
Nl	Letter number
No	Other number
P	Punctuation
Pc	Connector punctuation
Pd	Hyphen punctuation
Pe	Close punctuation
Pf	Final punctuation
Pi	Initial punctuation
Po	Other punctuation
Ps	Open punctuation
S	Symbol
Sc	Currency symbol
Sk	Modifier symbol
Sm	Mathematical symbol
So	Other symbol
Z	Separator

Property Code	Description
Zl	Line separator
Zp	Paragraph separator
Zs	Space separator

The special property `L&` is also supported; it matches a character that has the `Lu`, `Ll`, or `Lt` property (a letter that is not classified as a modifier or "other").

The long synonyms for these properties that Perl supports (such as, `\p{Letter}`) are not supported by SAS, and it is not permitted to prefix any of these properties with `Is`.

No character that is in the Unicode table has the `Cn` (unassigned) property. Instead, this property is assumed for any code point that is not in the Unicode table.

Specifying caseless matching does not affect these escape sequences. For example, `\p{Lu}` always matches only uppercase letters.

The `\X` escape matches any number of Unicode characters that form an extended Unicode sequence. The `\X` is equivalent to the following string.

```
(?>\PM\pM*)
```

That is, it matches a character without the "mark" property, followed by zero or more characters with the "mark" property, and treats the sequence as an atomic group. Characters with the "mark" property are typically accents that affect the preceding character.

Matching characters by Unicode property is not fast because SAS must search a structure that contains data for over fifteen thousand characters.

Simple Assertions

The fourth use for backslash is certain simple assertions. An assertion specifies a condition that must be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The backslashed assertions include:

Character	Description
\b	Matches at a word boundary
\B	Matches when not at a word boundary
\A	Matches at start of subject
\Z	Matches at end of subject or before newline at end
\z	Matches at end of subject

You cannot use these assertions in character classes, but note that \b has a different meaning, the backspace character, inside a character class.

A word boundary is a position in the subject string where the current character and the previous character do not both match \w or \W (one matches \w and the other matches \W) or the start or end of the string if the first or last character matches \w, respectively.

The \A, \Z, and \z assertions differ from the traditional circumflex and dollar in that they match at the beginning and end of the subject string. The difference between \Z and \z is that \Z matches before a newline at the end of the string as well as at the very end, whereas \z matches only at the end.

Circumflex and Dollar

Outside a character class, in the default matching mode, the circumflex character (^) is an assertion that is true only if the current matching point is at the beginning of the subject string. Inside a character class, circumflex has an entirely different meaning.

Circumflex need not be the first character of the pattern if a number of alternatives are involved, but it should be the first character in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex that is, if the pattern is constrained to match only at the start of the subject is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character ($) is an assertion that is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar does not be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

Full Stop (Period, Dot)

Outside a character class, a dot (.) in the pattern matches any one character in the subject string except (by default) a character that signifies the end of a line. The matched character can be more than one byte long. When the end of a line is defined as a single character (CR or LF), dot never matches that character. When the two-character sequence CRLF is used, dot does not match CR if it is immediately followed by LF. However, it matches all characters (including isolated CRs and LFs).

The handling of dot is entirely independent of the handling of circumflex and dollar. The only relationship is that they all involve newline characters. Dot has no special meaning in a character class.

Matching a Single Byte

Outside a character class, the escape sequence \C matches any one byte. Unlike a dot, it always matches CR and LF. The feature is provided in Perl in order to match individual bytes in UTF-8 mode. Since it breaks up UTF-8 characters into individual bytes, what remains in the string can be a malformed UTF-8 string. For this reason, the \C escape sequence is best avoided.

SAS does not allow \C to appear in look behind assertions, because this would make it impossible to calculate the length of the look behind for UTF-8 characters.

Square Brackets and Character Classes

An opening square bracket ([) introduces a character class, terminated by a closing square bracket (]). A closing square bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject. The character can occupy more than one byte. A matched character must be in the set of characters defined by the class, unless the first character in the class definition is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class `[aeiou]` matches any lowercase vowel and `[^aeiou]` matches any character that is not a lowercase vowel. Note that a circumflex is just a convenient notation for specifying the characters that are in the class by enumerating those that are not. It is not an assertion; it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

Characters with values greater than 255 can be included in a class as a literal string of bytes, or by using the `\x{` escaping mechanism.

When caseless matching is set, any letters in a class represent both their upper and lowercase versions. So, for example, a caseless `[aeiou]` matches `A` as well as `a`, and a caseless `[^aeiou]` does not match `A`, whereas a caseful version would.

Characters that might indicate line breaks (CR and LF) are never treated in any special way when matching character classes.

You can use the minus or hyphen (-) character to specify a range of characters in a character class. For example, `[d-m]` matches any letter between `d` and `m`, inclusive. If a minus character is required in a class, it must be escaped with a backslash or appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class.

It is not possible to have the literal character `]` as the end character of a range. A pattern such as `[W-]46]` is interpreted as a class of two characters (`W` and `-`) followed by a literal string `46]`, so it would match `W46]` or `-46]`. However, if the `]` is escaped with a backslash, it is interpreted as the end of range, so `[W-\]46]` is interpreted as a single class containing a range followed by two separate characters. You can also use the octal or hexadecimal representation of `]` to end a range.

Ranges operate in the collating sequence of character values. They can also be used for characters specified numerically for example, `[\000-\037]`. Ranges can include characters whose values are greater than 255 for example, `[\x{100}-\x{2ff}]`. If a range that includes letters is used when caseless matching is set, it matches the letters in either case. For example, `[W-c]` is equivalent to `[][\\^_`wxyzabc]`, matched caselessly.

The character types `\d`, `\D`, `\p`, `\P`, `\s`, `\S`, `\w`, and `\W` can appear in a character class, and add the characters that they match to the class. For example, `[\dABCDEF]` matches any hexadecimal character. A circumflex can conveniently be used with the uppercase character types to specify a more restricted set of characters than

the matching lowercase type. For example, the class `[^\W_]` matches any letter or digit, but not underscore.

The only meta-characters that are recognized in character classes are backslash, hyphen (where it can be interpreted as specifying a range), circumflex (at the start), opening square bracket (when it can be interpreted as introducing a POSIX class name, see POSIX Character Classes), and the terminating closing square bracket. However, escaping other non-alphanumeric characters does no harm.

POSIX Character Classes

Perl supports the POSIX notation for character classes. This uses names enclosed by `[:` and `:]` within the enclosing square brackets. SAS also supports this notation. For example, the string `[01[:alpha:]%]` matches "0", "1", any alphabetic character, or %. The supported class names are listed below.

Class Name	Description
alnum	letters and digits
alpha	letters
ascii	character codes 0 - 127
blank	space or tab only
cntrl	control characters
digit	decimal digits (same as \d)
graph	printing characters, excluding space
lower	lowercase letters
print	printing characters, including space
punct	printing characters, excluding letters and digits
space	white space (not quite the same as \s)
upper	uppercase letters
word	"word" characters (same as \w)
xdigit	Hexadecimal characters

The `space` characters are HT (9), LF (10), VT (11), FF (12), CR (13), and space (32). Notice that this list includes the VT character (code 11). This makes `space` different from `\s`, which does not include VT (for Perl compatibility).

The name `word` is a Perl extension, and `blank` is a GNU extension from Perl version 5.8. Another Perl extension is negation, which is indicated by a ^ character after the colon. For example, `[12[:^digit:]]` matches "1", "2", or any non-digit. SAS (and Perl) also recognize the POSIX syntax [.ch.] and [=ch=] where **ch** is a collating element, but these are not supported, and an error is given if they are encountered.

Vertical Bar

You can use the vertical bar character (`|`) to separate alternative patterns. For example, the pattern `gilbert|sullivan` matches either "gilbert" or "sullivan." Any number of alternatives can appear, and an empty alternative is permitted, matching the empty string. The matching process tries each alternative in turn, from left to right, and the first alternative that succeeds is used. If the alternatives are within a subpattern (defined below), **succeeds** means matching the rest of the main pattern as well as the alternative in the subpattern.

Internal Option Settings

Some matching options can be changed from within the pattern by a sequence of Perl option letters enclosed between `(?` and `)`. The option letters are listed below.

Option Letters	Description
i	for CASELESS
m	for MULTILINE
s	for DOTALL

For example, `(?im)` sets caseless, multi-line matching.

When an option change occurs at top level (that is, not inside subpattern parentheses), the change applies to the remainder of the pattern that follows. If the change is placed right at the start of a pattern, SAS extracts it into the global options for that expression.

An option change within a subpattern affects only that part of the current pattern that follows, so `(a(?i)b)c` matches abc and aBc and no other strings. By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example, `(a(?i)b|c)` matches "`ab`", "`aB`", "`c`", and "`C`", even though when matching `C` the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time (otherwise, there would be some strange behavior).

Subpatterns

Subpatterns are delimited by parentheses `(` and `)`, which you can nest. Marking part of a pattern as a subpattern does two things:

It localizes a set of alternatives. For example, the pattern `cat(aract|erpillar|)` matches one of the words "cat," "cataract," or "caterpillar." Without the parentheses, it would match "cataract," "erpillar," or the empty string.

It also sets up the subpattern as a capturing subpattern. When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller through the ovector argument `pcre_exec()`. Opening parentheses are counted from left to right (starting from 1) to obtain numbers for the capturing subpatterns.

For example, if the string "the red king" is matched against the pattern `the ((red|white) (king|queen))` the captured substrings are "red king," "red," and "king," and are numbered 1, 2, and 3, respectively.

The fact that parentheses fulfill two functions is not always helpful. There are often when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by a question mark and a colon `(?:`, the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string "the white queen" is matched against the pattern `the ((?:red|white) (king|queen))` the captured substrings are "white queen" and "queen", and are numbered 1 and 2. The maximum number of capturing subpatterns is 65535, and the maximum depth of nesting of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters can appear between the "`?`" and the "`:`." Thus, the two patterns `(?i:saturday|sunday) (?:(?i)saturday|sunday)` match exactly the same set of strings. Alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached. An option setting in one branch affects subsequent branches, so the above patterns match "SUNDAY" as well as "Saturday."

Repetition

Repetition is specified by quantifiers, which can follow any of the following items:

- a literal data character
- the `.` meta-character
- the `\C` escape sequence
- the `\X` escape sequence
- an escape such as `\d` that matches a single character
- a character class
- a back reference (see Atomic Grouping and Possessive Quantifiers on page 80)
- an enclosed in parentheses subpattern, unless it is an assertion

The general repetition quantifier specifies a minimum and maximum number of permitted matches by giving the two numbers in braces (`{` and `}`), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example, the string `z{2,4}` matches `zz`, `zzz`, or `zzzz`. A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit. If the second number and the comma are both omitted, the quantifier specifies an exact number of required

matches. Thus, `[aeiou]{3,}` matches at least three successive vowels, but can match many more, and `\d{8}` matches exactly eight digits. An opening brace that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, {,6} is not a quantifier, but a literal string of four characters.

Quantifiers apply to UTF-8 characters rather than to individual bytes. For example, `\x{100}{2}` matches two UTF-8 characters, and is represented by a two-byte sequence. Similarly, when Unicode property support is available, `\X{3}` matches three Unicode extended sequences, and might be several bytes long (and they can be of different lengths).

The quantifier `{0}` is permitted, causing the expression to behave as if the previous item and the quantifier were not present.

For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

- * is equivalent to {0,}
- + is equivalent to {1,}
- ? is equivalent to {0,1}

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit, like the string `(a?)*`. Earlier versions of Perl and SAS used to give an error for such patterns. However, because there are cases where this can be useful, such patterns are now accepted, but if any repetition of the subpattern does in fact does not match any characters, the loop is forcibly broken.

By default, the quantifiers match as much as possible (up to the maximum number of permitted times), without causing the rest of the pattern to fail. The classic example of where this causes problems is in trying to match comments in C programs. These appear between `/*` and `*/` and within the comment, individual `*` and **/** characters can appear. An attempt to match C comments by applying the pattern `/\*.*\*/` to this pattern `/* first comment */ not comment /* second comment */` fails, because it matches the entire string because of the .* item. However, if a quantifier is followed by a question mark, it does not match the maximum number, and instead matches the minimum number of times possible, so the pattern `/\*.*?\*/` does the right thing with the C comments. The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled, as in `\d??\d` it matches one digit by preference, but can match two if that is the only way the rest of the pattern matches. These quantifiers try to match the maximum number by default, but individual ones can be made to match a minimum by following them with a question mark. In other words, it inverts the default behavior.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after `(tweedle[dume]{3}\s*)+` has matched "tweedledum tweedledee" the value of the captured substring is "tweedledee". However, if there are nested capturing subpatterns, the corresponding captured values can have been set in previous iterations. For example, after `/(a|(b))+/` matches "aba" the value of the second captured substring is "b".

Atomic Grouping and Possessive Quantifiers

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to change the nature of the match, or to cause it to fail earlier, when the author of the pattern knows there is no point in carrying on.

For example, the pattern `\d+one` when applied to the subject line `123456two`, after matching all 6 digits, it fails to match "one", the normal action of the match is to try again with only 5 digits matching the `\d+` item, and then 4, and so on, before failing. "Atomic grouping" (a term used in "Mastering Regular Expressions" by Jeffrey Friedl) provides the means for specifying that once a subpattern has matched, it is not to be re-evaluated in this way.

If the atomic grouping for the previous example is used, the matcher gives up immediately on failing to match "one" the first time. The notation is a type of special parenthesis, starting with `(?>` as in the example `(?>\d+)one`. This type of parenthesis "locks up" the part of the pattern that it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern matches, if anchored at the current point in the subject string.

Atomic grouping subpatterns are not capturing subpatterns. Simple cases such as the above example can be thought of as a maximizing repeat that must swallow everything it can. So, both `\d+` and `\d+?` are prepared to adjust the number of digits they match in order to make the rest of the pattern match, `(?>\d+)` matches an entire sequence of digits.

Atomic groups in general can contain arbitrarily complicated subpatterns, and can be nested. However, when the subpattern for an atomic group is just a single repeated item, as in the example above, a simpler notation, called a "possessive quantifier" can be used. This consists of an additional + character following a quantifier. Using this notation, the previous example can be rewritten as `\d++one`.

Possessive quantifiers are always greedy. They are a convenient notation for the simpler forms of atomic group. However, there is no difference in the meaning or processing of a possessive quantifier and the equivalent atomic group. The possessive quantifier syntax is an extension to the Perl syntax.

When a pattern contains an unlimited repeat inside a subpattern that can itself be repeated an unlimited number of times, the use of an atomic group is the only way to avoid some failing matches taking a very long time. The pattern `(\D+|<\d+>)*[!?]` matches an unlimited number of substrings that either consist of non-digits, or digits enclosed in `<>`, followed by either `!` or `?`. When it matches, it runs quickly. However, if it is applied to the string, `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa` it takes a long time before reporting failure. The string can be divided between the internal `\D+` repeat and the external `*` repeat in a large number of ways, and all options are attempted. (The example uses `[!?]` rather than a single character at the end, because both SAS and Perl have an optimization that allows for fast failure when a single character is used. They remember the last single character that is required for a match, and fail early if it is not present in the string.)

If the pattern is changed so that it uses an atomic group, like `((?>\D+)|<\d+>)*[!?]` sequences of non-digits cannot be broken, and failure happens quickly.

Back References

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. A "forward back reference" of this type can make sense when a repetition is involved and the subpattern to the right has participated in an earlier iteration. See Backslash for more information about the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. See Subpatterns as Subroutines on page 113 for more information.). So the pattern `(sens|respons)e and \1ibility` matches "sense and sensibility" and "response and responsibility," but not "sense and responsibility." If caseful matching is in force at the time of the back reference, the case of letters is relevant. For example, `((?i)rah)\s+\1` matches "rah rah" and "RAH RAH," but not "RAH rah," even though the original capturing subpattern is matched caselessly.

There can be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, any back references to it always fail. For example, the pattern `(a|(bc))\2` always fails if it starts to match `a` rather than `bc`. Because there can be many capturing parentheses in a pattern, all digits following the backslash are taken as part of a potential back reference number. If the pattern continues with a digit character, some delimiter must be used to terminate the back reference. Here an empty comment (see Comments) can be used.

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used. So, for example, `(a\1)` never matches. However, such references can be useful inside repeated subpatterns. For example, the pattern `(a|b\1)+` matches any number of `a`'s and also `aba`, `ababbaa`, and so on. At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. For this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of 0.

Assertions

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as `\b`, `\B`, `\A`, `\Z`, `\z`, `^`, and `$` are described above. More complicated assertions are

coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except it does not cause the current matching position to be changed. An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed.

Assertion subpatterns are not capturing subpatterns, and cannot be repeated, because it makes no sense to assert the same thing several times. If any type of assertion contains capturing subpatterns within it, these are counted for the purposes of numbering the capturing subpatterns in the whole pattern. However, substring capturing is carried out only for positive assertions, because it does not make sense for negative assertions.

Look Ahead Assertions

Look ahead assertions start with `(?=` for positive assertions and `(?!` for negative assertions. For example, `\w+(?=;)` matches a word followed by a semicolon, but does not include the semicolon in the match, and `one(?!two)` matches any occurrence of "one" that is not followed by "two." Note that the apparently similar pattern, `(?!one)two` does not find an occurrence of "two" that is preceded by something other than "one". It finds any occurrence of "two", because the assertion `(?!one)` is always true when the next three characters are "two." A look behind assertion is needed to achieve this effect.

If you want to force a matching failure at some point in a pattern, the most convenient way to do it is with `(?!)` because an empty string always matches, so an assertion that requires there not to be an empty string must always fail.

Look Behind Assertions

Look behind assertions start with `(?<=` for positive assertions and `(?<!` negative assertions. For example, `(?<!one)two` does find an occurrence of "two" that is not preceded by "one." The contents of a look behind assertion are restricted so that all the strings it matches must have a fixed length. However, if there are several alternatives, they do not all have to have the same fixed length. For example, `(?<=leopard|donkey)` is permitted, but `(?<!dogs?|cats?)` causes an error at compile time. Branches that match different length strings are permitted only at the top level of a look behind assertion. This is an extension compared with Perl (for version 5.8), which requires all branches to match the same length of string. An assertion such as, `(?<=ab(c|de))` is not permitted because its single top-level branch can match two different lengths, but it is acceptable if rewritten to use two top-level branches `(?<=abc|abde)`.

The implementation of look behind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail.

SAS does not allow the `\C` escape to appear in look behind assertions, because it makes it impossible to calculate the length of the look behind for UTF-8 characters. The `\X` escape, which can match different numbers of bytes, is also not permitted.

Atomic groups can be used in conjunction with look behind assertions to specify efficient matching at the end of the subject string. Consider a simple pattern such as `abcd$` when applied to a long string that does not match. Because matching

proceeds from left to right, SAS looks for each `a` in the subject and then sees what follows matches the rest of the pattern. If the pattern is specified as `^.*abcd$`, the initial `.*` matches the entire string at first, but when this fails (because there is no following `a`), it backtracks to match all but the last character, and then all but the last two characters, and so on. Once again the search for `a` covers the entire string, from right to left, so it is no better. However, if the pattern is written as `^(?>.*)(?<=abcd)` or, equivalently, using the possessive quantifier syntax, `^.*+(?<=abcd)` there can be no backtracking for the `.*` item; it can match only the entire string. The subsequent look behind assertion does a single test on the last four characters. If it fails, the match fails immediately. For long strings, this approach makes a significant difference to the processing time.

Using Multiple Assertions

Several assertions (of any sort) can occur in succession. For example, `(?<=\d{3})(?<!999)one` matches "one" preceded by three digits that are not 999. Notice that each of the assertions is applied independently at the same point in the subject string. First, there is a check that the previous three characters are all digits, and then there is a check that the same three characters are not `999`. This pattern does not match "one" preceded by six characters, the first of which are digits and the last three of which are not `999`. For example, it does not match `123abcone`. A pattern to do that is `(?<=\d{3}...)(?<!999)one`. This time, the first assertion looks at the preceding six characters, checking that the first three are digits, and then the second assertion checks the preceding three characters are not `999`.

Assertions can be nested in any combination. For example, `(?<=(?<!one)two)cat` matches an occurrence of "cat" that is preceded by "two". Here, it is not preceded by "one" and `(?<=\d{3}(?!999)...)one` is another pattern that matches "one" preceded by three digits and any three characters that are not `999`.

Conditional Subpatterns

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern are:

```
(?(condition)yes-pattern)
```

```
(?(condition)yes-pattern|no-pattern)
```

If the condition is satisfied, the yes-pattern is used; otherwise the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are three types of conditions. If the text between the parentheses consists of a sequence of digits, or a sequence of alphanumeric characters and underscores, the condition is satisfied, if the capturing subpattern of that number or name previously matched. There is a possible ambiguity here, because subpattern names can consist entirely of digits. If a named subpattern cannot locate one and the text consists entirely of digits, it looks for a subpattern of that number, which must be greater than zero. Using subpattern names that consist entirely of digits is not recommended.

Consider the following pattern, which contains non-significant white space (to make it more readable and to divide it into three parts for ease of discussion).

```
( \( )? [^()]+ (?(1) \) )
```

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern that tests whether the first set of parentheses matched or not. If they did, if the subject started with an opening parenthesis, the condition is true, and the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses. Rewriting it to use a named subpattern results in the following pattern.

```
(?P \( )? [^()]+ (?(OPEN) \) )
```

If the condition is the string `(R)`, and there is no subpattern with the name `R`, the condition is satisfied if a recursive call to the pattern or subpattern has been made. At "top level", the condition is false. This is a SAS extension. Recursive patterns are described in the next section.

If the condition is not a sequence of digits or `(R)`, it must be an assertion. This can be a positive or negative look ahead or look behind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line.

```
(?(?=[^a-z]*[a-z])
\d{2}-[a-z]{3}-\d{2} | \d{2}-\d{2}-\d{2} )
```

The condition is a positive look ahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms `dd-aaa-dd` or `dd-dd-dd`, where aaa are letters and dd are digits.

Comments - The sequence `(?#` marks the start of a comment that continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching.

Recursive Patterns

Consider the problem of matching a string in parentheses, allowing for unlimited nested parentheses. Without the use of recursion, the best that can be done is to use a pattern that matches up to some fixed depth of nesting. It is not possible to handle an arbitrary nesting depth. Perl provides a facility that allows regular expressions to recurse (among other things). It does this by interpolating Perl code in the expression at run time, and the code can refer to the expression itself. A Perl pattern to solve the parentheses problem can be created like this.

```
$re = qr{\( (?: (?>[^()]+) | (?p{$re}) )* \)}x;
```

The `(?p{...})` item interpolates Perl code at run time, and in this case refers recursively to the pattern in which it appears. SAS cannot support the interpolation of Perl code. Instead, it supports special syntax for recursion of the entire pattern, and for individual subpattern recursion.

The special item that consists of `(?` followed by a number greater than zero and a closing parenthesis is a recursive call of the subpattern of the given number,

provided that it occurs inside that subpattern. (If not, it is a "subroutine" call, which is described in Subpatterns as Subroutines on page 113). The special item `(?R)` is a recursive call of the entire regular expression.

A recursive subpattern call is always treated as an atomic group. That is, once it has matched some of the subject string, it is never re-entered, even if it contains untried alternatives and there is a subsequent matching failure.

This pattern solves the nested parentheses problem.

```
\( ( (?>[^()]+) | (?R) )* \)
```

First it matches an opening parenthesis. Then it matches any number of substrings, which can either be a sequence of non-parentheses, or a recursive match of the pattern itself (that is, a correctly enclosed in parentheses substring). Finally, there is a closing parenthesis.

If this were part of a larger pattern, you would not want to recurse the entire pattern, so instead you could use this.

```
( \( ( (?>[^()]+) | (?1) )* \) )
```

The pattern is in parentheses, and causes the recursion to refer to them instead of the whole pattern. In a larger pattern, keeping track of parenthesis numbers can be tricky. It can be more convenient to use named parentheses instead. For this, SAS uses `(?P>name)`, an extension to the Python syntax that SAS uses for named parentheses (Perl does not provide named parentheses). You could rewrite the above example as follows.

```
(?P \( ( (?>[^()]+) | (?P>pn) )* \) )
```

This particular example pattern contains nested unlimited repeats. Therefore, the use of atomic grouping for matching strings of non-parentheses is important when applying the pattern to strings that do not match. For example, when this pattern is applied to the following pattern.

```
(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa()
```

it yields "no match" quickly. However, if atomic grouping is not used, the match runs for a very long time because there are so many different ways the `+` and `*` repeats can separate the subject, and all have to be tested before failure can be reported.

At the end of a match, the values set for any capturing subpatterns are those from the outermost level of the recursion at which the subpattern value is set. If you want to obtain intermediate values, a callout function can be used. See Subpatterns as Subroutines on page 113 and the precallout documentation for additional information. If the pattern above is matched against `(ab(cd)ef)` the value for the capturing parentheses is `ef`, which is the last value taken on at the top level. If additional parentheses are added, giving `\( ( ( (?>[^()]+) | (?R) )* ) \)` the string that they capture is `ab(cd)ef`, the contents of the top level parentheses. If there are more than 15 capturing parentheses in a pattern, SAS has to obtain extra memory to store data during a recursion. If no memory can be obtained, the match fails with an error.

Do not confuse the `(?R)` item with the condition `(R)`, which tests for recursion. Consider this pattern, which matches text in angle brackets, allowing for arbitrary nesting. Only digits are allowed in nested brackets during recursion, whereas any characters are permitted at the outer level.

```
< (?: (?(R) \d++ | [^<>]*+) | (?R)) * >
```

In this pattern, `(?(R)` is the start of a conditional subpattern, with two different alternatives for the recursive and non-recursive cases. The `(?R)` item is the actual recursive call.

Subpatterns as Subroutines

If the syntax for a recursive subpattern reference (either by number or by name) is used outside the parentheses to which it refers, it operates like a subroutine in a programming language. An earlier example demonstrated that the pattern, `(sens|respons)e and \1ibility` matches "sense and sensibility" and "response and responsibility", but not "sense and responsibility". If the pattern `(sens|respons)e and (?1)ibility` is used, it matches "sense and responsibility" as well as the other two strings. Such references, if given numerically, must follow the subpattern to which they refer. However, named references can refer to later subpatterns.

Like recursive subpatterns, a "subroutine" call is always treated as an atomic group. That is, once it has matched some of the subject string, it is never re-entered, even if it contains untried alternatives and there is a subsequent matching failure.

Unicode Support

Special information about support of Unicode characters is given below:

1. An unbraced hexadecimal escape sequence (such as `\xb3`) matches a two-byte UTF-8 character if the value is greater than 127.
2. Octal numbers up to `\777` are recognized, and match two-byte UTF-8 characters for values greater than `\177`.
3. Repeat quantifiers apply to complete UTF-8 characters, not to individual bytes, for example: `\x{100}{3}`.
4. The dot (.) meta-character matches one UTF-8 character instead of a single byte.
5. The escape sequence `\C` can be used to match a single byte in UTF-8 mode, but its use can lead to some strange results.
6. Similarly, characters that match the POSIX named character classes are all low-valued characters.

Differences from Perl

This section describes the differences in the ways that SAS and Perl handle regular expressions. The differences described here compare SAS with Perl version 5.8.

1. SAS has only a subset of the Perl UTF-8 and Unicode support. Details of what it does have are given above.

2 SAS does not allow repeat quantifiers on look ahead assertions. Perl permits them, but they do not mean what you might think. For example, `(?!a){3}` does not assert that the next three characters are not "a". It asserts that the next character is not "a" three times.

3 Capturing subpatterns that occur inside negative look ahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative look ahead assertion contains just one branch.

4 Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence `\0` can be used in the pattern to represent a binary zero.

5 The properties that can be tested with `\p` and `\P` are limited to the general category properties such as `Lu` and `Nd`, script names such as `Greek` or `Han`, and the derived properties `Any` and `L&`.

6 SAS does support the `\Q...\E` escape for quoting substrings. Characters in between are treated as literals. This is slightly different from Perl in that `$` and `@` are also handled as literals inside the quotation marks. In Perl, they cause variable interpolation (but of course SAS does not have variables). Note the examples in the table below.

Note: The `\Q...\E` sequence is recognized both inside and outside character classes.

Pattern	SAS Matches	Perl Matches
\Qabc$xyz\E	abc$xyz	abc followed by the contents of $xyz
\Qabc\$xyz\E	abc\$xyz	abc\$xyz
\Qabc\E\$\Qxyz\E	abc$xyz	abc$xyz

7 SAS does not support the `(?{code})` and `(?p{code})` constructions. However, there is support for recursive patterns using the non-Perl items `(?R)`, `(?number)`, and `(?P>name)`.

8 There are some differences that are concerned with the settings of captured strings when part of a pattern is repeated. For example, matching "`aba`" against the pattern `/^(a(b)?)+$/` in Perl leaves `$2` unset, but in SAS it is set to "`b`".

9 SAS provides some extensions to the Perl regular expression facilities:

   a Although look behind assertions must match fixed length strings, each alternative branch of a look behind assertion can match a different length of string. Perl requires them all to have the same length.

b Where a backslash is followed by a letter with no special meaning, the backslash is always ignored (Perl can be made to issue a warning).

c The greediness of the repetition quantifiers is inverted; that is, by default they are not greedy, but if followed by a question mark that they are.

d The `(?R)`, `(?number)`, and `(?P>name)` constructs allows for recursive pattern matching (Perl can do this using the `(?p{code})` construct, which PCRE cannot support.)

e SAS supports the possessive quantifier "++" syntax, taken from the Sun© Java© package.

Scheme Builder

You can use Scheme Builder to create custom standardization schemes. There are a number of ways to display the Scheme Builder. Some of these methods are as follows:

- Create a new scheme or open an existing scheme from the **Schemes** tab, as described in Navigating the Schemes Tab on page 124.
- From the main menu, select **Tools** > **Other QKB Editors** > **Scheme Builder**.
- Open an existing profile. Click the **Report** tab. Select a table in the navigation tree on the left. Click the **Standard Metrics** tab on the right. Right-click a field for which you want to define a custom scheme, and then select **Scheme Builder**.

Scheme Builder Editor

You can use Scheme Builder to create custom standardization schemes. There are a number of ways to display the Scheme Builder. Some of these methods are as follows:

- Create a new scheme or open an existing scheme from the **Schemes** tab, as described in Navigating the Schemes Tab on page 124.
- From the main menu, select **Tools** > **Other QKB Editors** > **Scheme Builder**.
- Open an existing profile. Click the **Report** tab. Select a table in the navigation tree on the left. Click the **Standard Metrics** tab on the right. Right-click a field for which you want to define a custom scheme, and then select **Scheme Builder**.

Menu

Following are descriptions of the menus on the **Profile - Viewer Scheme Builder** screen. You can access this screen from the **Profile - Viewer** main screen by selecting a field for which you want to build a scheme, and then choosing **Tools** > **Scheme Builder**.

File

New - Start a new scheme.

Open - Retrieve an existing scheme.

Close - Close the current scheme.

Save - Save the current scheme.

Save As - Save the current scheme with a new file name.

Note: Schemes are saved with a `.qkb` extension. Any scheme created in earlier versions are saved as `.qkb` file when it is open and saved. If you save this file in the new format, you are not be able to access the scheme in earlier versions of **DataFlux Data Management Studio**. Click **Yes** to save the scheme under the new format or **No** if you do not want to save in the new format.

Print - Print the current scheme.

Import From A Text File - Import a scheme using the Import From Text File screen.

Import N-Grams - Import N-Grams from a text file. N-Grams are used to analyze a string and guess its language. For more information, see Using N-Grams in Language Guess Definitions. on page 250.

Export to Text File - Export the current scheme as a scheme text file.

Export to Excel Worksheet - Export the current scheme as a Microsoft Excel worksheet.

Exit - Close the Scheme Builder screen.

Edit

Add - Add an entry to the current scheme.

Edit - Edit the selected scheme entry.

Delete - Delete the selected scheme entry.

Clear - Clear the selected scheme entry.

Paste - Paste into the **Standard** box at the bottom of the screen the latest string copied to the Windows Clipboard.

Find In Data - Use the **Find** screen to find a string in your data.

Find In Standard - Use the Find screen to find a string in the current scheme.

Sort By Data - Sort the current scheme alphanumerically by the data.

Sort By Standard - Sort the current scheme alphanumerically by the standard.

Reset - Clear the current scheme.

Build Scheme - Use Smart Clustering Data Analysis results to attempt to build a scheme automatically. This can serve as a starting point and greatly reduce your manual scheme building effort.

Modify Standards Manually - Specify whether to manually modify a single instance or all instances.

View

Toolbar - Show or hide the toolbar.

Status Bar - Show or hide the status bar.

Report

Load Report - Retrieve a previously saved Scheme Builder report.

Save Report - Save the current Scheme Builder report.

Find In Report - Use the Scheme Builder **Find** screen to find a string in the current report.

Add Selection to Scheme - After performing a data analysis, you can select multiple data permutations in the results. This option adds all selected permutations to the current scheme, along with the value in the **Standard** box at the bottom of the screen.

Set Current as Standard - Set the selected permutation as the standard value of the corresponding data element(s) selected in the Scheme Builder report for the scheme entry.

Compare Report to Scheme - Use these options when you want to compare the data from the current report to a scheme. They are enabled only when a scheme has been loaded.

Hide Existing Permutations - Hides any data in the report that is already contained in the scheme. You see only the data that has not already been added.

Highlight Unaccounted Permutations - When available, highlights all the data in the report that has not already been added to the scheme, in order to quickly identify what data has (or has not) been accounted for.

Print Report - Print the current Scheme Builder report.

Export Report - Export the current Scheme Builder report as a text file.

Sort Mode

Alphabetically - Sorts data results alphabetically.

By Occurrence - Sorts data results by occurrence.

Permutation Drill Down - Display the records in the current database that contain the selected data-analysis permutation.

Tools

Set QKB Locale - Set your QKB to the appropriate locale.

Note: To select a locale, you must have that locale installed.

Merge Schemes - Merge two existing schemes into a third scheme.

Generate SQL - Create a file of SQL commands that can reproduce the effects of using the current scheme.

Options - Use the **Options** screen to set Scheme Builder options.

Find

Following are descriptions of the items on the **Profile - Viewer Scheme Builder Find** screen. You can access this screen from the **Scheme Builder** screen by choosing **Edit** > **Find In Data**, **Edit** > **Find in Standard**, or **Report** > **Find in Report**.

Find What - Enter the string that you want to find.

Match Whole Word Only - Specify that you want to find your string only when it occurs as a whole word. For example, with this option selected, the string "dat" would find "dat," but not "data" or "date."

Match Case - Specify that you want to find your string only when it has the same case. For example, with this option selected, the string "data" would find "data," but not "Data."

Direction - Set the direction of the search: Up or Down.

Import from Text File

Following are descriptions of the items on the **Scheme Builder Import From Text File** screen. You can access this screen from the **Scheme Builder** screen by choosing **File** > **Import From Text File**.

Import File Name - Specify the name and location of your scheme text file. You can enter the file path and name manually (for example, c:\myfiles\myscheme.txt), or click the Browse icon to browse for the file.

Text Qualifier - Specify what character, if any, the Scheme Builder should expect at the beginning and end of each text field value in the scheme text file. The two available qualifiers are single quotation mark (') and double quotation mark (").

Note: If your scheme text file was created by another program, check that program's documentation to see what text qualifier, if any, it might have used. Or, look at the file directly in a text editor such as Microsoft Notepad.

Number of Rows to Skip - If your scheme text file has introductory or header information, specify the number of introductory lines to skip before the Scheme Builder starts reading the scheme text file's content as data.

Encoding - Specify the type of encoding used in the scheme text file.

Field Delimiter - Specify the type of separator (delimiter) used in the scheme text file for separating data fields.

Note: If your scheme text file was created by another program, check that program's documentation to see what field delimiter it might have used. Or, look at the file directly in a text editor such as Notepad

Options for Profiles

The following options are used when a scheme is applied in the context of a DataFlux Data Management Studio profile. In the **Report** tab for a profile, you can select a table in the tree on the left. A set of columns in that table is displayed in the **Standard Metrics** tab on the right. If you right-click a column, one of the options is **Build a Scheme**.

You can access the options from the Scheme Builder window by choosing **Tools** > **Options**.

Add Every Report Permutation when Using Build Scheme - Include at least one data (standard) rule for every value in the analysis report, even for values that are not part of matching clusters identified in the scheme build process.

Include Group Number when Printing and Exporting Reports - Entries that group together in the same cluster carry the same group number when printing and exporting to a text file.

Maximum Number of Rows to Display in the Drill Through - The number of rows to display in the permutation drill through.

Options for Jobs

The following options are used when a job node, such as the Standardization node, applies a scheme in the context of a job. Click **Options** at lower right of the Scheme Builder window.

You can select options that control how your scheme is applied to your data. The options that you select are stored in the scheme file so that they can be used in any data job that applies your scheme.

Note: Some options can be overridden in the Standardization node user interface.

Type Options

Select **Element** or **Phrase** to control whether a lookup is performed on individual elements of an input string or only on the entire input string as a whole. If you select **Element**, individual elements of an input string can be transformed by a scheme entry. For example, suppose you have the following entry in your scheme.

Data	Standard
MISTER	MR

If you apply this scheme to the input string MISTER JONES with the **Element** option selected, the resulting output is MR JONES.

If you select **Phrase**, the entire input string must match an entry in your scheme in order for the input string to be transformed. For example, , if you apply the example scheme above to MISTER JONES with the **Phrase** option selected, the output is MISTER JONES. In other words, no transformation occurs. In order for a transformation to occur with the **Phrase** option selected, your scheme would need to contain the following entry:

Data	Standard
MISTER JONES	MR JONES

Match Definition Options

Instead of using a simple exact-match lookup with your scheme, you can do a fuzzy lookup using a Match Definition. A match definition is a callable object in the QKB. A match definition is used to generate matchcodes, which are used for fuzzy matching. There are match definition available for use with different types of data.

If you select a match definition in your **Scheme Options**, a matchcode is generated for each entry in your scheme when the scheme is applied to your data in a data job. A matchcode is also generated for each input string to which the scheme is applied. If the matchcode for an input string is the same as the matchcode for a data entry in your scheme, the input string is replaced with the standard associated with that data entry. This means that the input string does not need to be identical to an entry in your scheme in order for a standard to be applied; the input string needs only to be a fuzzy match for an entry in your scheme.

For example, your scheme might include an entry such as this one:

Data	Standard
GREYWOOD HLDG	GREYWOOD HOLDINGS LLC

If your input string is GREYWOOD HLDG, an exact match does not occur when you apply the scheme to your data. But if you select a match definition in your **Scheme Options**, like Organization Match Definition GREYWOOD HLDG might still be deemed to match GREYWOOD HLDG. Therefore, GREYWOOD HLDG is replaced with the standard GREYWOOD HOLDINGS LLC.

Of course, if you use a match definition in your **Scheme Options**, there is a risk that you encounter false positives – inadvertent transformations that occur due to unwanted matches. It is recommended that you test your data job carefully on a data sample before deploying it to a production environment.

To use a match definition with your scheme, select a match definition in the **Name** drop box under the match definition area in the **Scheme Options**. The **Name** drop box shows all of the match definition that are available in your active QKB. For information about how to set up an active QKB, see Making a QKB Active on page 147.

After you have selected a Match Definition, choose a **Sensitivity value**. The **Sensitivity** controls the amount of fuzziness that is allowed when the scheme is applied to your data. The higher the Sensitivity, the more closely an input string must match an entry in the scheme in order for that input string to be replaced with the standard for that entry. You can experiment with different Sensitivity values before deploying your Data Job to a production environment.

Case sensitive - Select this check box if you want for data entries to be looked up in your scheme in a case-sensitive manner.

Trim whitespace in data - Select this check box if you want scheme lookups to be insensitive to leading and trailing whitespaces and/or the number of whitespaces between words. For example, suppose you add the following entry in your scheme:

Data	Standard
FOO BAR	BAZ

Note: There are two spaces before the word FOO and three spaces between FOO and BAR. If the **Trim whitespace in data** check box is selected, then the scheme is saved, this entry is converted.

Data	Standard
FOO BAR	BAZ

Leading and trailing whitespaces are removed, and multiple spaces between words are collapsed. Now when you apply this scheme to the input string FOO BAR, the result is BAZ.

Allow duplicate data - Select this checkbox if you want to be able to add multiple entries in your scheme with the same data value. You can use this option in combination with a Match Definition. For example:

Data	Standard
PRINCIPL	PRINCIPAL
PRINCIPL	PRINCIPLE

If you use a Match Definition when applying a scheme with the above entries, an input string containing a misspelling of either PRINCIPAL or PRINCIPLE could match both of these entries. If this happens, both matching entries are examined, and the standard that is closest to the value in the input string is chosen to replace that value. For example, if the input string is PRINSIPAL SYSTEMS or PRINNCIPAL SYSTEMS, then the output is PRINCIPAL SYSTEMS. But if the input is PRINSIPLE SYSTEMS or PRINNCIPLE SYSTEMS, then the output is PRINCIPLE SYSTEMS.

Note: This is an advanced option that is typically used only by users who have in-depth knowledge of phonetics and other aspects of the Match Definition that they have selected.

For more information about **Match Definitions** and **Advanced Options**, refer to the DataFlux Data Management Studio: User's Guide.

Adding and Editing Schemes

This topic describes how to add or edit a Transformation Scheme in your QKB. A Transformation Scheme is a file that maps text values to preferred standard representations of those values. Transformation Schemes are commonly referred to simply as "Schemes".

Add a new Scheme or update an existing Scheme when the current Schemes in your QKB do not produce the output that you desire . To understand Schemes, see the help for the Scheme Builder on page 115. See also the existing schemes on the **Schemes** tab.

Add a New Scheme

Perform these steps to add a new scheme to a locale in a QKB. It is assumed that you have opened the QKB from the **Administration Riser**, and the **Schemes** tab is displayed.

1 Click **New Scheme** on the lower toolbar, or select **New Scheme** from the menu on the lower toolbar. The **New Scheme** wizard opens.

2 Enter a name and description for the new scheme and click **Next**.

3 Select a locale for the scheme and click **Finish**. The new scheme appears in the scheme list, and the Scheme Builder opens.

4 Add one or more entries to the scheme. To add a new entry, select **Add** at the bottom of the window. Note that entries in a scheme are sorted automatically into alphabetic order; there is no need to order the entries manually.

5 To edit or delete an entry in the Scheme Builder, select an entry, and then the appropriate button at the bottom of the window.

6 You can also import data entries on page 118 from a text file.

7 If desired, set option values on page 119 for how your scheme is applied if it is applied in a DataFlux Data Management Studio data job node.

8 When you are satisfied with your edits, save the scheme. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.

9 Click **Yes** to reload the QKB.

Edit a Scheme

It is usually best to copy an existing scheme and modify the copy, rather than modify the original. If you modify an existing scheme, you might break a QKB definition or Data Management Studio Data Job that uses that scheme. Perform these steps to edit a scheme in a locale in a QKB. It is assumed that you have opened the QKB and locale from the **Administration Riser**, and the **Schemes** tab is displayed.

1 To open a scheme for editing, double-click on the scheme in the list, or select the scheme and click **Open** on the lower toolbar. You might also select the scheme and choose **Open** in the pop-up menu or in the menu on the lower toolbar. The scheme opens in the Scheme Builder.

2 Add one or more entries to the scheme. To add a new entry to the scheme, select **Add** at the bottom of the window. Note that entries in a scheme are sorted automatically into alphabetic order; there is no need to order the entries manually.

3 To edit or delete an entry in the Scheme Builder, select an entry, and then the appropriate button at the bottom of the window.

4 You can also import data entries on page 118 from a text file.

5 If desired, set option values on page 119 for how your scheme is applied if it is applied in a **DataFlux Data Management Studio** data job node.

6 When you are satisfied with your edits, save the scheme. If DataFlux Data Management Studio is open, you are prompted to reload the QKB.

7 Click **Yes** to reload the QKB.

Meta-operators

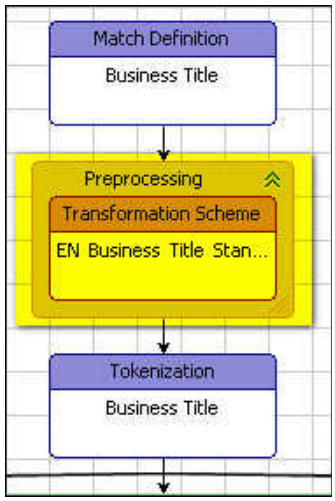
The following meta-operators can be specified in the **Standard** column to produce special behavior when the corresponding value in the **Data** column is encountered:

//Remove	Removes the data entry if it is found in the input string.
%	Ignores the data entry if it is found in the input string. This is a way to indicate that the author of the scheme considered the data value and determined that it should not be transformed.

Navigating the Scheme Editor

A transformation scheme, or simply a "scheme," is a type of QKB file that can be used to transform text data. A scheme contains a list of text values, along with preferred representations for those values. If a text value in the scheme matches a text value in an input data string, the text value in the input data string is replaced with the preferred representation. You can apply schemes to your data in a data job in DataFlux Data Management Studio. You can also use schemes in a QKB definition.

To see how you can use a scheme in a definition, right-click a definition in the **Quality Knowledge Base** tab and select **Open**. The **Definition Editor** displays a flow diagram for that definition. In most types of definitions you have one or more **Transformation Scheme** nodes. For example, the next display shows part of the diagram for the Business Title match definition in the English locale. You can see that there is a **Transformation Scheme** node that uses the EN Business Title Standards 2 scheme.



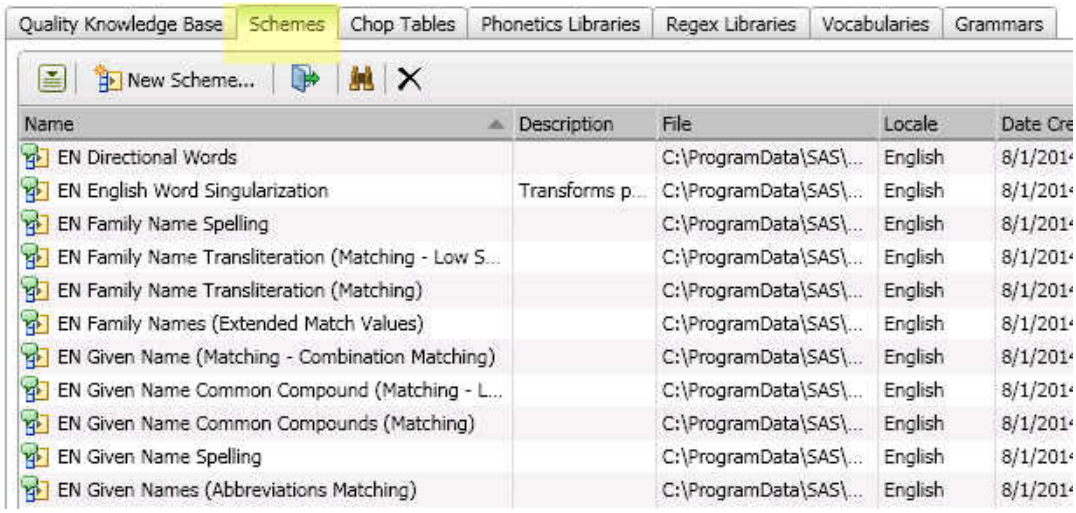
The next display shows some of the transformations for the EN Business Title Standards 2 scheme.

Scheme Entries: 49

Data	Standard
CO OWNER	CO-OWNER
COOWNER	CO-OWNER
CHIEF OPERATING OFFICER	COO
CHIEF OPERATIONS OFFICER	COO
CHIEF TECHNICAL OFFICER	CTO
DQ	DATA QUALITY
DATAWAREHOUSE	DATA WAREHOUSE
DW	DATA WAREHOUSE
DW ARCHITECT	DATA WAREHOUSE ARCHITECT
DATABASE ADMINISTRATOR	DBA
F & B	F&B
F&B	F&B
G M	GENERAL MANAGER
GEN MANAGER	GENERAL MANAGER
GENERAL MANAGER	GENERAL MANAGER
HR	HUMAN RESOURCES

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right-hand side of the screen. The **Schemes** tab shows a list of schemes that are contained in the QKB.

The list of schemes in the **Scheme** tab has several columns, as shown in the next display.



The **Name** column shows the name of a scheme. The **Description** column contains text about the scheme, such as "Transforms plural forms of words to singular forms." The **File** column contains the physical path to the scheme file.

The **Locale** column shows the QKB locale with which the scheme is associated. Note that a scheme might be listed even if it is not associated with the locale that you have selected in the **Quality Knowledge Bases** tree. This is because locales can inherit schemes from ancestors on page 163.

Select a locale and view at the values in the **Locale** column for a scheme, some schemes are associated directly with the selected locale. Others are associated with the locale's language, and still others are associated with the Global set of objects. This information is also presented in the icons that are shown for the schemes in the list. As shown in the previous display, if an icon has a flag overlay,

the scheme is associated directly with a locale. If the icon has a word balloon overlay, it means that the scheme is associated with a language. An earth overlay means that the scheme belongs to the Global set of schemes.

If you want to view only the schemes that are associated with a language, select that language in the tree for your QKB in the **Administration** riser.

If you want to view only the schemes in the Global set, click **Global** in the tree for your QKB.

Find a Scheme by Name - To find a scheme with a specific name, click **Find** in the lower toolbar. This toggles on the **Find** pane. To toggle off the **Find** pane, click the **Find** icon again.

Rename and Duplicate Scheme - You can rename a scheme by right-clicking the name and selecting **Rename** from the pop-up menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a scheme by right-clicking the name and selecting **Duplicate** from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Scheme - To create a new scheme, click **New Scheme** on the lower toolbar, or select **New Scheme** from the menu on the lower toolbar. After you have created the new scheme, the scheme is shown in the scheme list, and the Scheme Builder appears. For more information, see Adding and Editing Schemes n Scheme Bulider on page 122.

Open a Scheme for Editing - To open a scheme for editing, double-click on the scheme in the list, or select the scheme and click **Open**. You can also select the scheme and choose **Open** in the pop-up menu or in the menu on the lower toolbar. For more information, see Adding and Editing Schemes in Scheme Builder on page 122.

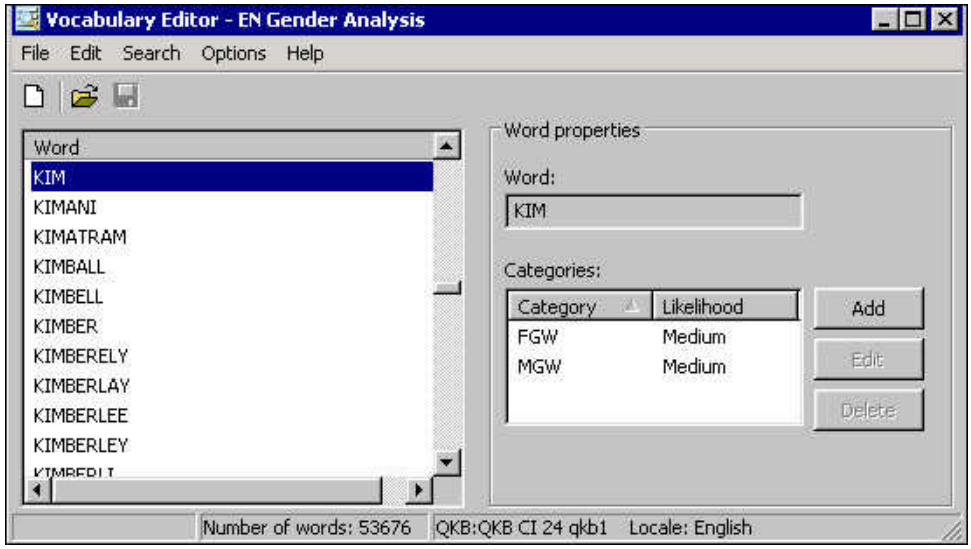
View Definitions That Use a Given Scheme - To view a list of QKB definitions that use a given scheme, select the scheme and choose **Usage** in the pop-up menu or in the menu on the lower toolbar. The usage option displays a list of the definitions that use the selected scheme.

Delete a Scheme - To delete a scheme, select the scheme and click **Delete**, or choose **Delete** from the pop-up menu or in the menu in the lower toolbar. The scheme is removed from the QKB.

Vocabulary Editor

The **Vocabulary Editor** enables you to build a vocabulary. When the parsing system needs to categorize a word, it can then easily search a single Vocabulary rather than multiple text files. It is recommended that you build one Vocabulary per parse definition. Before using the Customize Vocabulary Editor, you should define all of your basic categories and implement them in a Grammar, create your parse definitions, and create a text file for each basic category.

Each word in the Vocabulary is defined as belonging to one or more categories, which are defined in an associated Grammar. Each word is also assigned a likelihood, a score indicating a level of confidence (VERY HIGH, HIGH, MEDIUM, LOW, VERY LOW) that a word belongs to a certain category. For example, the next display shows the likelihood associated with each category for the word "Kim," a word in the EN Gender Analysis Vocabulary.



In the example above, the word "Kim" has two categories: FGW (Female Given Name Word) and MGW (Male Given Name Word). Both of these categories have a likelihood of MEDIUM. In some contexts, both the category and likelihood values are used to make a determination about a word, such as the gender associated with the word. In other contexts, only one of these values might be used to make a determination.

Within the Vocabulary Editor, you must specify which input text sources you want to combine to develop a Vocabulary, and indicate which category's data each library represents. Note that a Vocabulary is stored in a proprietary format. To help ensure Vocabulary integrity, you should not attempt to create or edit a Vocabulary directly, but rather through the **Vocabulary Editor**.

Building Vocabularies

Create a New Vocabulary File

You can use the **Vocabulary Editor** to build Vocabularies. It is recommended that one Vocabulary be created for each parse definition. The following steps assume that you have registered a QKB as described in Registering (Adding) a QKB on page 148. .

Perform these steps to add a new vocabulary file:

1. Select **Tools** > **Other QKB Editors** > **Vocabulary Editor** from the main window. The **Vocabulary Editor** is displayed, and you are prompted to specify a locale.
2. Specify a locale in the **Select Locale(s)** and click **OK**. Typically, this locale would be one with vocabularies that you want to view or maintain in the **Vocabulary Editor**.
3. In the **Vocabulary Editor**, select **File** > **New**. You are prompted to specify a locale where the new vocabulary file should be stored.
4. Specify a locale in the **Select Locale(s)** and click **OK**. A new, empty vocabulary file opens in the **Vocabulary Editor**.

Import Categories from a Grammar

You can import categories and likelihoods so that you can associate those values with the words that you add to your vocabulary. Each word must be associated with one or more categories from the Grammar.

1. To import categories into the **Vocabulary Editor**, choose **Options** > **Categories**.
2. Click **Import** to display the **Select Grammar** option.
3. Select the **Grammar** that you want to associate with your new Vocabulary, and then click **OK**.
4. On the **Categories** screen, the Grammar's standard category abbreviations and descriptions appear. (Derived categories are not imported.)
5. Use the **Delete** button to delete any unwanted categories, and then click **Close**.

Import Words into the Vocabulary

After you import categories, you can import the words to create a vocabulary that fits those categories.

Planning Your Import

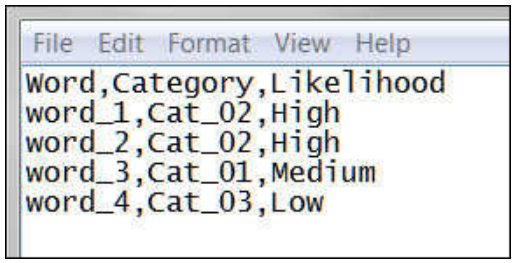
Before you import a vocabulary file, review its contents. Make sure that the file meets the criteria described in this section. You can import vocabularies in the following formats:

- Vocabulary
- Scheme
- Text files
- Delimited text file

For the delimited text file format, the file to be imported must have the following format:

```
<word><delimiter><category><delimiter><likelihood>
```

For example, the following is an example of a delimited text file, which uses commas as the delimiter and contains a header row.



The delimiter character is specified as part of the import action.

If the chosen delimiter character occurs as part of a word, category, or likelihood value, you must encapsulate the word, category, and likelihood values with a text qualifier in the import action. For example, if the delimiter is a space value and the word, category, and likelihood values contain space values, you might use a double-quotation mark to demarcate the values as follows:

“This is the word” “This is the category” “Very High”

If the text identifier occurs naturally within a word or category value, it must be escaped as well. For example:

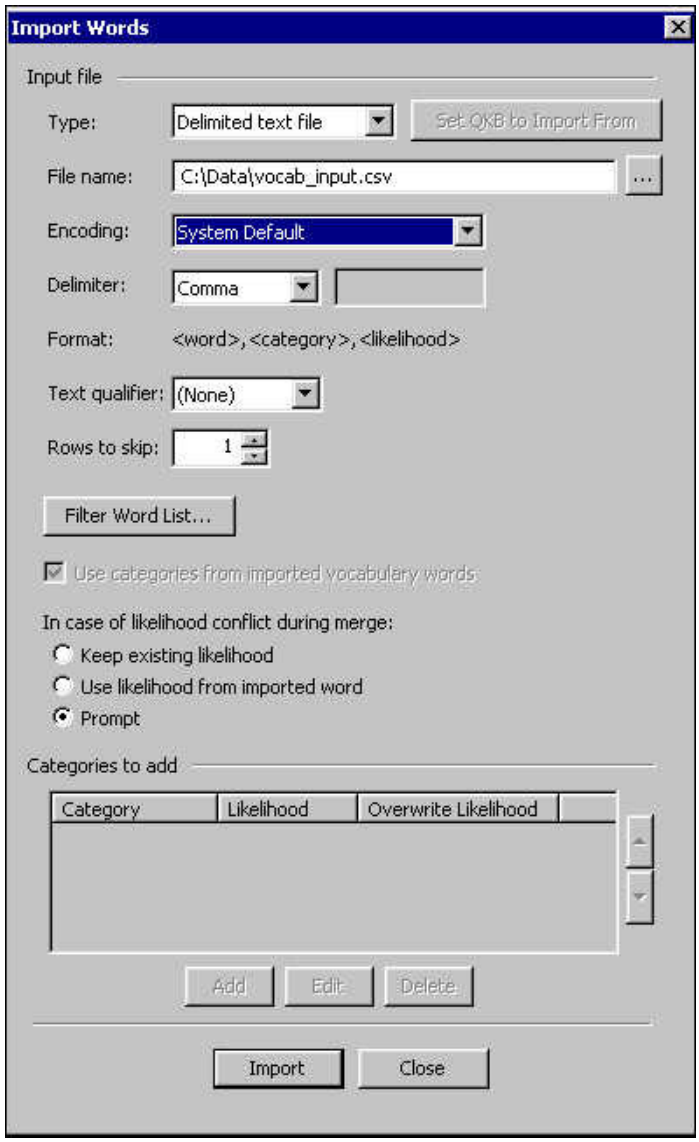
"This is an escaped ""quotation"" in a word" "Quoted Words category" "Very Low"

The valid likelihood values are: Very Low, Low, Medium, High, and Very High. All other values are considered an error and causes the import to fail.

All spaces before and after the delimiter are removed during the import.

Performing the Import

1 In the **Vocabulary Editor**, select to display **Import Words**. A window similar to the following displays:



The following describes the parameters for **Import Words**.

Note: Some parameters are enabled or disabled depending on choices made on other parameters.

- **Type** - Specify the input file type.
- **Set QKB to Import From** - Specify which Quality Knowledge Base to import the Vocabulary or Scheme from if it is not the current QKB.
- **File name** - Specify the name of the input file to import.

> **TIP** Select the proper file extension filter in the file to find the desired input file (that is, *.csv for comma separated files, *.tab for tab delimited files).

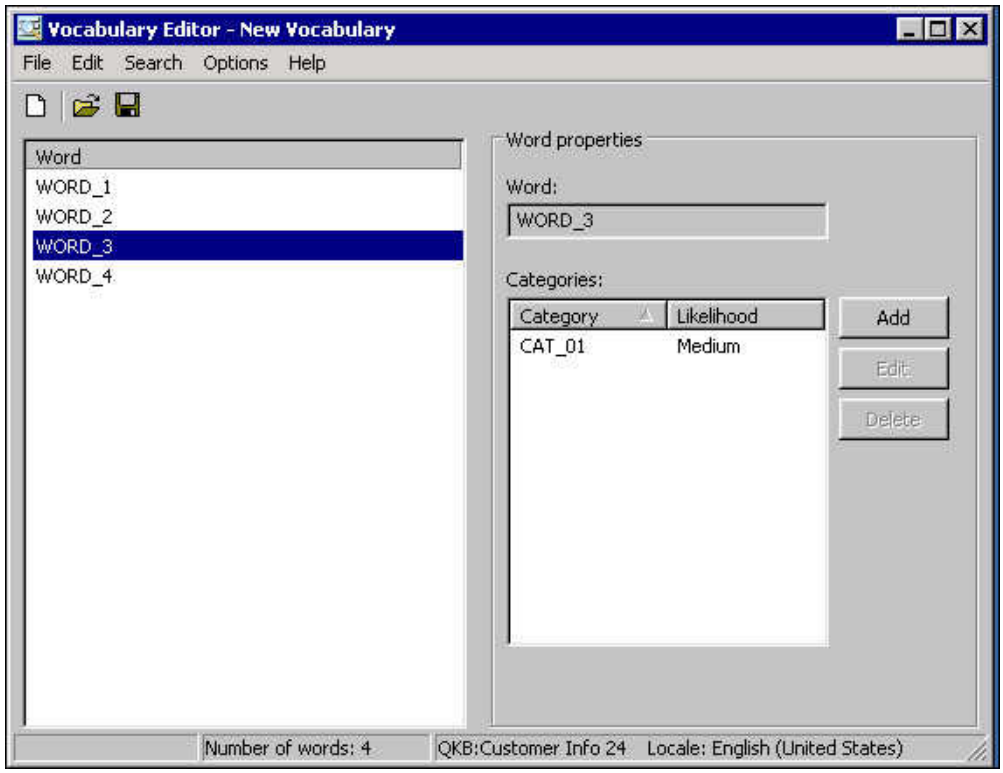
- **Encoding** - Specify the encoding of the input file.

- **Delimiter** - Specify the delimiter used in the delimited text file.
- **Text qualifier** - Specify the text qualifier used in the delimited text file.
- **Rows to skip** - Specify the number of rows to skip at the beginning of the delimited text file.
- **Filter Word List** - Specify a category filter so that only words that have a category, which matches the filter is imported.
- **Use categories from imported vocabulary words** - Specify if the categories for a given imported vocabulary word should be added when the word already exists in the vocabulary being imported into.
- **In case of likelihood conflict during merge** - If the imported word exists in the vocabulary where it is being imported into, if both words share categories, and the likelihood values differ, you must decide how to resolve the likelihood conflict. If this situation is encountered during the import, use this option to specify whether the existing likelihood should be preserved, the imported word should be used, or a prompt should be displayed to reconcile the conflict.
- **Categories to add** - Specify the categories and likelihoods to assign to every word imported. For each category that you select, you can select **Overwrite Likelihood**. Choosing this option adds your specified likelihood in place of a different value that can already exist for that word and category in the vocabulary being imported into.

2. Once all inputs have been provided, click **Import** to import words from the specified file. If you are importing categories, which do not already exist in the vocabulary being imported to, you are prompted to enter a description of each new category being imported. Enter a description for each new category and click **OK** to continue.

3. When the import is complete, a status message about the import displays. Click **OK** to dismiss the status message.

4. Click **Close** to close **Import Words**. The imported words display as shown below.



Review Categories and Likelihoods

Now that you are looking at the imported words and categories in your new Vocabulary, you can add or delete individual categories, or change likelihood values.

Here is an example of when you might want to update a likelihood value. If your vocabulary contains the name Scott, then that word might have the categories Family Name Word (FNW) and Given Name Word (GNW). You might determine that Scott is more likely to be a Given Name Word than a Family Name Word. You could then increase the likelihood value of the Given Name category for the word Scott.

To change a likelihood value, select the word and click **Edit**.

Although there can be some adjustments that you want to make to the likelihoods at this point, later testing with the **Parse Test Tool** reveals other necessary adjustments to give the desired result.

Save the Vocabulary

Now that your Vocabulary is built, you need to save it. Select **File** > **Save**. If this is a newly built Vocabulary, the **Vocabulary Editor** prompts you for a name.

Modifying Vocabularies

Other than altering the likelihood for specific words in a Vocabulary, it is recommended that you do not make many other modifications. However, certain situations can warrant it, so the **Vocabulary Editor** allows these operations. This can provide a good way to temporarily make changes for testing purposes.

Add a Word to a Vocabulary

1. On **Vocabulary Editor**, select **File** > **Open**.
2. Select the Vocabulary to which you want to add a word, and then click **Open**. The Vocabulary's details appear in the **Vocabulary Editor**.
3. Select **Edit** > **Add Word**.
4. Enter your new word, and then click **OK**. The word now appears selected under **Word** in the **Vocabulary Editor**.
5. On the right side of the screen, add at least one category with a likelihood value.

Note: The **Vocabulary Editor** alerts you if you attempt to add a word that already exists in the Vocabulary.

Modify a Word in a Vocabulary

1. In the **Vocabulary Editor**, select **File** > **Open**.
2. Select the Vocabulary that contains the word that you want to modify, and then click **Open**. The Vocabulary's details appear in the **Vocabulary Editor**.
3. Under **Word**, select the word or words that you want to modify. The word's categories appear on the right side.
4. Click **Add** to display **Add Word Category**. If you change the likelihood of a category that already belongs to the selected word or words, the **Overwrite Category** option appears. The **Overwrite Category** enables you to accept or refuse one or more changed likelihood values.
5. Select a category and click **Edit** to change category settings and likelihood values.
6. Select a category and click **Delete** to remove that category from the selected word or words.

Delete a Word from a Vocabulary

1 In the **Vocabulary Editor**, select **File** > **Open**.

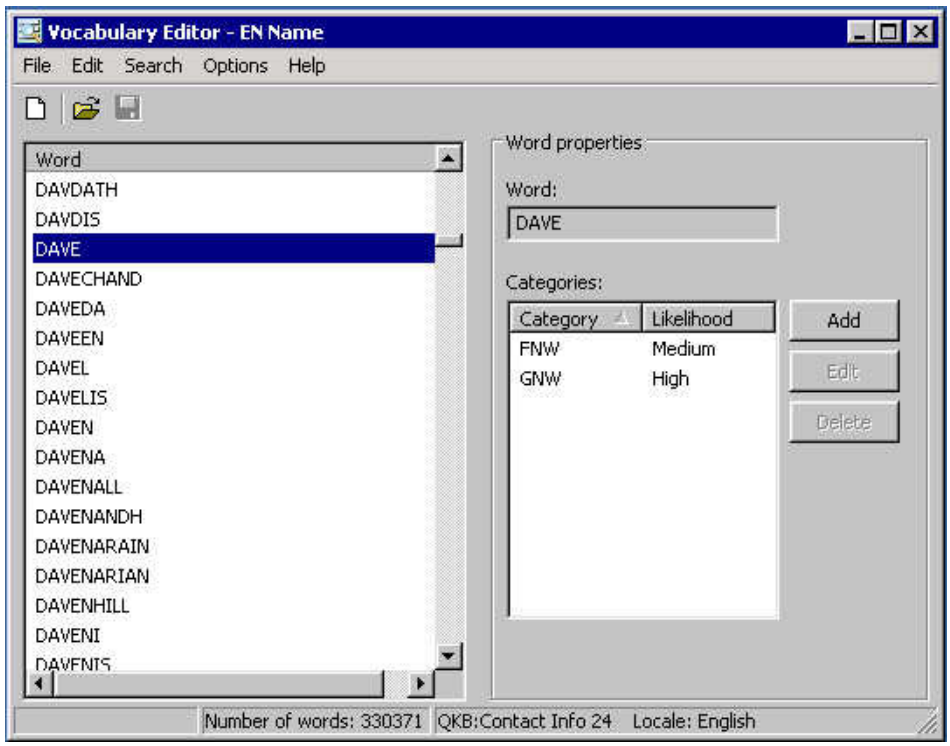
2 Select the Vocabulary that contains the word that you want to delete, and then click **Open**. The Vocabulary's details appear on the **Vocabulary Editor**.

3 Under **Word**, select the word that you want to delete.

4 Select **Edit** > **Delete Word**.

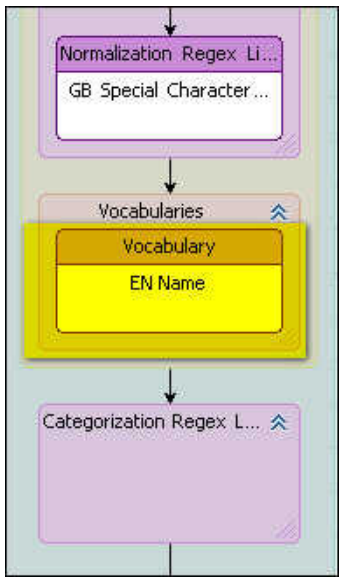
Navigating the Vocabulary Editor

A vocabulary is a type of QKB file that can be used to categorize words in text data. After a word is categorized by a vocabulary, it can be used in analysis and pattern-matching in subsequent nodes in a QKB definition. For example, the next display shows some of the words listed in the **EN Name** vocabulary in the English locale in the SAS Quality Knowledge Base.



In this example, the word Dave is assigned to two categories: Family Name Word (FNW) and Given Name Word (GNW). The FNW category has a likelihood score of Medium and the GNW category has a likelihood score of High. These scores reflect the fact that the word Dave is likely to be a name, and it is more likely to be a given name than a family name.

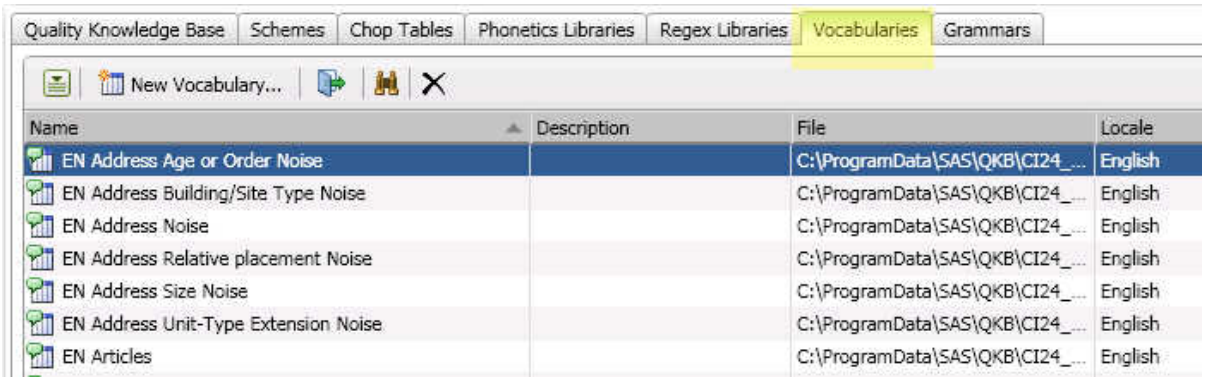
This information can be used in subsequent nodes in a QKB definition. The next display shows the **EN Name** vocabulary in the context of the **Name** parse definition in the **English** locale.



Vocabularies Tab

When you open a QKB in DataFlux Data Management Studio, the contents of the QKB are displayed in a set of tabs on the right-hand side of the screen. The **Vocabularies** tab shows a list of vocabularies that are contained in the QKB.

The list of vocabularies in the **Vocabularies** tab has several columns, as shown in the next display.



The Name column shows the name of a vocabulary. The Description column contains text about the vocabulary. The File column contains the physical path to the vocabulary file.

The Locale column shows the QKB locale with which the vocabulary is associated. Note that a vocabulary might be listed even if it is not associated with the locale that you have selected in the Quality Knowledge Bases tree. This is because locales can inherit vocabularies from ancestors.

If you select a locale and look at the values in the Locale column, you see that some vocabularies are associated with the selected locale. Others are associated with the locale's language, and still others are associated with the Global set of objects. This information is also presented in the icons that are shown for the vocabularies in the list. As shown in the previous display, if an icon has a flag overlay, the vocabulary is associated directly with a locale. If the icon has a word balloon overlay, it means that the vocabulary is associated with a language. An earth overlay means that the vocabulary belongs to the Global set of vocabularies.

If you want to view only the vocabularies that are associated with a language, select that language in the tree for your QKB in the **Administration** riser.

If you want to view only the vocabularies in the Global set, click on **Global** in the tree for your QKB.

Find a Vocabulary by Name

To find a vocabulary with a specific name, click the **Find** icon (binocular icon) in the lower toolbar. This toggles on the **Find** pane. To toggle off the **Find** pane, click the **Find** icon again.

Rename and Duplicate a Vocabulary

You can rename a vocabulary by right-clicking the name and selecting **Rename** from the pop-up menu. You can also click the name and select **Rename** from the menu on the lower toolbar.

You can make a copy of a vocabulary by right-clicking the name and selecting **Duplicate** from the pop-up menu. You can also click the name and select **Duplicate** from the menu on the lower toolbar.

Create a New Vocabulary

To create a new vocabulary, click the **New Vocabulary** button on the lower toolbar, or select **New Vocabulary** from the menu on the lower toolbar. After you have created the new vocabulary, the vocabulary is shown in the vocabulary list, and the Vocabulary Editor on page 126 appears.

Open a Vocabulary for Editing

To open a vocabulary for editing, double-click on the vocabulary in the list, or select the vocabulary and click **Open** on the lower toolbar. You might also select the vocabulary and choose **Open** in the pop-up menu or in the menu on the lower toolbar. See Vocabulary Editor on page 126 for information about how to edit a vocabulary.

View Definitions That Use a Given Vocabulary

To view a list of QKB definitions that use a given vocabulary, select the vocabulary and choose **Usage** in the pop-up menu or in the menu on the lower toolbar. A list of the definitions that use the selected vocabulary is displayed.

Delete a Vocabulary

To delete a vocabulary, select the vocabulary and click the delete icon in the lower toolbar, or choose **Delete** from the pop-up menu or in the menu in the lower toolbar. The vocabulary is removed from the QKB.

4

QKB Tools

QKB Merge Tool

The QKB Merge Tool is used to merge one Quality Knowledge Base (QKB) with another QKB. For example, if you customized your QKB, you might want to merge that customized QKB with a more recently released QKB.

In order to perform a merge, the following conditions must be met:

- the source and destination QKBs need to be accessible from the system where Data Management Studio is executing.
- the version of Data Management Studio used to perform the merge must be greater than or equal to the version used to customize the QKB prior to the merge.

IMPORTANT Use caution when using the QKB Merge Tool. Once you begin the merge process, you cannot go back. For this reason, it is recommended that you back up the destination QKB before using the QKB Merge Tool.

Access the QKB Merge Tool by selecting **Tools** > **QKB Merge Tool** from the **DataFlux Data Management Studio** menu.

Source QKB	Select the source QKB for the merge process. During the merge process, the source QKB merges into the destination QKB. The source QKB does not change during this action.

Destination QKB	This is the destination for the merged QKB. The contents of the source QKB are combined with the contents of the destination QKB. The destination QKB is overwritten with the result. **Note:** Once you click **Merge**, the destination QKB is modified.
Log File	This is an optional field. Specify a location for the log file. If no log file location is given, no log is created.
Merge	Click **Merge** to run the merge process. The source QKB combines with the destination QKB. If Merge is disabled, check that the source and destination QKBs are selected and are not the same QKB.
Exit	Click **Exit** to close the QKB Merge Tool when you are finished.

QKB Difference Viewer

Overview

The QKB Difference Viewer is a tool used to view the differences between SAS Quality Knowledge Base (QKB) files. Some of the possible uses are listed here.

- Determining the changes that have occurred to a QKB since installation
- Determining the differences between two different QKB files
- Determining the differences between two separate QKB files
- Viewing what changes have occurred before merging QKBs

You can access QKB Difference Viewer by selecting **Tools** > **QKB Difference Viewer** from the main menu or by selecting **Tools** > **QKB Difference Viewer** from the menu in Customize. You can also use a command-line interface to view the differences between QKB files. For more information, see QKB Difference Viewer Command Line Interface on page 146.

QKB Difference Viewer Screen

The following sections explain the QKB Difference Viewer screen including the main menu, toolbar, left navigation, and right navigation.

Main Menu

This section describes the QKB Difference Viewer main menu.

File

Open - Click **File** > **Open** to open a saved difference file. The difference file is saved with the .qkb extension.

Build Difference File - To build a difference file, click **File** > **Build Difference File**. The **Build Differences** opens. See Build Differences on page 143 for more information.

Recent - Displays recently created difference files. Click the file name to open.

Exit - Click to close the QKB Difference Viewer.

View

Status Bar - By default the **Status Bar** appears as a part of the QKB Difference Viewer screen. Click **Status Bar** to toggle the status bar.

Back - Click **Back** to go to the previous screen viewed.

Forward - Click **Forward** to return to the next screen.

Filters

You can use filters to hide items that you do not want to see. To set a filter, select the item that you want to filter under **Properties**. Then, click **Filters** > **Filter Name** or **Filters** > **Filter Name/Value Pair**.

Note: You can also use the toolbar options, **Filter on name** or **Filter on name and value**.

Filter Name - This option is used to filter all items with the chosen name. This filters on name only.

Filter Name/Value Pair - Filters all items that have the selected name and value. This filters on name and value.

Clear Filters - Select **Clear Filters** to clear all filters.

Filter List - Click **Filters** > **Filter list** to view the filters. The filters appear under **Filters**.

Note: You can remove filters only when the User Set column shows Yes. If **Remove Filter** is not available, the button is disabled.

To remove a filter, you can click on the item on **Filter** and click **Remove Filter** or if you want clear all filters, click **Clear All**. You can also clear all filters from the QKB Difference Viewer screen, click **Clear filters**.

To view filters, click **Show filters** and **Filters** opens.

Help

Help Topics - To access the QKB Difference Viewer online Help, click **Help** > **Help Topics**.

About - Click **Help** > **About qkbmt_viewer** to view version information about the QKB Difference Viewer.

Toolbar

The following toolbar buttons are available to quickly access certain functions of the QKB Difference Viewer.

Left Navigation

The left navigation window shows an expandable and collapsible list of the differences between two component QKBs. Click **+** to expand the view of the list, click **-** to collapse.

When you select items in the left navigation, the details in the right navigation change.

Right Navigation

The right navigation window shows
the changes between the current file and the new file. The sections within the right pane include **Properties**, **Data Records**, and **Dependent Definitions**.

Icons

The following icons might appear in the **Properties** or **Data Records** section of the right navigation pane.

Icon	Name	Description
	Information	Provides file details.
	Added	Indicates items added to the new file.
	Removed	Indicates an item has been removed.
	Current Values	The current value, before any changes.

Icon	Name	Description
	New Values	The new values in the new file.
	File Added	Indicates the file has been added.
	File Changed	Indicates a file has changed.
	Reference Changed	Indicates there is a change in the reference.

You can also put your cursor over the icon to see the name of each icon.

Properties

The **Properties** section shows differences in the properties. The icon on the left indicates the type of change.

Show Filtered - Click **Show Filtered** to view the filtered items in this section. If the **Show Filtered** button is disabled, there are no filters.

Data Records

The **Data Records** section displays the differences in the records. The icons on the left indicate the type of change.

Find Record - Click **Find Record** to locate a record within the **Data Records** list.

Dependent Definitions

This section displays items dependent on the item currently being viewed. The dependent items are generally definitions but can also be files. Click the link under Definition Name to go directly to the folder and file associated with the Locale and Match Definition.

The right navigation view changes to show details for the item selected in the left navigation view.

Build Differences

From the QKB Difference Viewer, click **File** > **Build Difference File**. The Build Differences opens.

The following sections explain the options available for Build Differences.

Comparison Mode

Separate QKBs - Select this option to compare two different QKBs. All of the files in the first QKB are compared with the files in the second QKB. During this process, if a file exists in one QKB but not the second, the file is flagged as added or deleted.

When this option is selected, two list boxes appear under Selection. Select two different QKBs to compare.

Installed QKB - If this option is selected, the current QKB is compared with the installer version. The "clean install" version of the QKB is saved as a .anc file, at the time of installation. Each file with the .qkb extension is compared with the corresponding .anc file to show the changes made since the QKB was installed. In the case of multiple installations, the difference shows only the changes since the most recent installation.

Note: If multiple QKB versions were installed into the same directory, a merge has occurred among the QKB files. The .anc files are identical to the files contained in the most recently applied installer package, and do not contain any information about the previous installations.

When this option is selected, a list box appears under Selection. Select the QKB that you want to compare with the installer version.

Single QKB Files - If you select this option, two files are compared during the difference process. These files can be from the same or different QKBs. These files do not have to have the same name or similar contents but they do have to be the same type. Examples include regexlib and scheme.

When this option is selected, a File Type drop-down list appears under Selection. Select one file from each list box.

Note: The same QKB file can be selected from each list box.

As you select a QKB in each list box, the **File** drop-down list appears. Here, you can select a specific file that you want to compare.

Selection

File Type

The **File Type** drop-down list appears when you select the Single QKB Files option under Comparison Mode. Click the drop-down list and select a file type.

Current QKB

Name - The **Name** column displays the names of all the QKBs available.

Directory - This is the directory path for each QKB displayed.

New QKB

Name - This column displays the names of all the QKBs available.

Directory - The **Directory** column displays the directory path for each QKB displayed in the list.

File

The **File** drop-down list appears when you select **Single QKB Files** > File Type under **Comparison Mode**. Click the drop-down list and select a file.

Output

Difference File - In each of the Comparison Modes, you must specify a location for the difference file. The difference file is saved with a .qkd extension. If you do not specify this extension, it is appended to the file name.

Log File - The log file is saved with an .log extension. If you do not specify the .log extension, it is appended to the file name.

Include Vocab differences in main file - When this option is selected, the vocabulary is included in the difference process. This check box is selected by default.

Note: If OK is disabled, all of the required fields for the selected Comparison Mode have not been populated.

Once you are finished building your QKB Differences, click **OK**. The **QKB Diff Tool** opens.

The process steps through the QKB Differences. Once the process is finished, the final line shows, Completed.

Note: This process can take a considerable amount of time. The amount of time is dependent on the size of the QKB and the number of locales installed.

Once the process is complete, close the QKB Difference Tool and the difference results appear in the QKB Difference Viewer.

QKB Difference Viewer Command Line Interface

You can use `qkbdiffbuilder.exe`, a command-line interface, to view the differences between QKB files. Perform the following steps to display usage notes for the interface.

1. Display a command prompt and change to the bin folder for DataFlux Data Management Studio. The default path is as follows: `C:\Program Files\SAS\Data Management Studio\[version_number]\bin`.
2. Type `qkbdiffbuilder.exe` and then press **Enter**.

5

Maintaining QKBs

Making a QKB Active

After a QKB has been registered, you can make it active. When you make a QKB active, DataFlux Data Management Studio uses that QKB whenever it performs data management operations on your data.

After you specify a QKB as being active, it remains active until you unregister the QKB (remove the QKB from the **Quality Knowledge Base** folder on the **Administration** panel), or you make another QKB active. You can have only one active QKB at a time.

Write permission is required for the DataFlux Data Management Studio resource files. For more information, see Read and Write Access to the etc Folder on page 149. Perform the following steps to make a QKB active:

1 Display the **Administration** riser.

2 Expand the **Quality Knowledge Bases** folder. A list of registered QKBs is displayed.

3 Click the QKB that you want to make active.

4 Do one of the following to display the **Make Active** window:

   a Click **Make Active** in the **Administration** riser toolbar on the left.

   b Right-click the selected QKB on the left and select **Make Active**.

   c Click **Make Active** in the properties panel toolbar on the right.

5 In **Make Active**, select the option that makes this QKB active for you only or for all DataFlux Data Management Studio users on this host.

6 Click **OK**.

Note: If one user makes a QKB active for all users, that QKB does not become active for the other user's applications until they restart DataFlux Data Management Studio.

Registering (Adding) a QKB

If a QKB has been registered, it appears in the **Quality Knowledge Base** folder on the **Administration** riser. You can view it as described in Accessing QKBs from the Administration Riser on page 3. .

If a QKB has not been registered, you must register it so that it is accessible from the **Administration** riser. You must have write access to DataFlux Data Management Studio resource files to register a QKB. For more information, see Read and Write Access to the etc Folder on page 149.

Perform the following steps:

1 Open the **Administration** riser and right-click the **Quality Knowledge Base** folder.

2 Select **Add Quality Knowledge Base**.

3 Enter a descriptive name for the QKB that you want to register, such as *Contact Info 24*.

4 Use the control in the next field to navigate to the installation folder for the QKB that you want to register. Some common installation locations are as follows.

For newer QKBs	`drive:\ProgramData\SAS\QKB\<qkb-name><qkb-version>_<installation-instance>`
For older QKBs	`drive:\Program Files\SAS\QKB\<qkb-name>\<qkb-version>`

5 The **Private reference** check box is deselected by default. Select this option if you want only the current user to access this QKB in DataFlux Data Management Studio.

6 Click **OK** to register the QKB. It appears in the **Quality Knowledge Base** folder on the **Administration** riser. You can view it as described in Accessing QKBs from the Administration Riser on page 3. See also Making a QKB Active on page 147.

Read and Write Access to the Etc Folder

All DataFlux Data Management Studio users must have write access to the **etc** folder and its subfolders in the installation directory. If you do not have write access to these folders, you might get an error such as "You do not have permission to perform this action."

The person who installs DataFlux Data Management Studio must ensure that all users of this application have write access to the resource files in the **etc** folder and its subfolders.

Editing, Removing, or Unregistering a QKB

To edit the registration record for a QKB, right-click the QKB in the **Quality Knowledge Base** folder and select **Edit QKB Properties**.

To remove the registration record for a QKB, right-click the QKB in the **Quality Knowledge Base** folder and select **Remove QKB**. This does not delete the QKB files from the file system. It deletes the registration record in DataFlux Data Management Studio.

Select to unregister the selected QKB from DataFlux Data Management Studio. The QKB files are not deleted from the file system.

Exporting and Importing QKB Definitions

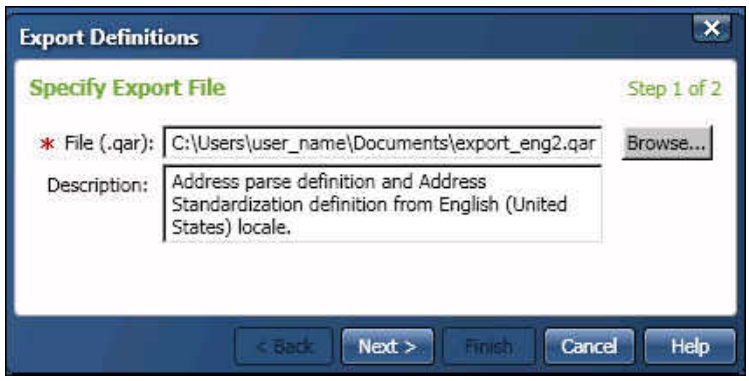
Export QKB Definitions

You can export QKB definitions from one QKB and import them into another. Identify the definitions that you want to export. Review Import and Export QKB Definitions on page 216. Then perform the following steps to export QKB definitions.

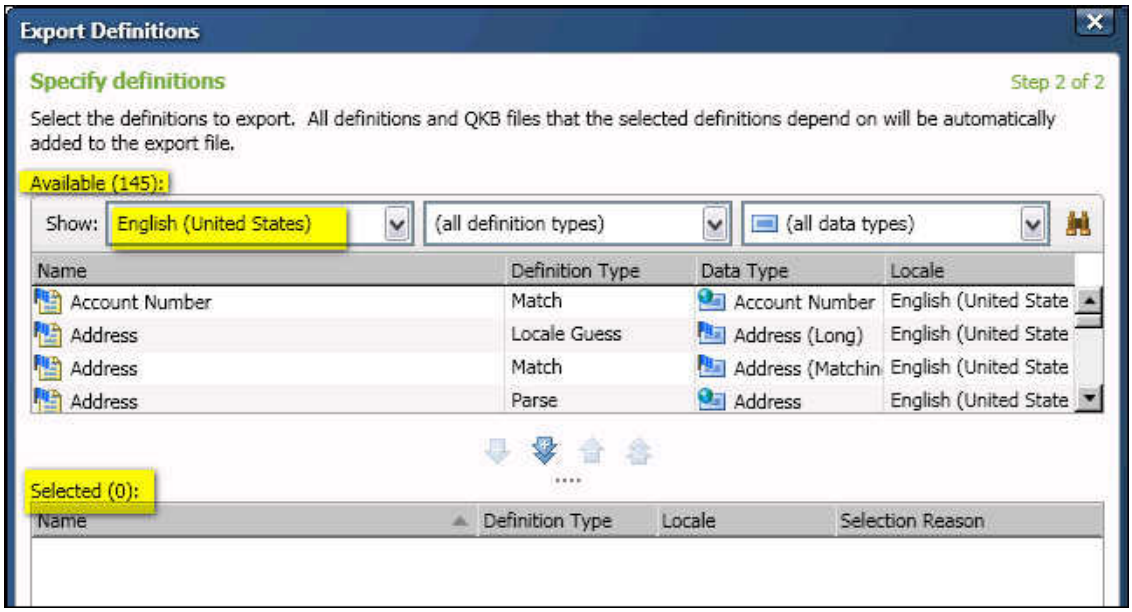
1 Open a QKB in the **Quality Knowledge Bases** folder in the **Administration** riser bar. For example, you could right-click a QKB named **Contact Info 24** and select **Open**. The contents of the QKB are displayed in the right panel.

2 Right-click a level that contains definitions that you want to export. The level could be a locale, a language, or the Global object. Then select **Export Definitions** from the pop-up menu. For example, you could right-click the **English (United States)** locale and select **Export Definitions**. The Specify Export File page is displayed.

3 Specify a full path and a description for the export file. The file must have a .qar extension. An example path might be: `C:\Users\user_name\Documents\export_eng2.qar`. The next display shows what these values might look like on the Specify Export File page.



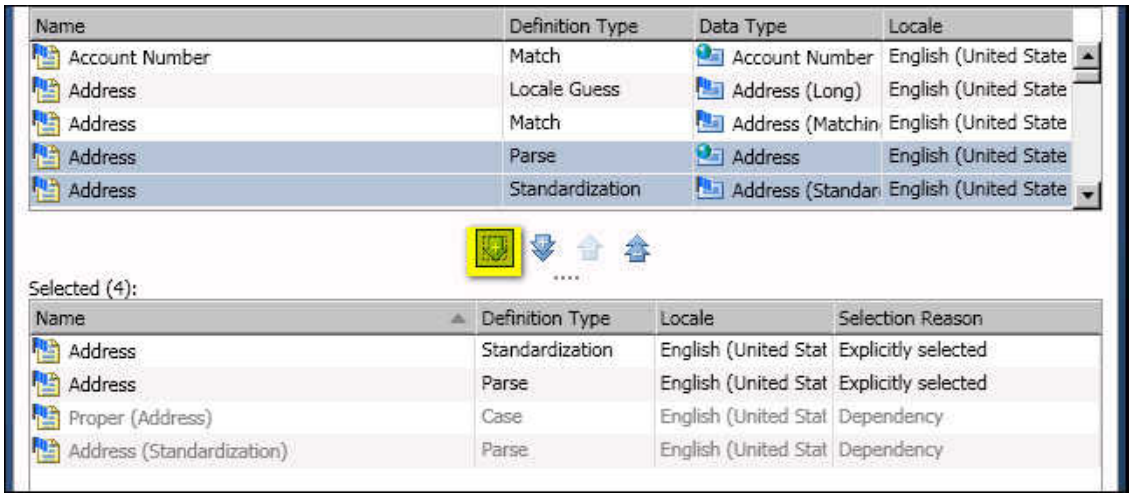
4 Click **Next**. A list of QKB definitions is displayed in the Specify definitions page. The next figure shows the definitions that are available for export from the **English (United States)** locale.



You can use the controls at the top of the page to select a different level (different locale, and so on) or restrict the list to certain definition types or certain data types.

5 Select the definitions that you want to export. Use Control-click to select multiple definitions. For example, you could select the **Address** parse definition and the **Address** standardization definition.

6 Click the single down-arrow to move the selected definitions into the **Selected** area at the bottom of the page. Alternatively, you could click the double down-arrow to select all of the available definitions. The selected definitions and any dependent definitions will be moved to the **Selected** area, as shown in the next display.



7 If you have selected all of the definitions that you want to export, go to the next step. If you want to export definitions from a different locale, use the **Show** control at the top of the window to select a different locale. Select definitions from that locale as described above.

8 When you have finished selecting definitions for export, click **Finish**. The export is performed. The Export status page is displayed.

9 Review the status of the export operation. If satisfied, click **Close**.

Import QKB Definitions

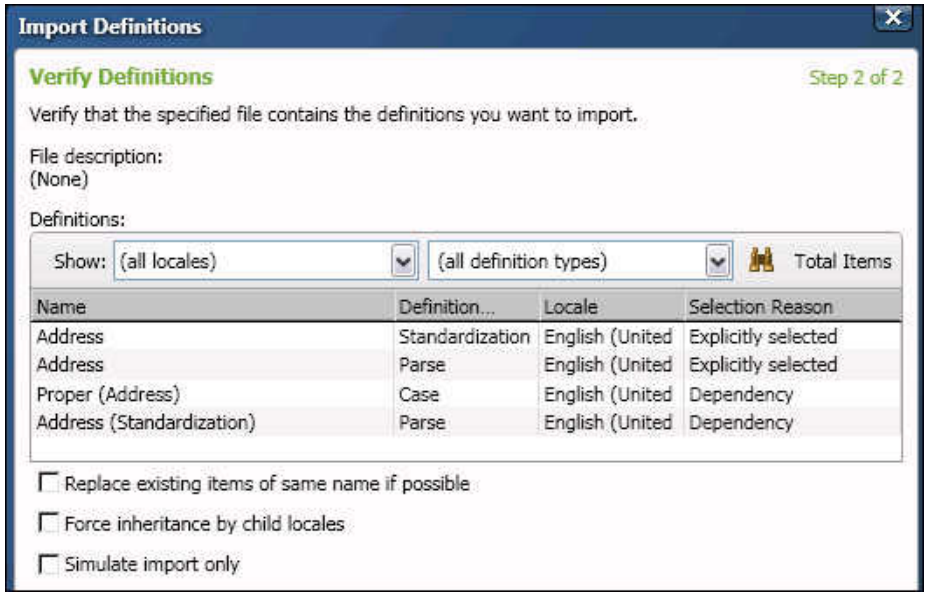
You can import QKB definitions that have been exported to a package file, as described above. Identify the file name of the package file that you want to import. Review Export and Import QKB Definition Usage Notes. Then perform the following steps to import QKB definitions.

1 Open a QKB in the **Quality Knowledge Bases** folder in the **Administration** riser bar. For example, you could right-click a QKB named **Contact Info 24** and select **Open**. The contents of the QKB are displayed in the right panel.

2 Right-click anywhere in the QKB tree and select **Import Definitions** from the pop-up menu. For example, you could right-click the **English (United States)** locale and select **Import Definitions**. The Specify Import File page is displayed.

3 Specify a full path for the package file that you want to import. The file must have a `.qar` extension. An example path might be: `C:\Users\user_name\Documents\export_eng2.qar`.

4 Click **Next**. The Verify Definitions page appears, as shown in the next display.



5 Use the controls at the top of the page to view all of the definitions that are in the import package. Use the check boxes at the bottom of the page to select one or more of the following options.

Replace existing items of same name if possible - Do not use in the current release.

Force inheritance by child locales - When selected, specifies that a definition imported into a locale is forced to be available from all child locales of the parent locale. This occurs even if the child locale already contains a definition of the same name. Some renaming can occur to allow this.

Simulate import only - When selected, specifies that the import is simulated but not actually run. No changes are made to the target QKB. Only the Import Log is written.

6 When you have finished reviewing the definitions that is imported, click **Finish**. The Import status page is displayed.

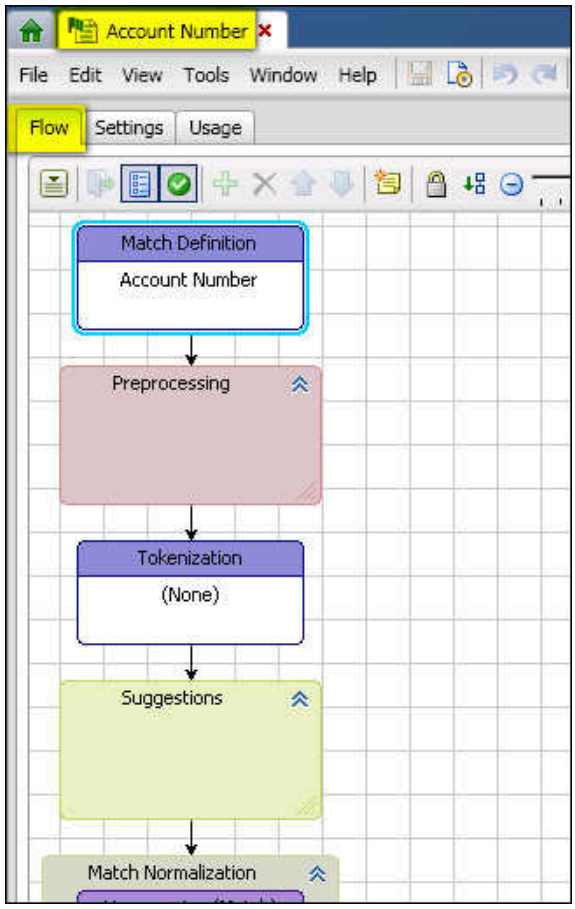
7 Review the status of the import operation. If satisfied, click **Close**.

Editing a QKB Definition

QKB Definition Editor

A QKB definition is a collection of metadata that defines an algorithm used to perform a data quality operation. The algorithm is represented as a flow diagram.

When you open a definition from the **Quality Knowledge Base** tab, the QKB Definition Editor appears. It displays the flow diagram for that definition. The next display shows part of the flow for the Account Number Match definition, which is in the English (United States) locale for the Contact Information 24 QKB.



The diagram contains nodes that represent processing steps that are performed by the definition. Some nodes are organized into groups. You can add or modify nodes and test the output of individual nodes until you are satisfied with the behavior of the definition. When finished, you can save changes to the definition. The updated definition is then available for use in DataFlux Data Management Studio and in other SAS data management software.

When you make changes to a definition, an asterisk is shown next to name of the definition in the tab title. When closing a tab that contains an edited definition, you must choose whether to save or cancel your changes. If you choose to save changes, any changes you have made on other tabs are also saved.

The following sections describe the tabs on the edit **QKB Definition** window: the **Flow** tab, the **Settings** tab, and the **Usage** tab.

Flow Tab

The **Flow** tab shows the flow diagram for the definition. This tab contains a toolbar that can be used to manage the flow diagram and perform editing operations on the definition's nodes. The toolbar contains the following selections.

Open - Open a definition or library that is referenced by a property in the selected node. This operation is enabled only when the selected node references some other definition or library.

Show/Hide Property View - Toggles on or off of **Properties** and shows the properties of the selected group or node.

Show/Hide Testing View - Toggles on or off of **Test** and shows test output for the selected group or node.

Insert - Insert a new node into a group. Enabled only when a selected group can accept new nodes. Some groups allow multiple nodes; some allow only a single node.

Delete - Delete a node. Enabled only when a node can be deleted.

Move Up - Move the selected node up relative to other nodes in its group. Enabled only when the selected node belongs to a group.

Move Down - Move the selected node down relative to other nodes in its group. Enabled only when the selected node belongs to a group.

Insert Note - Add a Note object to the flow diagram.

Lock Diagram - Lock the positioning of objects in the flow diagram to prevent accidental movement of objects.

Auto Layout - Re-draw the diagram with objects positioned in a default layout.

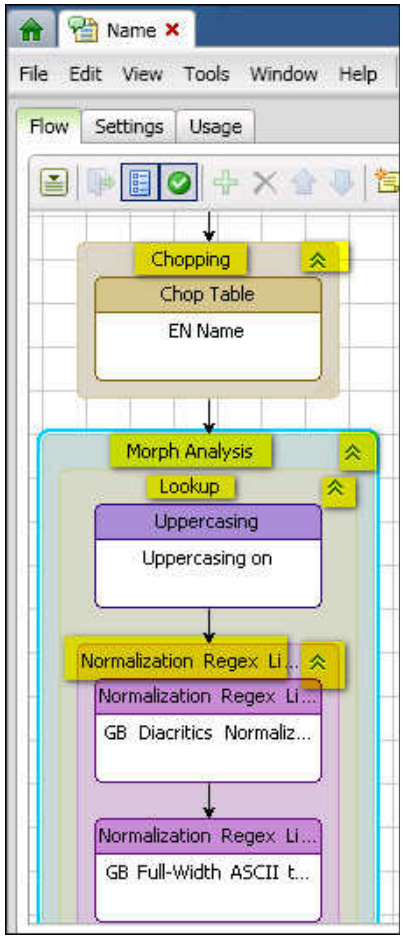
Zoom Out/In - Re-size the diagram for enhanced visibility.

Reset Zoom - Return to default Zoom setting.

Nodes and Groups

Each node in a definition represents a processing step performed by that definition. Nodes are arranged in a flow diagram, with the output of each node serving as input to the following node. Nodes have properties that you can edit to modify the behavior of the node.

Some nodes are organized into groups. A group is a logical collection of nodes that work together to perform a certain function. For example, the next display shows a number of groups in the **Name** parse definition, which is in the **English** locale for the **Contact Information 24** QKB.



The groups shown in the previous display are **Chopping**, **Morph Analysis**, **Lookup**, and **Normalization Regex Libraries**. The easiest way to identify a group is by the double up-arrow at upper right of the group's icon in the flow. You can use this control to expand or collapse a group.

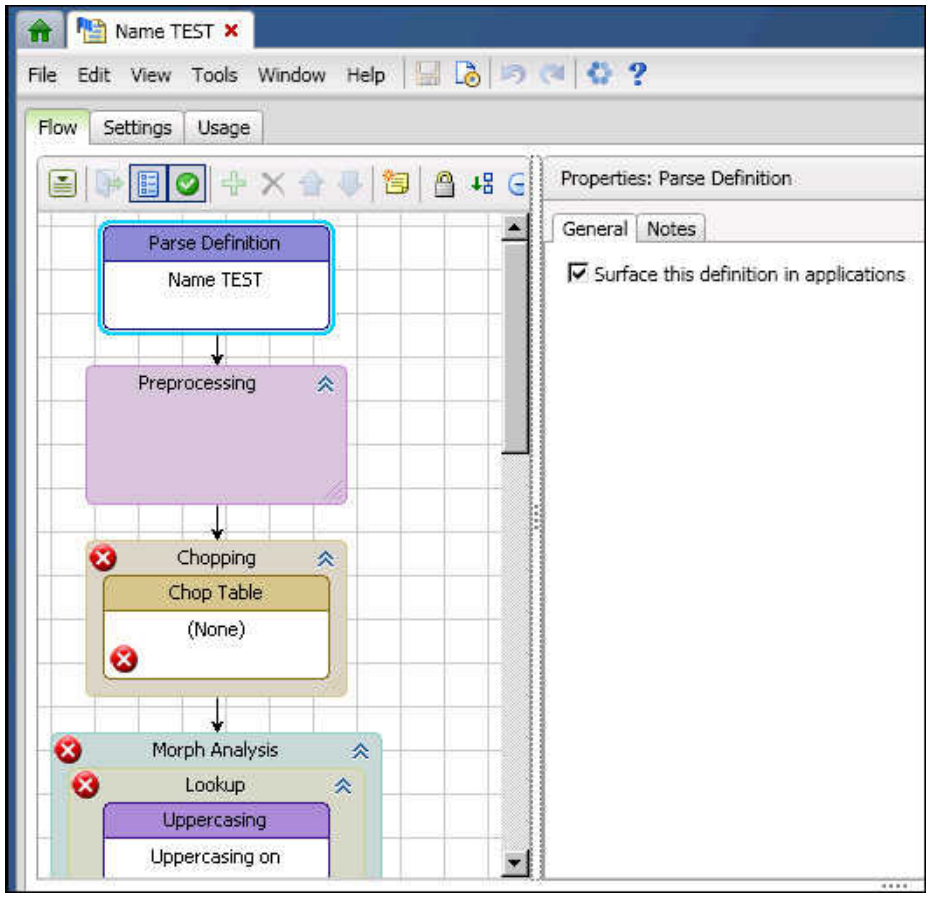
Some groups always contain exactly one node. Other groups can contain multiple nodes. If a group can contain multiple nodes, you can add or remove nodes from the group and rearrange the order of nodes within the group. For readability, you can re-size a group and drag it to a different location on the flow diagram. You can collapse or expand a group depending on whether you want to view the nodes in that group. You can also change the colors of groups and nodes.

A node's properties affect only the behavior of that individual node. However, some groups have properties that can be edited to modify the behavior of all nodes in that group. See more about testing and editing groups and nodes in the sections describing the **Properties** and **Test** views below.

To expand or collapse a group, right-click on the group and choose **Collapse Group** or **Expand Group** in the pop-up menu. Or select the group and choose **Collapse Group** or **Expand Group** in the menu on the **Flow** tab. To change the color of a group or a node, right-click on the group or node and select **Change Color** in the pop-up menu. Or select the group or node and choose **Change Color** in the menu.

Incomplete QKB Definitions

When a node in the flow diagram is displayed with a red circle with an X, this means that the node's properties are incomplete. If a group is displayed with a red circle with an X, this means that one or more of the group's nodes has incomplete properties, as shown in the next display.



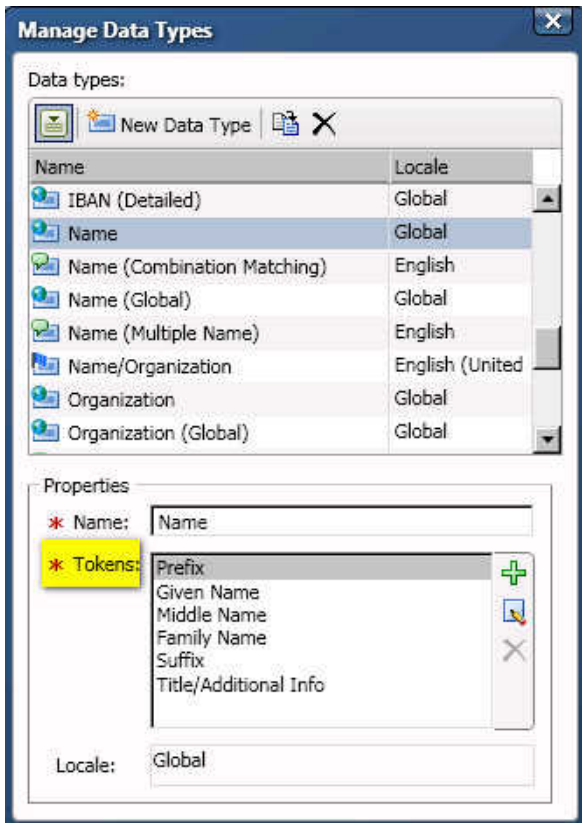
A definition does not function until the properties of all nodes are complete. Note that when a new definition is created, properties selections for many nodes might be incomplete. Hover over the affected nodes for a brief description of what is missing from each node. See also Viewing the QKB Error Log on page 162. .

Select the **Surface** this definition in applications property to allow the definition to be available to external applications. However, any QKB definition that contains an error is not available for use in an external application, regardless of the Surface property setting. For example, if a definition contains an error and the surface property is enabled, you cannot select that definition for use in a DataFlux Data Management Studio job or profile.

Tokens

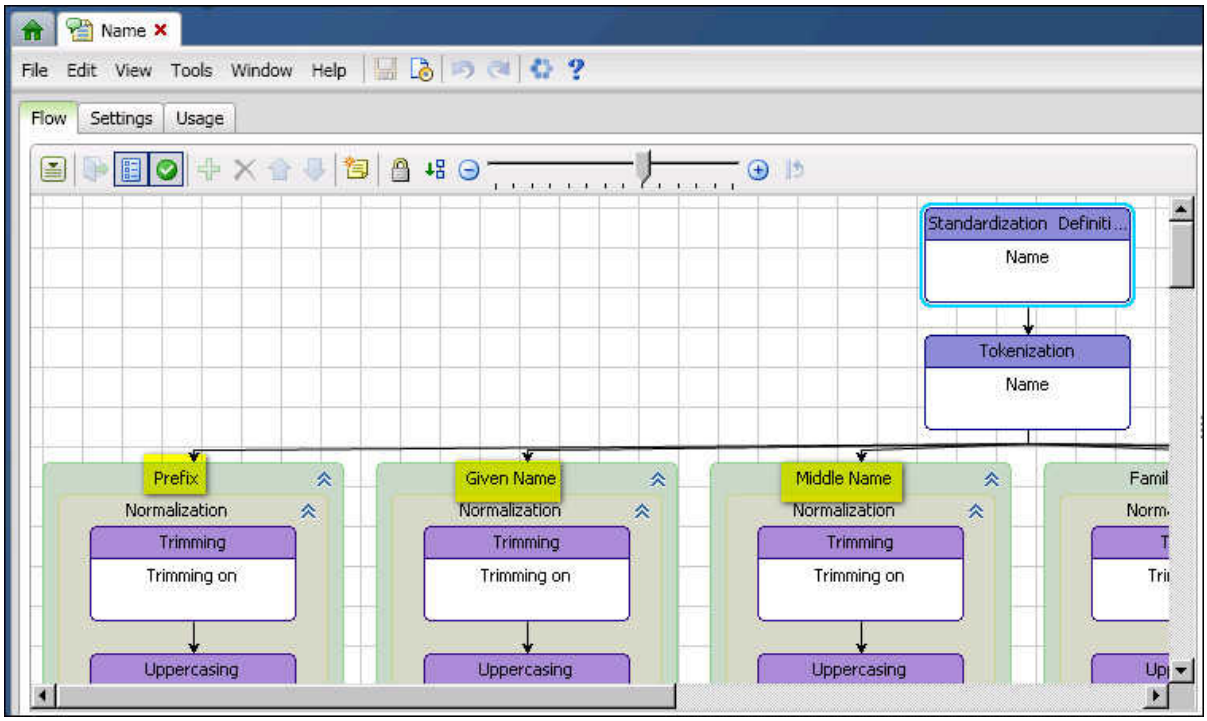
A QKB contains definitions developed for use with common types of data such as names, addresses, and phone numbers. Each data type contains one or more tokens. A token is a variable for the smallest meaningful part of a data value. For

example, the next display shows the tokens in the **Name** data type in the **Global** locale for the **Contact Information 24** QKB.



The tokens used by a QKB definition are defined by the definition's data type. For example, the tokens used by the **Name** standardization definition are defined by the **Name** data type.

In a definition, a token is represented as a group. Typically, each token group appears as a separate branch in the definition's flow diagram. A **Tokenization** node determines which portions of the input string are routed through each token's branch. The outputs of the different token branches are then combined to create an output for the definition. The next display shows some of the tokens in the flow for the **Name** standardization definition.

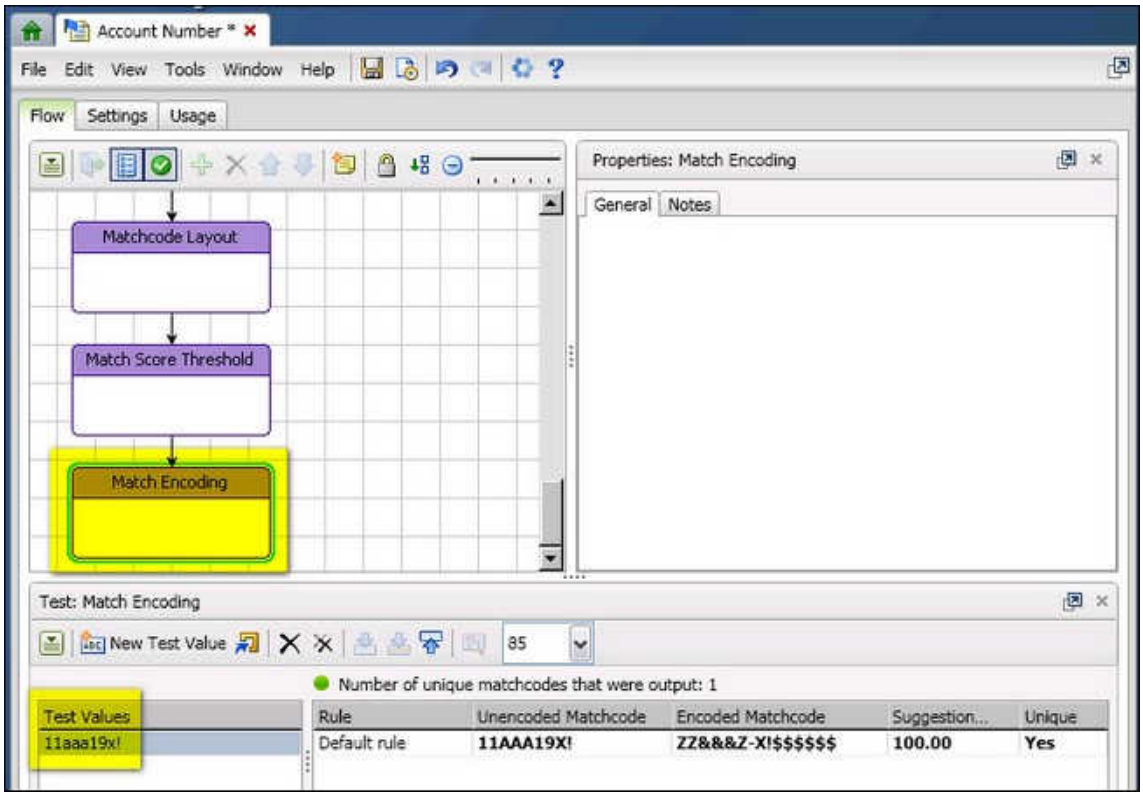


Properties View

When you select a node or group in the **Flow** tab, a **Properties** view appears on the right. You can toggle the **Properties** view on and off with the **Show/Hide Property View** on the **Flow** tab toolbar. You can edit the values of properties and then test the effects of your edits by viewing results for the selected node or group in the **Test** view. You can also add notes to a node or group in the **Properties** view. When notes are added, a note icon is displayed in the lower left corner of the node or in the upper left corner of the group.

Test View

The **Test** view appears at the bottom of the **Flow** tab. You can toggle it on and off with the **Show/Hide Testing View** option in the **Flow** tab toolbar. Use the **Test** view to verify node outputs for specific test values, as shown in the next display.



In the previous display, the **Match Encoding** node is selected in the **Flow** tab. A test value has been entered into the **Test Values** section on the left of the **Test** view. When the test value is selected, the output from the **Match Encoding** node is displayed on the right.

You can enter new test values manually or import new test values from a table or a text file. You can also modify existing values. To enter a new test value, click the **New Test Value** button on the toolbar or select **New Test Value** in the menu. A new line is added in the **Test Values** list. Enter a new test value at the prompt, and press **Enter**.

To import test values from a table or a text file, click the **Import Test Values** button on the toolbar or select **Import Test Values** in the menu. Then choose whether to import values from a table or a text file. To edit an existing test value, click on the value in the **Test Values** list, make edits, and press **Enter**.

The section at bottom right of the **Test** view shows the output of a node. After you have entered one or more test values, select a test value and then select a node in the flow diagram. The output for the selected node is displayed. Select a different test value to see the node's output for that test value. Remember that the output of a node serves as the input for the next node in the flow diagram. To view the output of the next node, you can click on that node or you can select **Step to Next Node** on the toolbar or in the menu.

To see the output for the previous node, click on the previous node or select **Step to Previous Node** on the toolbar or in the menu. Note that transformations made by nodes might be conditional, so for some nodes the output is the same as the input. If you want to step ahead to the next node that has a different output, select **Step to Next New Result** on the toolbar or in the menu.

Using QKB Libraries in Definitions

Some nodes in a definition contain references to QKB libraries such as Vocabularies, Grammars, and Schemes. When you select a library in a node, the contents of that library are used by the node when the node processes a string. You can open a library in the appropriate QKB library editor by clicking on the link in **Properties** for that node.

If changes are made to a library used by one of the nodes being edited, you are prompted to confirm if you want to reload the QKB. Reloading the QKB ensures that the in-memory copy of the QKB that you are editing is in sync with the copy that is stored on disk. The test results in the **Test** window are up to date. Click **Yes** to reload. If you choose not to reload, your QKB remains out of sync with the updated library until you close and reopen the QKB. Typically, if you are prompted to reload a QKB, you should do so.

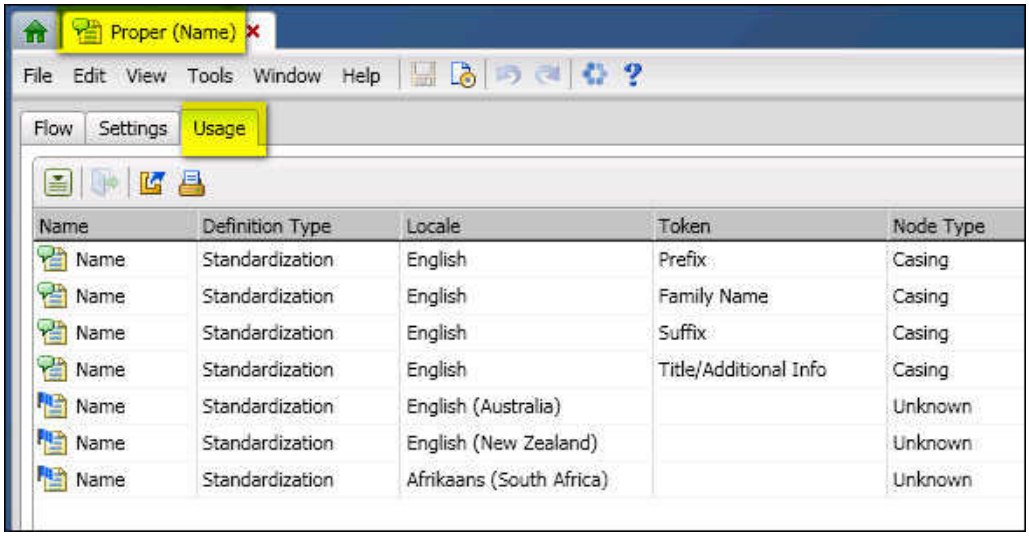
You can temporarily turn off these automatic prompts to reload the QKB by setting a global option. You might want to do this if you were making a lot of changes to a library. Select **Tools** > **Data Management Studio Options** > **QKB Definition Editor** to set these options. See the Warning Messages section of options.

Settings Tab

The **Settings** tab shows the properties settings for the definition. These settings were assigned when the definition was created. The name of the definition can be changed on the **Quality Knowledge Base** tab. The other settings cannot be changed. See Navigating the Quality Knowledge Base Tab on page 12 for general information about the **Quality Knowledge Base** tab in DataFlux Data Management Studio.

Usage Tab

The **Usage** tab shows other definitions that use the current definition. This enables you to see the definitions affected if you edit this definition. The list on this tab shows the name of the affected definition, the type of the definition (such as Match, Standardization, or Gender Analysis), and the locale with which the definition is associated. The next display shows the **Usage** tab for the **Proper (Name)** case definition in the **English** locale for the **Contact Information 24** QKB.



The list also shows the type of the node that refers to the definition that you are editing, so that you can easily locate the relevant node if you choose to edit the affected Definition. Finally, where applicable, the list shows the token branch on which the relevant node is located in the affected definition. You can open a definition in this list by double-clicking or by selecting the definition and choosing **Open** in the menu or by clicking the **Open** button on the toolbar. When you do this, the newly opened definition appears. You can also choose to print the list of affected files or export the list to a file.

Viewing the QKB Error Log

The QKB error log is populated when a given QKB locale is loaded. It is not dynamically updated. For example, if you load a locale and add a definition that errors, it is not displayed in the log until the QKB is closed and reopened.

A link to an error log appears at upper right of the main **QKB** window, as shown in the next display.

The word **Log** at upper right is a link to an error log. You can also select **View Error Log** from the top menu shown in the previous display. The log lists any errors with all definitions that have been loaded for the current locale. For example, if you open and view the **Global** locale only, the log lists errors with definitions within the **Global** locale only and not errors in other locales.

Any QKB definition that contains an error is not available for use in an application that consumes a QKB, regardless of the **Surface this definition in applications** property setting for the definition. For example, if a definition contains an error and the surface property is enabled, you cannot select that definition for use in a DataFlux Data Management Studio job or profile.

Inheritance in a QKB

In a QKB, a locale's ancestors are a language and the **Global** set of definitions and data types. For example, the **English (United States)** locale has ancestors **English** and **Global**, as shown in the next display.

The locale **English (United States)** inherits definitions and data types that are associated with the **English** language, as well as all **Global** definitions and data types. In other words, all **English** and **Global** definitions and data types are available in the **English, United States** locale. The reason for this architecture is that it is sometimes desirable to share a definition or data type across locales that have the same language, or even across locales that have different languages. The inheritance model allows this sharing.

Note that when you view the list of available definitions in a node in a data job, you see all of the definitions that are associated with the QKB locale that you have selected, plus the definitions associated with the locale's language, plus all **Global** definitions. This inheritance happens transparently. You cannot see the ancestor information for a definition in a data job. You can see this implementation detail only when you open and browse the QKB.

A locale can contain a definition or data type that has the same name as a definition or data type that is associated with an ancestor of the locale. In this case the definition or data type associated with the locale overrides on page 202 the definition or data type that is associated with the ancestor. For example, in the next display, an **English (United States)** definition has been added for **Account Number**. The local definition overrides the **Global** definition of the same name. The **Global** definition is automatically overridden and appears in a dimmed font.

Name	Definition T...	Data Type	Locale
Account Number	Match	Account Number	English (United States)
Account Number	Match	Account Number	Global
Address	Locale Guess	Address (Long)	English (United States)

Sometimes a locale can have another locale as an ancestor. For example, locale **French (Canada)** has locale **English (Canada)** as an ancestor, as shown in the next display.

The locale **French (Canada)** inherits all objects from locale **English (Canada)**. This allows an easy way to share definitions and data types between locales that have the same country.

6

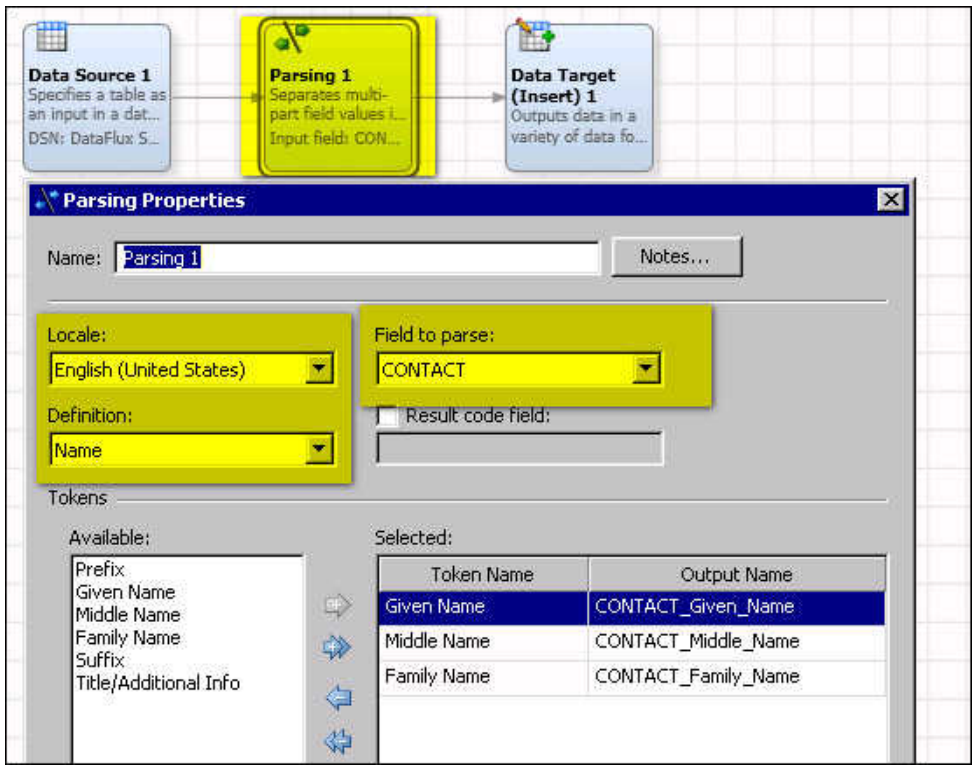
Using QKB Definitions

Using QKB Definitions in Jobs, Profiles, and Explorations

Using QKB Definitions in Data Jobs

Data jobs are the main way to process data in DataFlux Data Management Studio. Each data job specifies a set of data-processing operations that flow from a source to a target.

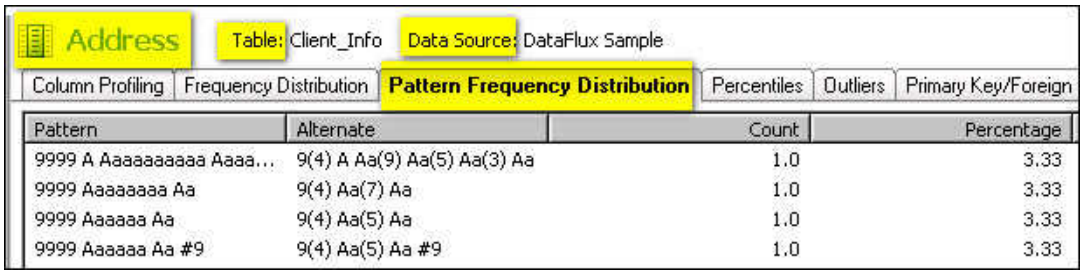
The **Quality** folder in the node tree for data jobs includes a number of nodes that apply a QKB definition to analyze or transform data. For example, the next display shows the sample data job df_sample_parse.



The Parsing node in the job specifies the English (United States) locale. The node uses the Name parse definition in that locale to parse names stored in a data-source field called CONTACT.

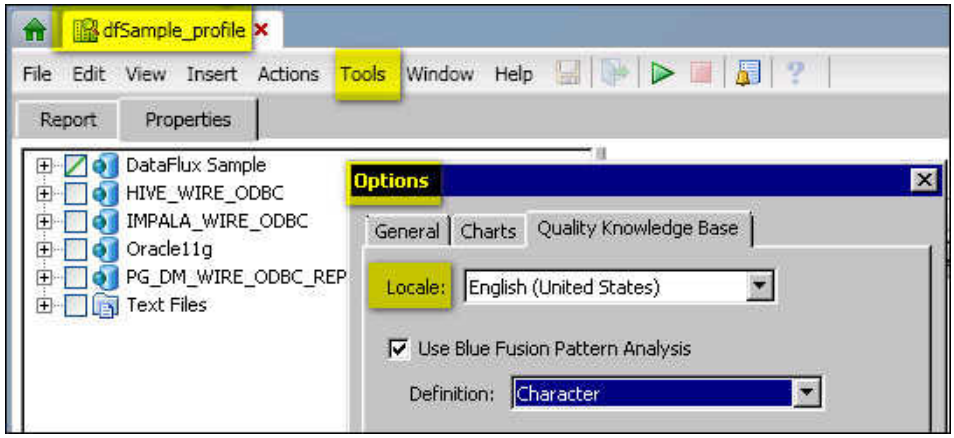
Using QKB Definitions in Data Profiles

A data profile is a job that analyzes a data source and produces reports. These reports reveal patterns, identify scarcity in the data, and calculate frequency and basic statistics. For example, the next display shows a report for the sample profile df_sample_profile.

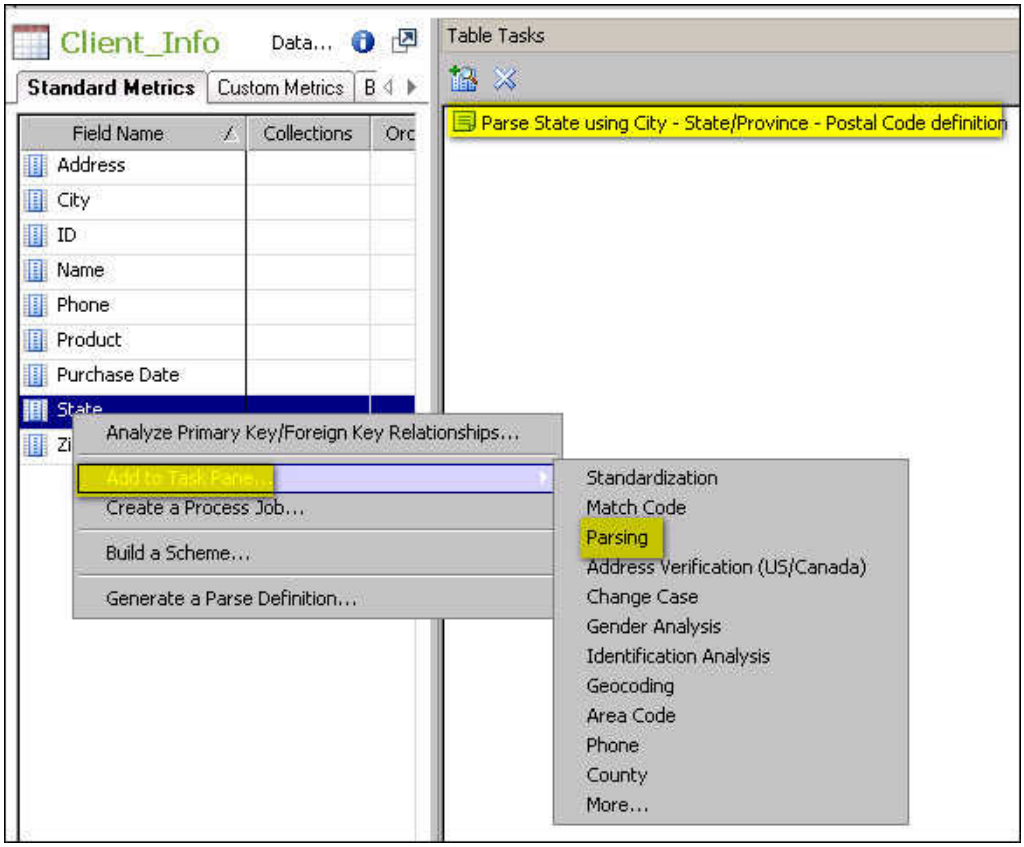


Pattern	Alternate	Count	Percentage
9999 A Aaaaaaaaaa Aaaa...	9(4) A Aa(9) Aa(5) Aa(3) Aa	1.0	3.33
9999 Aaaaaaaa Aa	9(4) Aa(7) Aa	1.0	3.33
9999 Aaaaaa Aa	9(4) Aa(5) Aa	1.0	3.33
9999 Aaaaaa Aa #9	9(4) Aa(5) Aa #9	1.0	3.33

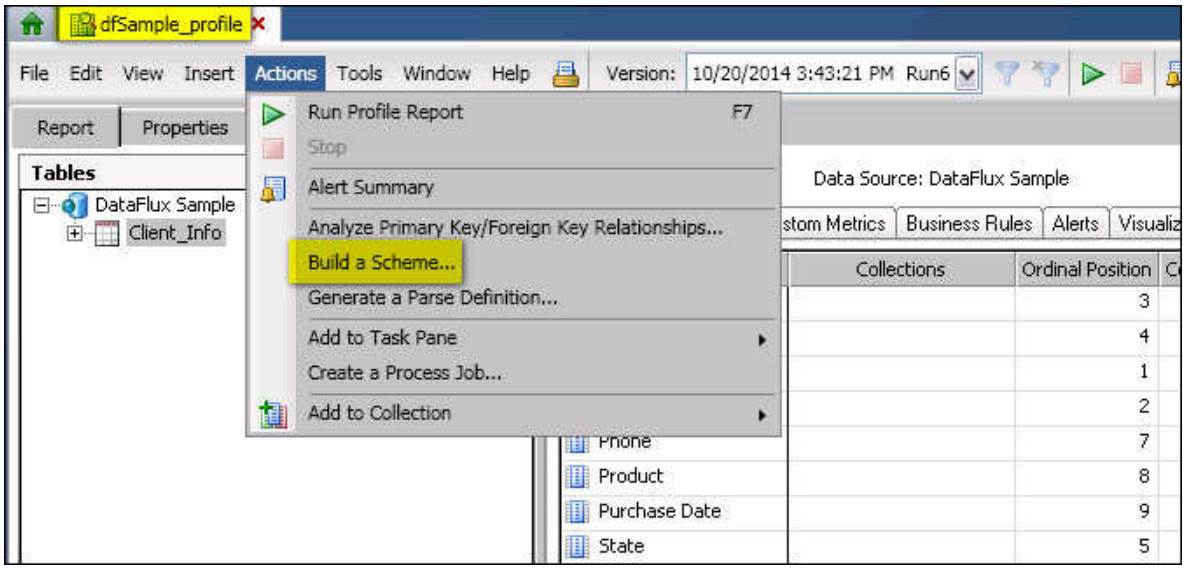
If you use the Pattern Frequency Distribution metric in a profile, you can use a QKB pattern analysis definition instead of the default pattern-generation feature for profiles. You select a definition on the **Quality Knowledge Base** tab in the profile **Options**.



You can also create a process job that contains a data job with Quality nodes by assigning tasks for individual data fields in a profile report.



Finally, you can generate a QKB scheme from the output of a profile report, as shown in the next display.

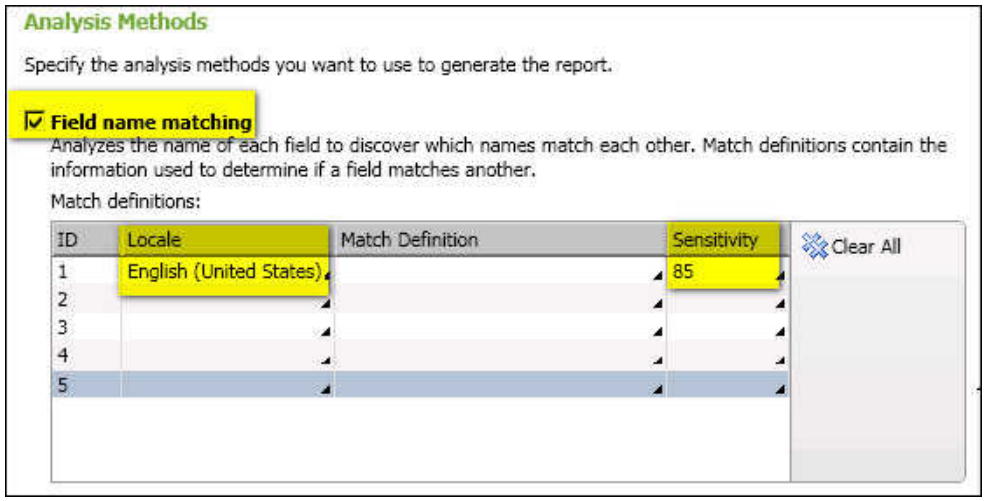


That scheme becomes part of your QKB and can then later be used to standardize data values.

Using QKB Definitions in Data Explorations

A data exploration reads data from databases and categorizes the fields in the selected tables into categories. These categories have been predefined in a QKB. Data explorations perform this categorization by matching column names. You also have the option of sampling the data in the table to determine whether the data is one of the specific types of categories in the QKB.

You can use match definitions and identification analysis definitions from your QKB to match and analyze database field names and field contents in a data exploration. This is done by selecting the QKB definition in the Analysis Methods section of properties for the exploration, as shown in the next display.



For more information about jobs, profiles, and explorations, see the DataFlux Data Management Studio: User's Guide.

Combination-Based Matching

This section explains how to use the combination-based matching feature for results from multiple matchcodes per input using token combination rules.

Overview

Match definitions have the ability receive multiple matchcodes per input (at a given sensitivity setting) by using token combination rules—this is known as combination-based matching. These rules allow the user to omit different combinations of tokens or change the order of tokens, so that multiple token

assignments can be evaluated, resulting in multiple matchcodes for a given input string.

One application of token combination rules is to address parsing ambiguity. For example, given a date string "2/4/2012", this can read as either the 2nd of April, 2012 or the 4th of February, 2012, depending on whether the D/M/Y or M/D/Y convention is used. When the date convention is not known with certainty, scenarios can arise where both interpretations are possible. A similar problem arises with personal names where the given and family names can be interchanged (for example, "ELTON JOHN", "JOHN ELTON").

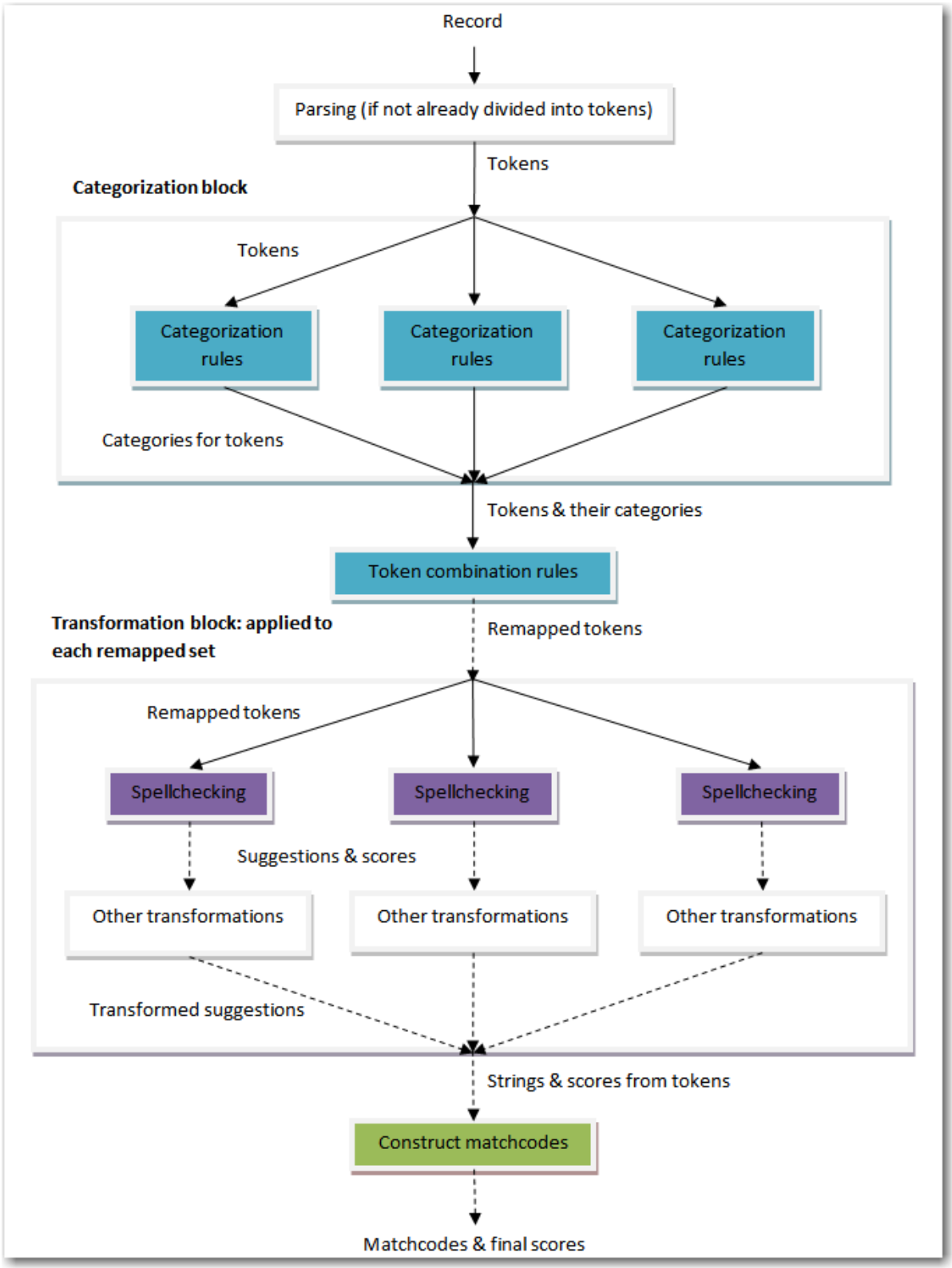
Another common application concerns entities where certain tokens can be omitted at will. Addresses written in the style of the United Kingdom are an example. According to postal regulations, the only mandatory elements of a United Kingdom address are Town and Postcode (though it is, naturally, possible to find records that incorrectly omit one or both of those tokens). The two mandatory elements are insufficient to fully identify an address, so some other information must be provided for a complete address. Depending on the specific situation, different combinations of tokens can be used to complete the address, and there is no way of knowing in advance which combination of these tokens are populated for any given record. Several simple examples are shown in the following table; all of these records could refer to the same physical address. Note that the last two records include mandatory tokens that have been omitted.

ID	Building	Number	Street	District/ Village	Town	County	Postcode
0	The Grange	32	Green Rd	Bishops Cleeve	Cheltenham	Gloucestershire	GL52 8XX
1	The Grange	32		Bishops Cleeve	Cheltenham		GL52 8XX
2			Green Rd		Cheltenham	Gloucestershire	GL52 8XX
3	The Grange		Green Rd		Cheltenham		GL52 8XX
4	The Grange				Cheltenham	Gloucestershire	
5		32	Green Rd				GL52 8XX

A token combination rule expresses a mapping between the tokens at the input of the match definition, and the tokens that are actually used in the matchcode generation processing steps within the match definition. A rule consists, broadly, of two main parts: the conditions and the actions. The conditions specify when the rule is applied, and are expressed in terms of a predicate involving one or more incoming tokens. The actions describe the output of the rule—specifically, what incoming tokens are mapped to the effective tokens that are used in the rest of the matchcode generation.

Each token combination rule can produce a matchcode, depending on whether its conditions are met for the particular record under consideration. In addition, the "default" matchcode, that is, the matchcode that would have been produced by the legacy definition with no token combination rules, be in the output. A single input to a combination-based match definition generally produces multiple output matchcodes. This means that a given record can appear in more than one potential cluster. In the case of combination-based matching, each matchcode corresponds to a different combination of tokens.

Suggestion-based matching is another, entirely separate, mechanism that allows a match definition to produce multiple output matchcode for a single input. A match definition can be configured for both combination-based matching and suggestion-based matching, if desired. The diagram below gives a conceptual overview of the match definition flow when both types of functionality are in use. Naturally, the required computation and storage resources increase when these features are used.


Record
Parsing (if not already divided into tokens)
Tokens
Categorization block
Tokens
Categorization rules
Categorization rules
Categorization rules
Categories for tokens
Tokens & their categories
Token combination rules
Transformation block: applied to each remapped set
Remapped tokens
Remapped tokens
Spellchecking
Spellchecking
Spellchecking
Suggestions & scores
Other transformations
Other transformations
Other transformations
Transformed suggestions
Strings & scores from tokens
Construct matchcodes
Matchcodes & final scores

When to Use Combination-Based Matching

The following is a list of some situations where switching from legacy matching to combination-based matching could be useful.

- An increase in computation time, storage and/or complexity in the de-duplication workflow is acceptable, in exchange for increased flexibility or accuracy. For example, when compared to a Matchcodes node that uses the legacy Address (Full) definition, runs several times slower and produces many more matchcode output rows, but also greatly reduces the error rate in the following clustering step.

  Note: In this example, legacy Address (Full) definition refers to the Matchcodes node that uses the Address (Full) (with Combinations) match definition shipped in CI 2011A in the English (United Kingdom) locale.

- The recommended best practice of using low sensitivities for a single match definition and then reducing false matches by combining matchcodes over multiple data types (for example, Name and Address) is not practical or does not sufficiently address the ambiguity present. This is commonly seen when working with records that contain a single piece of information. For example, an entity name, or when the other data types simply do not contain information reliable or relevant enough to the application at hand.
- The records that do not match (but are desired to match) differ from each other at the token level. This occurs, for example, when the differences between the records are due to a transposition of tokens (as in the M/D/Y versus D/M/Y date example), or to tokens that are missing in one record and present in the other. Note that in situations where the difficulty arises at the sub-token or character level, use suggestion-based matching.
- Behavior that is desired actually equivalent to allowing a record to appear in more than one cluster. For example, the desired behavior might be that "ELTON JOHN" should match "ELTON A JOHN" or "JOHN ELTON", but "JOHN ELTON" should not match "ELTON A JOHN". This is not possible using standard legacy matching, because each record must appear in exactly one cluster.
- A numeric score is needed to aid decision-making when resolving clusters or selecting survivors. Each combination-based matchcode includes a score, and after clustering, these scores are aggregated for the cluster. These scores can be used to aid manual review of clusters. The presence of scores also paves the way for business rules that automatically resolve clusters and select surviving records.

Working with Combination-Based Matching

In Customize, two new groups of nodes have been added to the match definition to support combination-based matching functionality: the Morph Analysis group and the Token Combination Rule (TCR) group. The Match Score Threshold node, added to support suggestion-based matching, is also useful in combination-based matching. A match definition is considered "combination-based" if the TCR placeholder group contains a TCR node. A new match definition has an empty TCR group. It is therefore still possible to create legacy match definitions, simply by not inserting any TCR nodes.

In this document, a simple definition, designed to tackle the understood family name/given name transposition problem, is used for illustration. In the following examples, the existing Name match definition is used as a starting point to which the trappings of combination-based matching are then added. Advanced users can wish to examine the Address (Full) (with Combinations) match definition in the English (United Kingdom) locale, which illustrates the handling of a far more complex problem.

Configuring the Morph Analysis Group

Each token has a Morph Analysis group, which is precisely analogous to the corresponding named group in a Parse definition. The purpose of this group is to assign categories to the incoming tokens; a token can be assigned zero or more categories, together with likelihoods for each of those categories. It is necessary to configure Morph Analysis groups only for those tokens that are used in the conditions of TCR nodes.

Note: More information about the Morph Analysis group and the nodes contained therein, including their properties sheets and testing outputs, can be found in the Customize documentation for the Parse definition.

To set up Morph Analysis, consider what conditions are necessary for the ambiguity to occur: the transposition of names must be plausible (as in the example of "JOHN ELTON" and "ELTON JOHN"). In principle, there is no limitation on what words can be used as either given or family names. The probability that a particular name is transposed does increase when the words concerned are commonly used as both given and family names. For simplicity, the examples shown here assumes that there is a fixed, known group of words that can be used as both given and family names. This allows the problem to be addressed using a single TCR node.

Note: Here, it is assumed that all the words in the vocabulary are equally likely to be present as family or given names. However, some words are more likely to be used as family names than as given names, or vice versa. Multiple TCR nodes could

be used to handle the different likelihoods; less-likely transpositions could be given a lower rule weight.

The only configuration now necessary in the Morph Analysis group is to insert a Vocabulary node into the group for both Given Name and Family Name tokens.

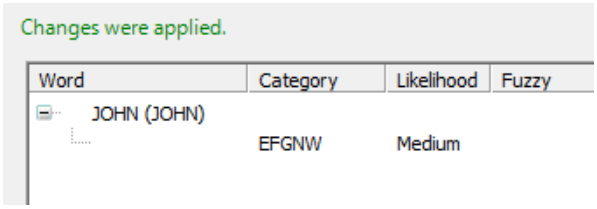
Additional complexity can be added to handle more realistic situations. For example, the input "ELTON A JOHN" strongly suggests that "ELTON" is the given name and "JOHN" the family name, because of the existence of the middle initial. In this case, transposition of family and given names is probably not desirable. To implement this, an additional category can be added to the vocabulary to describe a token that is not empty. A Categorization Regex node can then tag the Middle Name token whenever it is non-empty.

Testing the Morph Analysis Group

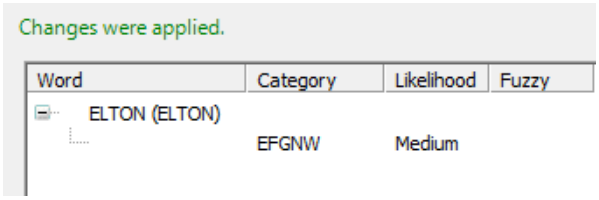
With the Vocabulary node configured as described above, a category is assigned to the Given Name token if one or more of its words is present in the vocabulary. Otherwise, no category is assigned . The same sort of result is produced for the Family Name token.

The next two figures show the test output of the Morph Analysis group when the input string contains only words that are present in the vocabulary. In this example, the input string is "JOHN ELTON". The desired category has been arbitrarily named EFGNW (standing for Either Family or Given Name Word).

This example shows the test output for Given Name token's Morph Analysis group with input string "JOHN ELTON". Both words were present in vocabulary.



This example shows the test output for Family Name token's Morph Analysis group with input string "JOHN ELTON". Both words were present in vocabulary.



Configuring the Token Combination Rule Node

The Token Combination Rule (TCR) node specifies the conditions under which the rule should take effect, as well as the desired effect of the rule. Each node describes a single rule and is responsible for producing one matchcode (if the rule's conditions are satisfied).

The properties sheet for a TCR node is shown in the following figure. Here, the rule is configured to use the categories previously assigned by the Morph Analysis groups. There are three conditions; these specify that the rule is applied when the Family and Given Name tokens both have the category EFGNW, and the Middle Name token does not have the category NEMP.

When all these conditions are satisfied, the Family and Given Name tokens is transposed. The rule is evaluated at all sensitivities (50 - 100) and the output of the rule is assigned a weight of 80.

Properties

Enable this rule - Check this box to enable the rule. By default, this box is checked; it is provided as a convenience to users who can want to selectively enable and disable certain rules while developing or debugging a new match definition.

Conditions - Each row of this grid specifies a single condition. All conditions must be satisfied to apply the rule. The Token column contains the name of the incoming token that should be examined. The Category and Likelihood columns contain, respectively, the name of the category that the incoming token must be assigned, and the minimum likelihood that must be associated with that assignment. The value in the Negate column should be set to "Yes" if the condition must be negated —this is sometimes useful when it is easier to express the opposite of a desired condition using categories. By default, the grid is empty, meaning that no conditions need be satisfied for the rule to be applied.

Actions - An action is the mapping of an incoming token to the effective token that is used in the matchcode generation processing steps. The available incoming tokens are listed in the Nominal Token column. Each incoming token can be replaced by another token, as specified in the Replacement column. By default, no Replacement tokens are assigned when the node is first created; leaving a token with no replacement, which causes that token to be replaced by a blank string (omitted).

Sensitivity - As with other nodes in the match definition, the sensitivity slider selects the range of sensitivities within which the node takes effect. This defaults to the entire range of sensitivities.

Weight - This specifies a weight that is associated with the rule, and defaults to 100. The weight is used to generate a score for each matchcode arising from this rule.

Testing the Token Combination Rule Node

The output of the TCR node indicates whether the conditions have been satisfied for the given input; if so, the rule is applied. The possible output includes, "**This rule is applied**" (if the conditions are satisfied) or "**This rule will not be applied**" (if the conditions not satisfied). The next two figures show the possible test outputs for the TCR node.

Configuring the Token Combination Rule Group

The properties sheet for the TCR group is shown below. These properties concern the handling of the "default rule"—the identity mapping of all incoming tokens to themselves, without condition. The default rule is defined implicitly and does not take up a TCR node; unless otherwise specified in these properties, this default rule is always evaluated with a weight of 100.

Users can wish to disable the default rule when the "default" case—the case where all defined tokens are present in their natural state—is not one that can reasonably occur. This can happen, for example, when all possible combinations are already covered by the user-defined TCR nodes.

Note: For an advanced use case, see the Address (Full) (with Combinations) match definition in English (United Kingdom).

Properties

Never evaluate default rule if there are any token combination rules defined - If this box is checked, inserting any TCR nodes into the group causes the default rule to be disabled.

Do not evaluate default rule if any token combination rules are applied to the input - If this box is checked, the default rule is not evaluated if at least one TCR node conditions satisfied.

Weight of default rule - This specifies the weight that should be assigned to the default rule.

Testing the Token Combination Rule Group

The output for the Token Combination Rule (TCR) group indicates whether any of the rules in the group are applied. It also informs the user if the default combination (the combination where all tokens are mapped to themselves) are evaluated. The possible test outputs for the TCR group include:

Output	Description
One or more rules are applied. The default token combination is evaluated .	Output of the TCR group in Customize if at least one rule condition is satisfied and the default rule is enabled
No rules are applied . The default token combination is evaluated .	Output of the TCR group in Customize if no rule conditions are satisfied and the default rule is enabled
No rules are applied. The default token combination is not evaluated.	Output of the TCR group in Customize if no rules conditions are satisfied and the default rule is disabled
One or more rules are applied. The default token combination is not evaluated.	Output of the TCR group if at least one rule condition is satisfied and the default rule is disabled

Configuring the Match Score Threshold Node

The Match Score Threshold node has only one parameter, the Minimum Score Threshold. This threshold value is applied to the combined score for the entire matchcode as laid out in the Matchcode Layout node. Any matchcodes that have scores below this value are discarded.

The threshold value should be set to limit the output of the match definition to some reasonable number of matchcodes for the typical input string. Since very low scoring matchcodes are likely to be ignored during entity resolution anyway, discarding them at this stage saves resources when the match definition is run in a data job.

Testing the Match Score Threshold Node

The test window output of the Match Score Threshold node is very similar to that of the Matchcode Layout node. The matchcodes are shown prior to obfuscation, together with their scores and the token combination rule from which each arose. If there were duplicate matchcodes generated (in other words the same matchcode arising from several token combination rules), this is also indicated. Only the matchcodes, which passed the threshold are shown.

Assuming that suggestion-based matching is not used, the score for each matchcode is, by default, the weight of the rule that produced it. If suggestion-based matching is in use, then this score also includes a contribution from the score produced at the Suggestions node. (Note that the final score is the suggestion's score multiplied by the rule weight, and divided by 100.) In addition, if the match definition option Weight scores by sensitivity is enabled, then the rule weights are

further modified by the current sensitivity setting. (Note that the score due to suggestion and/or combinations is multiplied by the current sensitivity setting and divided by 100.)

Using Combination-Based Match Definitions in a Data Job

See the DataFlux Data Management Studio: User's Guide for information about combination-based match definitions in a data job.

Suggestion-Based Matching

This section explains how to use the suggestion-based matching feature to detect data entry errors. The feature is implemented in match definition nodes, which you configure using Customize.

Overview

With the DataFlux Data Management Studio suggestion-based matching feature, match definitions have the potential to provide multiple matchcodes based on alternative "suggestions" for words. You can specify possible alternatives for words within a token. This addresses such problems as incorrect spelling of a word, typographical errors, or otherwise poor data entry.

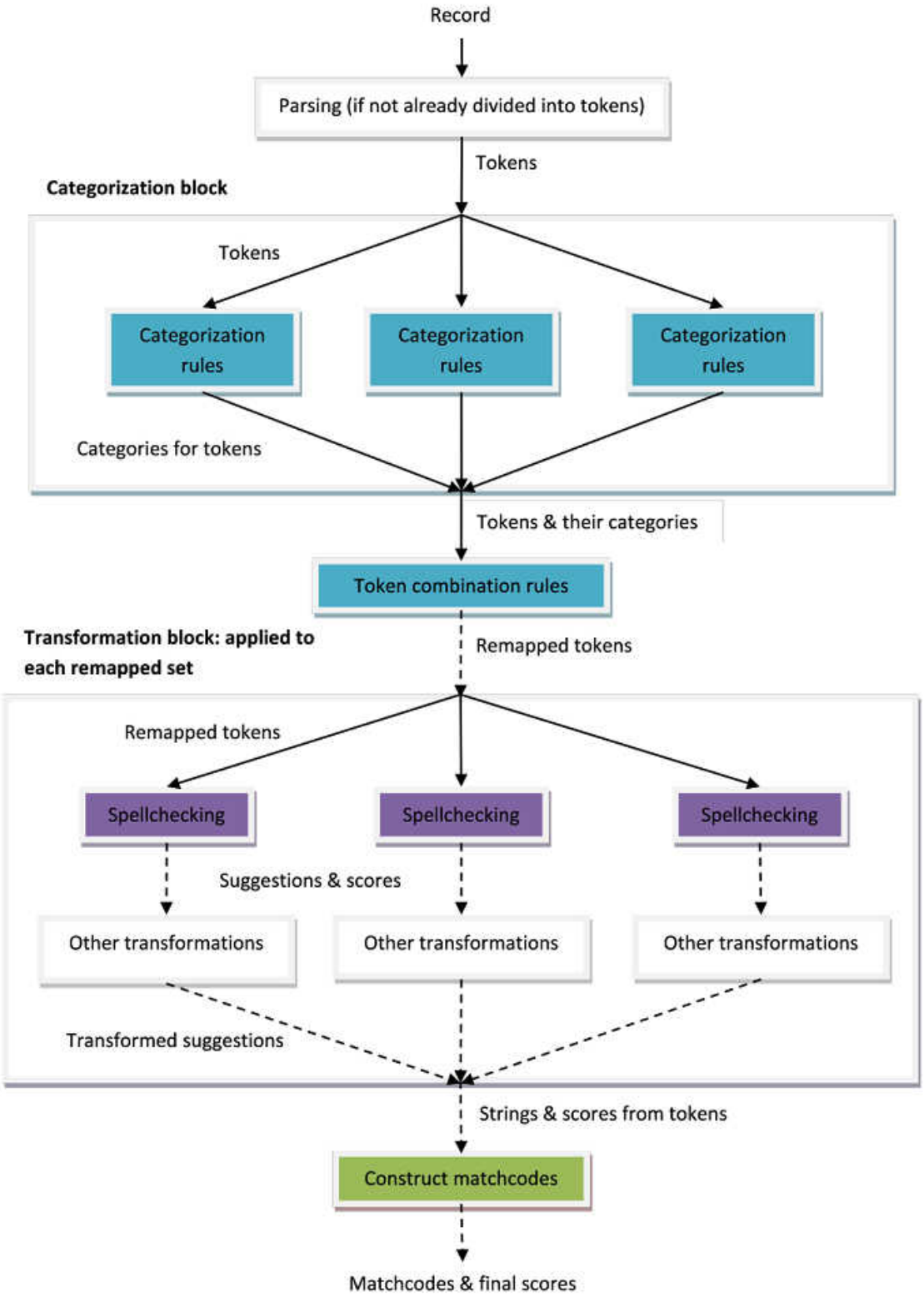
The suggestion-based matching feature is implemented using an internal spelling checker mechanism and (optionally) using frequencies of terms that have been obtained from reference sources or through data mining. Potentially misspelled words generate spelling checker–like "suggestions" from a known dictionary. This notion of spelling errors includes character deletions, insertions, replacements, and transpositions. Whitespace character insertion, casing, and context-dependent pronunciation can also be taken into account.

Each suggestion includes a score that reflects the closeness of that suggestion to the input word based on the operations that transform the input word to the suggestion, and the cost of these operations. Frequency counts obtained from a reference (or otherwise trusted) dataset and then further modified the scores for each suggestion. Therefore, if two suggestions are equally close to the input word, the more frequent suggestion scores higher. The input string is viewed as a potentially corrupted version of the real entity, so the higher the score of a suggestion, the more likely it is to have been "what was really intended."

As with combination-based matching, a single input to a suggestion-based match definition produces multiple output match codes. This means that a given record

can appear in more than one potential cluster. In the case of suggestion-based matching, each match code corresponds to a different suggestion or, more typically, to a mixture of suggestions for the individual tokens in the input string.

Suggestion-based matching can be used in conjunction with combination-based matching, if desired. The following diagram shows a conceptual overview of the match definition flow when both combination-based matching and suggestion-based matching are used. Note that the required computation and storage resources increase when both types of matching are used.


Record
Parsing (if not already divided into tokens)
Tokens
Categorization block
Tokens
Categorization rules
Categorization rules
Categorization rules
Categories for tokens
Tokens & their categories
Token combination rules
Transformation block: applied to each remapped set
Remapped tokens
Remapped tokens
Spellchecking
Spellchecking
Spellchecking
Suggestions & scores
Other transformations
Other transformations
Other transformations
Transformed suggestions
Strings & scores from tokens
Construct matchcodes
Matchcodes & final scores

When to Use Suggestion-Based Matching

The following is a list of some situations where switching from combination-based matching to suggestion-based matching might be useful:

- An increase in computation time, storage, and/or complexity in the de-duplication workflow is acceptable, in exchange for higher accuracy. For example, when compared to a Match Codes node that uses the combination-based Name match definition, a Match Codes node that uses the legacy Name (with Suggestions) match definition runs several times slower and produces more match code output rows. However, it reduces the error rate in the following clustering step.

  Note: In this example, legacy Name (with Suggestions) definition refers to the Matchcodes node that uses the Name (with Suggestions) match definition shipped in CI 2011A in the ENUSA locale.

- The recommended best practice of using low sensitivities for a single match definition and then reducing false matches by combining match codes over multiple data types (for example, Name and ZIP code) is not practical. This is commonly seen when working with records that contain only a single piece of information. For example, an entity name or when the other data types do not contain information reliable or relevant enough to the application at hand. When it is not possible to match over multiple data types, the individual match definition must be more discriminating.
- The records that do not match (but are desired to match) differ from each other at the sub-token or character level. Some of these differences are implicitly handled by the transformation schemes, regexes, and phonetic reduction rules in traditional match definitions. However, with these libraries, it is difficult to address character-level errors in a comprehensive way while avoiding over-generalization. For example, traditional match definitions cannot easily match records that differ by a transposition of characters.

  Note: In situations where the difficulty arises at the inter-token level (that is, from transposition of certain tokens or missing tokens), combination-based matching should be used.

- Behavior that is actually equivalent to allowing a record to appear in more than one cluster. For example, "LINA" should match "LINDA" and/or "LENA", yet "LINDA" should not also match "LENA". This is not possible using traditional matching, because each record must appear in exactly one cluster.
- A numeric score is needed to aid decision making when resolving clusters or selecting survivors. Each suggestion-based match code is assigned a score, and after clustering, these scores are aggregated for the cluster. These scores can be used during manual review of clusters. The presence of scores also paves the way for business rules that automatically resolve clusters and select surviving records.

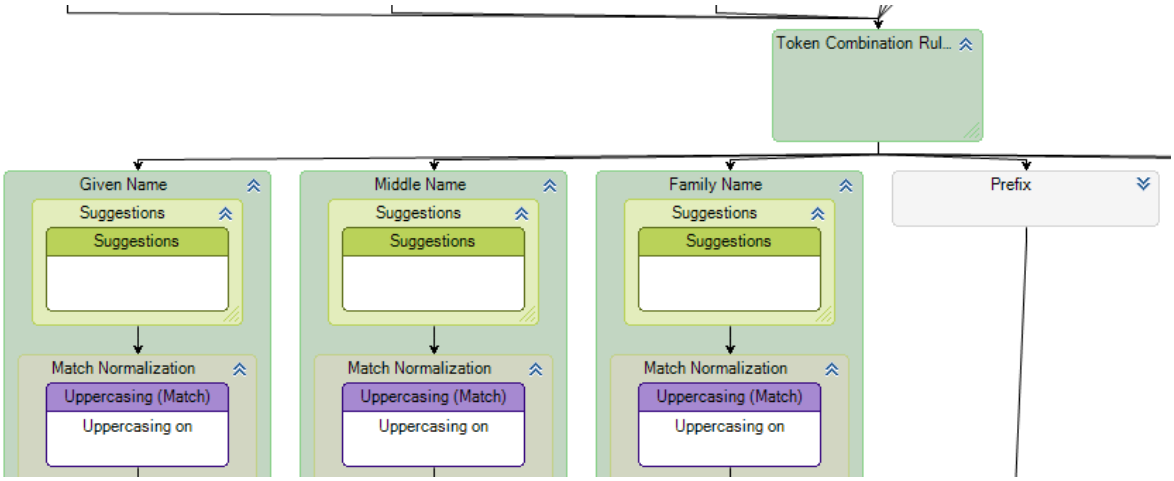
Note: Suggestion-based matching is not available in all locales. In locales that do not support suggestion-based matching, the system does not permit you to add a Suggestions node to a match definition.

Working with Suggestion-Based Match Definitions in Customize

In Customize, two node types are available for match definitions to support suggestion-based matching functionality: the Suggestions node and the Match Score Threshold node. For instructions for configuring these node types, see the following topics:

- Configuring the Suggestions Node on page 183
- Configuring the Match Score Threshold Node on page 189

The following example shows the middle portion of a match definition diagram with Suggestions nodes configured for three tokens.



A match definition is considered "suggestion based" if the Suggestions placeholder group of any token contains a Suggestions node. A newly created match definition has empty Suggestions placeholder groups. Thus, it is still possible to create traditional match definitions, by not inserting Suggestions nodes in any tokens.

Configuring the Suggestions Node

The Suggestions node for each token is where the suggestion-based matching mechanism is configured for that token. Each token can have only one Suggestions node.

Properties

Known words scheme The known words scheme must be specified, and the file should be in the following format:

- The first column contains the words or terms that are expected to exist in the token. Unless casing is important, the word should be in uppercase (as is the usual convention with schemes). The comma character should not be used (such characters are ignored).
- The second column contains the relative frequencies of the terms. These can be actual frequency counts, percentages, or fractions; only the relative magnitudes are important. Frequencies must be nonnegative. This column is optional; it can be left blank or set to zero, or it can be populated for some terms and not others. The higher the frequency, the higher the score of a suggestion. If a term that has no frequency (or zero frequency) is suggested, the score for that suggestion does not include a frequency component. Therefore, a term with zero frequency will score lower than any similar term that has a nonzero frequency (all other things being equal).

When manually creating a known words scheme, the source used to create the vocabulary used by the corresponding parse definition could also provide words for the scheme. However, this scheme is specific to the token in which the Suggestions node is contained, so its contents should be limited to only words that could appear in that particular token.

Fuzziness settings - Each Suggestions node has three fuzziness levels (High, Medium and Low), which each encompass a set of values for many individual configuration parameters. In definitions shipped by SAS in a QKB release, the values underlying these levels are obtained through extensive experimentation with test data appropriate to the particular data type. The High level provides the highest accuracy and the Low level requires the least computation time. The Medium level is a reasonable compromise.

In new user-created definitions, these three preset levels are populated with default values as a convenient starting point. These settings are not guaranteed to perform well, or even sensibly, for arbitrary data. For better results, it can be necessary to configure the individual parameters.

To access the individual configuration parameters, click **Edit**. For more information, see Suggestions Node Advanced Configuration (Fuzziness Settings) on page 185. If any of the individual parameters are edited, the changes are saved under a Custom fuzziness level.

Sensitivity - As with other nodes in the match definition, the Sensitivity slider selects the sensitivity range for which the Suggestions node has an effect.

Suggestions are case insensitive - If this option is selected, then the check box is selected and the incoming token is converted to uppercase at the input to the Suggestions node.

Disable phonetics if suggestions found - If this option is selected, the Phonetics Libraries farther down in the match definition flow does not take effect whenever there is at least one suggestion output by the Suggestions node (other than the input token itself). This is useful when converting an existing match definition that uses Phonetics Libraries because those phonetics rules tend to overly reduce the string.

Suggestions Node Advanced Configuration (Fuzziness Settings)

The individual configuration parameters for a Suggestions node can be adjusted in its advanced configuration.

Properties

Sample pronunciations scheme - The sample pronunciations scheme is used to define the pronunciation of some words so that suggestions can be generated with phonetically equivalent characters. This is optional, but if a scheme is specified, it must be in a specific format and should not be indiscriminately edited. For more information, see Editing the Sample Pronunciations Scheme in the Scheme Builder on page 189.

Note: If no sample pronunciations scheme is specified, phonetic equivalences is not used.

Settings - The **Settings** drop-down list contains the preset fuzziness levels (High, Medium, Low, and Custom, if it exists). Changing this selection sets the active preset level (the level that is used when the definition is run). When the fuzziness level is changed, all the individual parameters corresponding to that level are shown on the **Suggestions**, **Operations**, and **Costs** tabs below the list.

Initially, the Custom fuzziness level does not exist. It is created and made active if any of the parameters on the **Suggestions**, **Operations**, and **Costs** tabs are changed.

Suggestions Tab - The **Suggestions** tab contains general parameters pertaining to the suggestions that is generated for the currently selected preset level.

- **Max number of suggestions per word** - No more than this number of suggestions is generated for each word in the token. For example, if the token contains two words ("OCEAN VIEW"), up to two suggestions might be generated for "OCEAN", and two suggestions for "VIEW". Thus, up to four potential suggestions might be generated for the entire token.
- **Max number of suggestions total** - No more than this number of suggestions is generated for the entire token. For example, if the token contains two words ("OCEAN VIEW"), even if each word could produce two suggestions (resulting in four potential combined suggestions), a maximum of the two top-scoring combined suggestions are shown.
- **Do not make suggestions for words of this length or less** - Because it is often not useful to generate suggestions for very short words, they can be excluded from the spelling checker mechanism. For example, if this parameter is set to 1 or greater, the word "A" does not generate any suggestions other than "A" itself. If this parameter is set to zero, the word "A" could generate "B", "C", "D", and so on.
- **Discard suggestions with score below** - This parameter is a threshold that allows suggestions (for the token) that have low scores to be discarded and not pass through the rest of the token processing. This threshold is applied even if

the maximum number of suggestions for the token has not been exceeded. For example, if this parameter is set to 80.00 and the potential suggestions have scores 100.00, 80.00, and 79.00, only the first two suggestions are shown.

- **Score for unknown word** - If an input word does not exist in the known words scheme, this is the score that is assigned to it by the spelling checker mechanism. The input word results are a suggestion, regardless of what other suggestions are generated.
- **Weight for frequency in scoring** - This parameter controls the contribution of frequency to the score. If this parameter is set to zero, frequency does not affect the score. If the weight is nonzero, more frequent terms ("SMITH") scores higher than less-frequent terms ("SMOTH") even if they are equally close to the input ("SMKTH"). The effect of frequency is not linear. Doubling this weight has very little effect on the score of terms with low frequencies, and approaches double the effect on the score for terms with very high frequencies.
- **Make suggestions for words containing digits** - This parameter determines whether words containing digits (for example, "Win7") are exempted from the spelling checker mechanism.

Operations Tab - The parameters on the **Operations** tab control the types of operations that are allowed in the spelling checker for the selected preset level.

- **Allow phonetic equivalences** - This operation occurs when a character is phonetically equivalent in the input and in the suggestion. Phonetic equivalence is contextual – that is, a character can be pronounced differently depending on what word it is in.

  As long as the sample pronunciations scheme contains words containing the relevant characters, the spelling checker automatically creates equivalence classes for the set of all vowels and some sets of consonants. For example, input "VERDENT" with suggestion "VERDANT" could contain a phonetic equivalence operation because of the automatic phonetic equivalence of vowels.
- **Allow insertion** - This operation occurs when a character is added to the input word. For example, insertion could lead to the suggestion "JOAN" for input "JON".
- **Allow deletion** - This operation occurs when a character is removed from the input word. For example, deletion could lead to the suggestion "JON" for input "JOAN".
- **Allow replacement** - This operation occurs when a character in the input word is replaced by another character. For example, replacement could lead to the suggestion "JOHN" for input "JOAN".
- **Allow transposition** - This operation occurs when two adjacent characters switch positions. For example, transposition could lead to the suggestion "JON" for input "JNO".
- **Allow compound whitespace insertion** - This operation occurs when a whitespace character is inserted in the input word to make two words that form a known compound. For example, if the Known words scheme contains the compound "MARY ANN", whitespace character insertion could lead to the suggestion "MARY ANN" for input "MARYANN".

Costs Tab - The **Costs** tab contains numeric parameters that control the costs of operations and the search strategy by which suggestions are constructed.

As a general guideline, the cost of a particular operation should be set lower for common operations—that is, if the type of error associated with that operation is more commonly encountered in the data. The cost of an operation should also be lower if (for some reason) it is considered more important for that type of error to be caught.

- **Identity operation cost** - This parameter controls the cost of the operation where the same character is used in both the suggestion and the input word. Generally, the cost should be set to a non-positive quantity. For example, input "ANDY" with suggestion "ANDY" consists of four identity operations. If this operation's cost is -0.05, the total cost is -0.20.
- **Phonetic equivalence cost** - This parameter controls the cost of the operation where a phonetically equivalent character is substituted. For example, input "VERDENT" with suggestion "VERDANT" contains one phonetic equivalence operation (vowel equivalence). If this operation's cost is 0.10 and the identity cost is -0.05, total cost is 6*-0.05 + 0.10 = -0.20.
- **Phonetic/spelling conflict cost** - This parameter controls the cost of the operation where the character is the same, but is pronounced differently in the input word and the suggestion. For example, consider the Spanish-origin name "JOSE" (pronounced "ho-zay" in English) compared to the English name "JOE" (pronounced "jo"). Although both words contain "J" and "E", the pronunciation of those letters differs in the two words. Assume that the sample pronunciations scheme contains an entry for both these words (or similar words that contain these sounds). Further assume that the only allowed operations are identity, deletion and phonetic equivalence. For input "JOSE", there could be a suggestion "JOE", with a score resulting from one identity operation ("O"), one deletion ("S") and two phonetic and spelling conflicts ("J" and "E").
- **Insertion cost** - This parameter controls the cost of the insertion operation.
- **Deletion cost** - This parameter controls the cost of the deletion operation.
- **Replacement cost** - This parameter controls the cost of the replacement operation.
- **Transposition cost** - This parameter controls the cost of the transposition operation. This cost is applied once for the pair of transposed characters, and neither transposed character applies an identity cost. For example, input "JNO" with suggestion "JON" contains one identity operation and one transposition operation.
- **Whitespace insertion cost** - This parameter controls the cost of the whitespace character insertion operation.
- **Compound whitespace insertion cost** - This parameter controls the cost of the compound whitespace character insertion operation.
- **First letter uppercasing cost** - This parameter controls the cost of changing the first letter of the input word to uppercase. This parameter is not available if the suggestions are case-insensitive option was selected in the **Suggestions** node properties.

- **Lowercasing cost** - This parameter controls the cost of changing an entire converted to uppercase input word to lowercase. For this cost to be applied, the suggestion must be listed in the known words scheme in lowercase. This parameter is not available if the suggestions are case-insensitive option was selected in the **Suggestions** node properties.

  Note: This applies to an input word that is entirely in uppercase. If only some of the letters in the input word are converted to uppercase, they are treated as individual replacements.

- **Max number of states** - This parameter controls the breadth of searches in the state graph. The graph consists of all possible transitions between characters found in words (including lowercase and uppercase). After this number of states has been checked, the search will end and will return the complete suggestions that have been found before that point. Thus, the larger this setting (relative to the number of states generated from all the terms in the Known words scheme and the input string), the more suggestions are likely to be found. Also see the following discussion of the Cost threshold parameter.
- **Cost threshold** - A running cost total is kept along every search path traversed in the state graph. If, at any time, a certain path exceeds this threshold, that path is abandoned. For example, assume that the insertion cost is set to 0.4, the identity cost is -0.05, and the cost threshold is 0.25. If the input is "BRTHOLOMEW", the running cost at the insertion of "A" is -0.05 + 0.4 = 0.35, which is greater than the threshold. Although the final cost of the suggestion "BARTHOLOMEW" would in theory pass the threshold (because there are nine other identical characters toward the end of the word, giving a final total cost of 10*-0.05 + 0.4 = -0.1), the current running cost total at that state does not pass the threshold. Therefore, that suggestion's path is abandoned. The Cost threshold parameter, together with the Max number of states, is the main determiner of the computation time needed for the spelling checker.

Testing the Suggestions Node

The following display shows the output of the Suggestions node in Customize (assuming no token combination rules). The rule-weighted score is the score after applying the weight of the applicable token combination rule, if combination-based matching is also in use. (If token combination rules are not in use, the rule-weighted score is the same as the raw score.) The rule-weighted score is then passed down into the rest of the token processing, and is used with the other tokens' scores when the complete match codes are assembled.

Note: If token combination rules are in use, you can view the output for a rule by selecting the rule in the rule selector drop-down list in the **Test Window**.

If the match definition option Weight scores by sensitivity is enabled, all rule-weighted scores are further modified by the current sensitivity setting.

Configuring the Match Score Threshold Node

The Match Score Threshold node on page 276 has a single parameter: the Minimum Score Threshold. This threshold value is applied to the combined score for the entire match code as specified in the Matchcode Layout node on page 248. Any match codes that have scores below this value are discarded.

The threshold value should be set to limit the output of the match definition to a reasonable number of suggestions for the typical input string. Because very low-scoring match codes are likely to be ignored during entity resolution anyway, discarding them at this stage saves resources when the match definition is run in a data job.

Testing the Match Score Threshold Node

The **Test Window** output of the Match Score Threshold node is very similar to that of the Matchcode Layout node on page 248. The match codes are shown prior to obfuscation, together with their scores and the token combination rule from which each arose. If there were duplicate match codes generated (that is, from several token combination rules), this is also indicated. Only the match codes that passed the threshold are shown.

Using Suggestion-Based Match Definitions in a Data Job

See the DataFlux Data Management Studio: User's Guide for information about suggestion-based match definitions in a data job.

Editing the Sample Pronunciations Scheme in the Scheme Builder

Note: The sample pronunciations scheme should be edited only by advanced users.

If the optional sample pronunciations scheme is used in a Suggestions node, it must be in the following format:

- The first column of the scheme has a word in lowercase only; any uppercase characters are forcibly converted to lowercase when the speller is built.

- The second column contains the word's pseudo-phonetic equivalent (according to some system of pronunciation respelling, subject to the restrictions in the following text), which represents how the word is pronounced.

Not all known words must be present in this scheme, but a sufficient number of words should be included to cover any pronunciation nuances of the words likely to be associated with the token. As a general guideline, at least 50 distinct words should be included.

Note: Entries in the first or second column need not be unique within the scheme. The same word can be present multiple times with different pronunciations. The same pronunciation can also be present multiple times for different words.

The main challenge with editing the sample pronunciations scheme is in the pseudo-phonetic representation:

- The pseudo-phonetic equivalent of a word can be expressed in any ASCII characters except whitespace character. The character used to represent a sound need not bear any relation to the printed character to which it corresponds.
- It is not necessary to conform to any published pronunciation respelling system or transcription standard, as long as the usage of a character is internally consistent within the scheme.
- The pseudo-phonetic equivalent must be the same length as the word itself.
- The - (hyphen) character indicates "no sound" and is used for padding the length.
- Sounds that are normally considered to be made by a combination of adjacent characters are assigned to only one character. The other characters are assigned "no sound".
- Each pseudo-phonetic character can (in principle) stand for any sound. However, there are conventions that the spell-builder code relies on to create automatic equivalence classes, so these conventions should either be honored or, if deemed undesirable, worked around.
  - The character `&` should never be used.
  - The following characters form an equivalence class that is intended to represent vowel sounds: {`a`, `e`, `i`, `o`, `u`, `A`, `E`, `I`, `O`, `U`, `x`, `W`, `@`, `c`, `^`}.
  - The following characters form an equivalence class that is intended to represent alveolar fricatives: {`s`, `z`, `!`}. Note that the `!` character actually represents an alveolar plosive plus fricative, as in the "z" of "pizza".
  - The following characters form an equivalence class that is intended to represent dental fricatives: {`T`, `D`}.
  - Glottal fricatives and "no sound" form an equivalence class {`h`, `-`}.

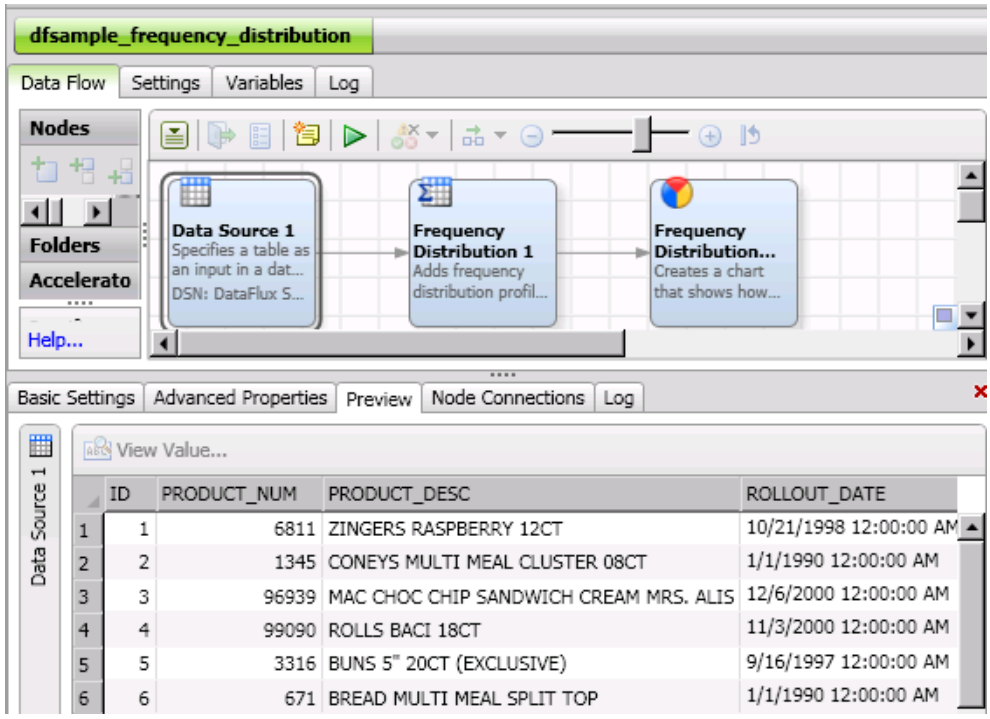
Illustrative Examples - The following examples are taken from a sample English pseudo-phonetics file.

Word	Pseudo-phonetic	Example IPA (for comparison)
aardvark	a-rdvark	ˈɑrdˌvɑrk
abandon	xb@ndxn	əˈbændən
abbreviate	xb-riviet-	əˈbriviˌeɪt
ashamed	xS-em-d	əˈʃeɪmd
them	D-Em	ðɛm
theft	T-Eft	θɛft
quick	kwIk-	kwɪk
brown	brW-n	braʊn
fox	faX	fɒks
jump	J^mp	dʒʌmp
over	ov-R	ˈoʊvər
lazy	lezi	ˈleɪzi
dog	dcg	dɔg
pizza	pi!-x	ˈpitsə
intermezzo	Int-RmE!-o	ˌɪntərˈmɛtsoʊ
interpose	Int-Rpoz-	ˌɪntərˈpoʊz
recline	rIklAn-	rɪˈklaɪn
recoil	rIkO-l	riˈkɔɪl
rouge	ru-Z-	ruʒ
roof	ru-f	ruf
rook	rU-k	rʊk

Previewing, Running, and Reviewing a Data Job

Preview the Nodes and Tables in the Data Job

You should consider previewing the nodes in the data job to ensure the results that you expect are returned when you run the job. For example, previewing the **Data Source** node in a frequency distribution job generates the results shown in the following display:



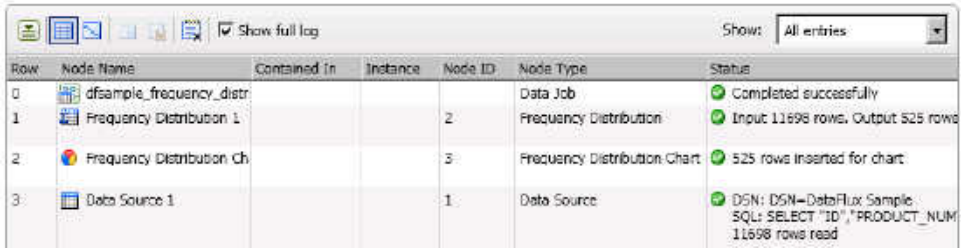
You can preview a node by selecting it and clicking the **Preview** tab. Note that the data viewer in preview displays might display fewer rows than you need to see. If you need to see more rows, open the **Tools** > **Data Management Studio Options** > **DataViewer** and enter an appropriate value in the **Number of rows to show in the preview pane** field.

Finally, you can quickly review the output of a table in a data job by previewing it after the run is successfully completed.

Run the Data Job

After the flow for a data job is complete, you can run and review the output. Perform the following steps:

1. Open the data job, if necessary.
2. Click **Show Details Pane**, if needed, to view the **Log** tab for the data job.
3. Click **Run Data Job** to run the job.
4. Review the log to make sure that all of the nodes in the data job have completed successfully. The following display shows the log for the sample job described in Creating a Data Job on page 195 with the **Show full log** check box selected.



Note that all of the nodes have completed successfully. Also note that the 525 rows that result from the **Frequency Distribution** node were inserted into the chart in the **Frequency Distribution Chart** node.

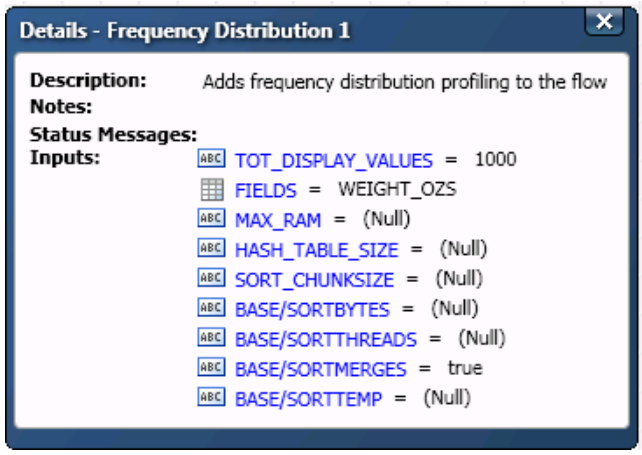
5 If you double-click the first row in the log table, you can see detailed information about the job that includes sections covering inputs, outputs, and job performance information. The following display shows part of the input section for a sample job:

```
Description:    Flow transforming data into a desired form
Notes:
Status Messages:
Variables:
Inputs:         JOB =  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
                       <datafluxdocument class="architectjob" version="1.2">

                        <jobid id="8"/>

                       <propset>
                         <prop name="_DESCRIPTION"/>
                         <prop name="_NOTES"/>
                         <prop name="AutoLayoutRequired">
                           <value dtype="String">False</value>
                         </prop>
                         <prop name="DefaultLayout">
                           <value dtype="String">LeftToRight</value>
                         </prop>
                         <prop name="GroupNodes"/>
                         <prop name="name">
                           <value dtype="String">dfsample_frequency_distribution</value>
                         </prop>
                         <prop name="ShowGridLines">
                           <value dtype="String">True</value>
                         </prop>
                         <prop name="SnapToGrid">
                           <value dtype="String">False</value>
                         </prop>
                         <prop name="STICKYNOTES"/>
                         <prop name="ZoomLevel">
                           <value dtype="String">26</value>
                         </prop>
                       </propset>
```

You can also click the row for a node or table in the job to review the **Details**. The details for a node in the sample job is shown in the following display:



Note that you can double-click an input to review its run-time value.

Finally, you can also see a summary of the log in a single row or review the full text of the Platform log. To see the summarized version, deselect the **Show full log** check box. See Accessing Logs on page 9 to learn how to access the Platform log and the other log files that are provided for DataFlux Data Management Studio.

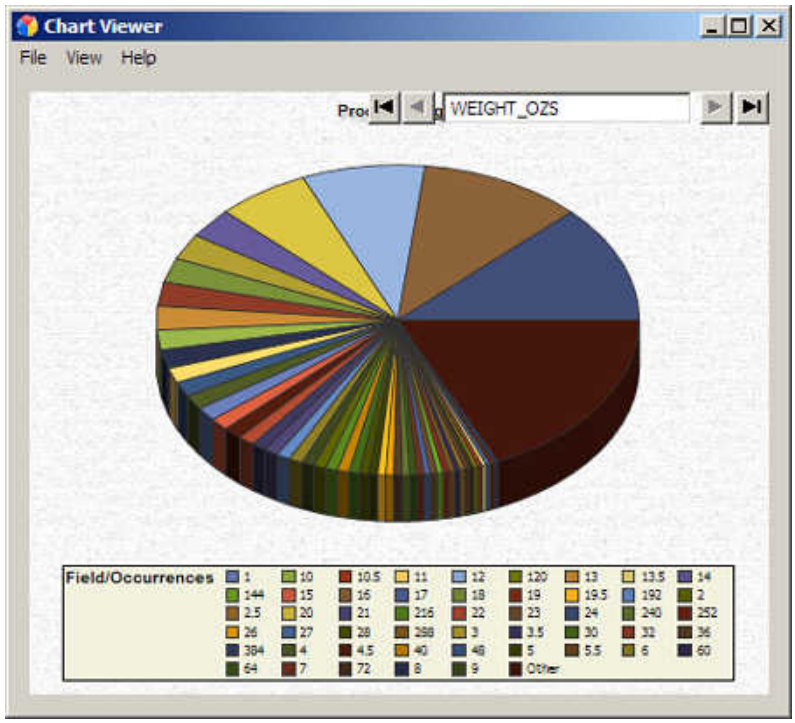
Any temporary files that are generated by the job, such as sort files, are saved to the location that is specified in the BASE/TEMP option in a SAS configuration file, if this option has been set. If the BASE/TEMP option has not been set, temporary files are written to the temporary file location that is specified for the operating system. To see if the BASE/TEMP option has been set, select **Tools** > **Data Management Studio Options** > **Job** > **Advanced**.

Note: You can also execute a data job using batch mode. From the toolbar, select **Actions** > **Run as Batch Job on Management Server...** and follow the instructions in the Export to Data Management Server wizard.

You can also run a **Data Job** node in a process job. Select the **Data Job** node and click **Run Node**. You can review the log entries for the node and its components in the **Log** tab. You can also review the node's output as described above.

Review the Data Job Output

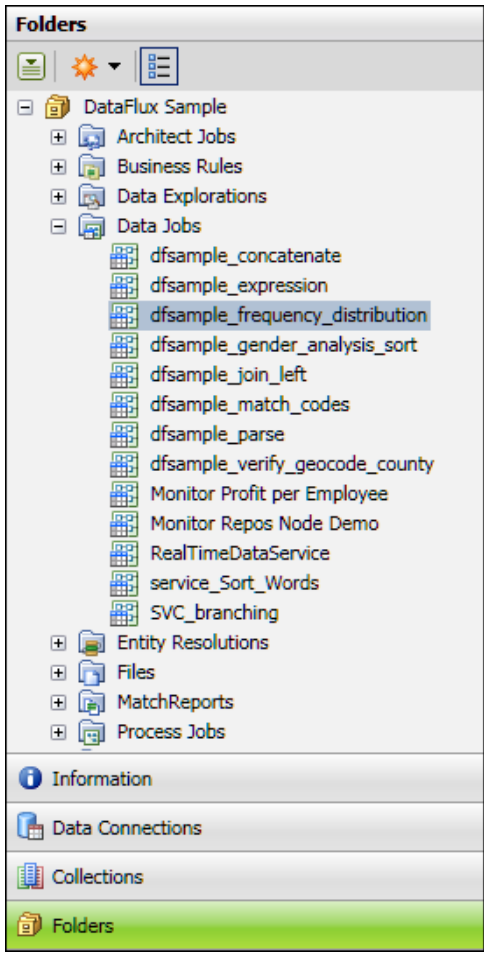
The output displayed by a data job depends, of course, on the data and data nodes included in the job. The frequency distribution data job generates a pie chart that appears in the **Chart Viewer**, as shown in the following display:



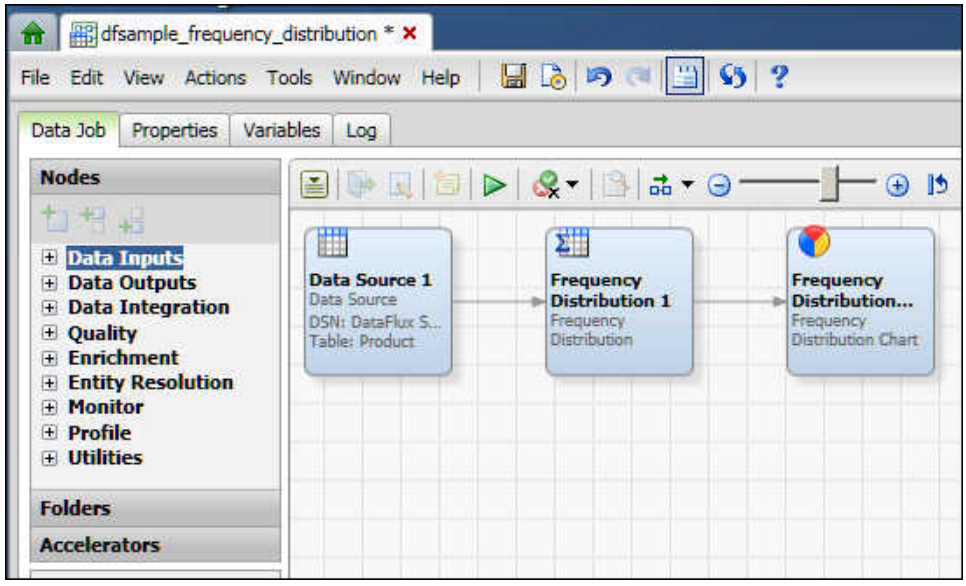
You can right-click anywhere in the **Chart Viewer** to see the pop-up menu. You can use this menu to perform a variety of functions. These functions include changing properties and options for the graph or the data, changing the chart type, and controlling whether the chart tips and legend are displayed.

Creating a Data Job

This section steps you through creating a data job that can be executed independently or can be called by another job. Data jobs illustrated in this section use the data job **dfsample_frequency_distribution**. This is a sample job that is included with the SAS Sample repository.



If you right-click a data job in the **Folders** tree, then select **Open**, the data job opens, as shown in the next display.



On the left side of the data job, the **Nodes** tree lists all of the types of nodes that are available for use in a data job. For an overview of all data job nodes, see **Data Job Nodes** in the *DataFlux Data Management Studio: User's Guide*. On the right side,

you specify a data flow. In the previous display, the flow goes from a source node (Data Source 1), to a processing node (Frequency Distribution 1), to an output node (Frequency Distribution Chart 1).

Plan the Data Job

The first task in creating a data job is to plan the flow. Planning the data flow is helpful whether you are creating a data job in the **Folders** tree or you are adding a **Data Job** node to a process job. Review the goals that you need to reach and determine which data job nodes are most appropriate for you to use. You need components such as the following:

- One or more nodes that specify a data source such as a table or a text file. See the nodes in the **Data Inputs** folder in the **Nodes** tree on the left of the data job. In the sample job **dfsample_frequency_distribution**, a **Data Source** node specifies the **Product** table in the **SAS Sample** connection.
- One or more nodes that process your data. In the sample job, a **Frequency Distribution** node adds frequency distribution profiling to the flow.
- An output node. See the nodes in the **Data Outputs** folder in the **Nodes** tree. In the sample job, the **Frequency Distribution Chart** node creates a pie chart from output of the **Frequency Distribution** node.

After you have a general idea of what nodes you need and how they should be connected in a flow, you are ready to add a new data job.

Add a New Data Job

To add a new, empty data job in the **Data Job** folder in the **Folders** tree, perform the following tasks:

1 Click the **Folders** riser. Then, select **New** > **Data Job** from the **Main Menu**.

2 Enter a name for the data job in the **Name** field.

The following restrictions apply to the name of a job that is deployed to a DataFlux Data Management Server. To avoid problems, you might want to follow these restrictions for all jobs. A job name can contain any alphanumeric characters, white spaces, and any characters from the following list:

```
, . ' [] { } () + = _ - ^ % $ @ ! ~ / \
```

DataFlux Data Management Server does not upload or list a job name with characters other than those cited above.

3 Specify a location for the data job in the **Save** in field.

4 Click **OK** to save the new data job. An empty job opens in the data job.

The next task is to specify a data source for the job.

Specify Source Data

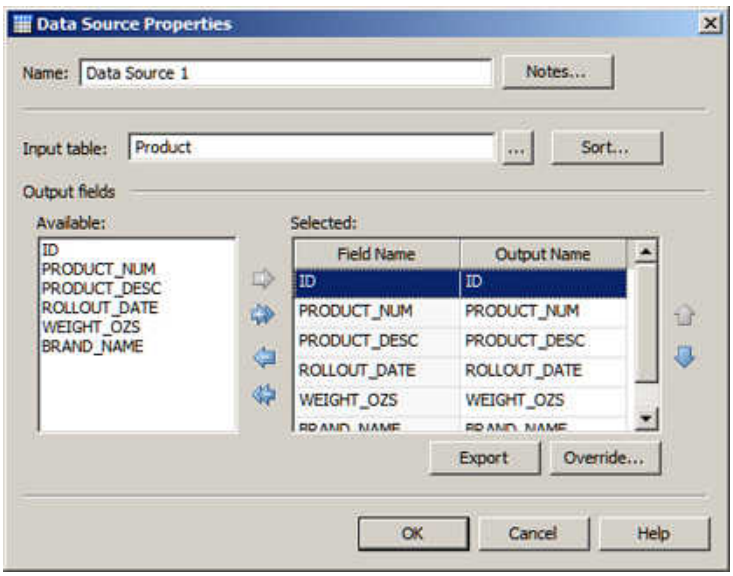
After you have created a new empty job, you can specify a data source for the job. The steps in this section are illustrated with the sample job **dfsample_frequency_distribution**.

1 Add a **Data Source** node to the flow. For the sample job, you would expand the **Data Inputs** folder in the **Nodes** tree on the left of the data job and drag the **Data Source** node into the flow editor on the right.

2 Double-click the **Data Source** node to display its properties. For the sample, the properties for the **Data Source** node would be displayed.

3 Use the properties to select a data source. The fields that are available in that data source would be displayed. For the sample, you would click the selector in the **Input table** field and navigate to the **SAS Sample** connection. Then you would select the **Products** table. The fields that are available in the **Products** table would be displayed.

4 From the fields available in the data source, select the fields that should be available in the job. For the sample, all fields in the **Products** table should be available in the current job. You would click the right double-arrow to move all of these fields from the **Available** area to the **Selected** area, as shown in the next display.



5 Click **OK** to save your changes.

The next task is to add one or more data processing nodes to the job.

Specify Data Processing

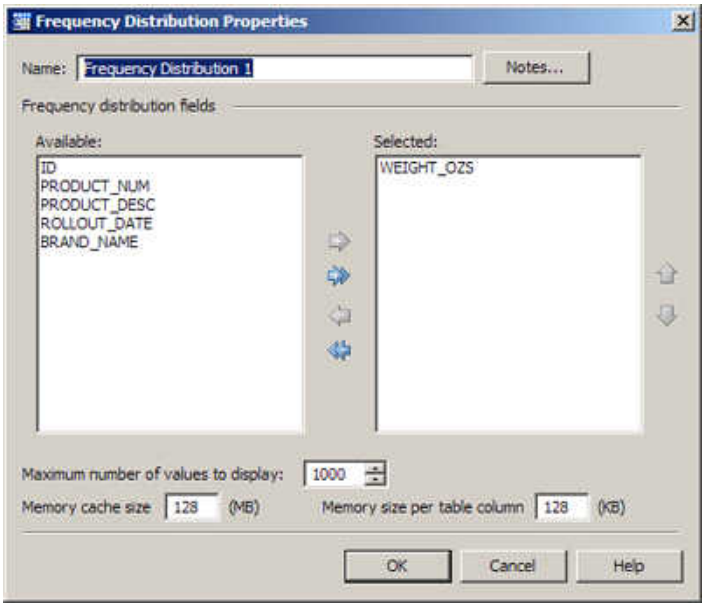
After you have specified a data source for your job, you can add one or more data processing nodes.

1 Add a processing node to the flow. For the sample job, you would expand the **Profile** folder in the **Nodes** tree and drag the **Frequency Distribution** node into the flow editor on the right.

2 Connect the processing node to the source node. Drag the mouse between them until a connecting line appears. For the sample, you would connect the **Data Source** node to the **Frequency Distribution** node.

3 Double-click the processing node to display its properties. For the sample job, the properties for the **Frequency Distribution** node would be displayed. The fields that were selected in the **Data Source** node would be in the **Available** area for the **Frequency Distribution** node.

4 Review the fields and controls in the processing node and make appropriate adjustments. For the sample job, you could decide that you wanted to calculate the frequency distribution of one field, the WEIGHT_OZS field. In that case, you would select the WEIGHT_OZS field in the **Available** area, and then click the single right arrow to move that field to the **Selected** area. Other fields could be left at their default values, as shown in the next display.



5 Click **OK** to save your changes. Add other data processing nodes as desired.

The next task is to add one or more output nodes to the job.

Specify Output

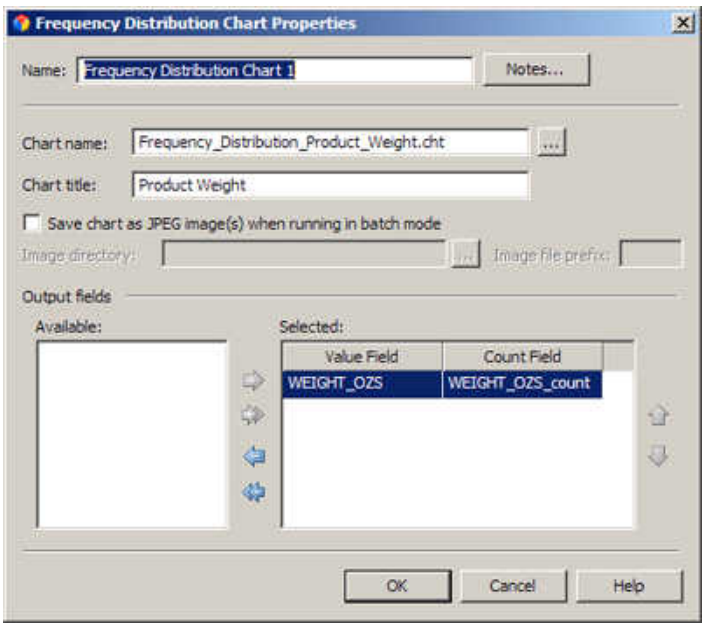
After you have specified data processing operations in your job, you can add one or more data output nodes.

1 Add an output node to the flow. For the sample job, you would expand the **Data Outputs** folder in the **Nodes** tree and drag the **Frequency Distribution Chart** node into the flow editor on the right.

2 Connect the output node to a processing node. Drag the mouse between them until a connecting line appears. For the sample, you would connect the **Frequency Distribution** node to the **Frequency Distribution Chart** node.

3 Double-click the output node to display its properties. For the sample job, the properties for the **Frequency Distribution Chart** node would be displayed. The field that was selected in the **Frequency Distribution** node would be in the **Available** area for the **Frequency Distribution Chart** node: the WEIGHT_OZS field.

4 Review the fields and controls in the output node and make appropriate adjustments. For the sample job, you could decide that you wanted to calculate the frequency distribution of one field, the WEIGHT_OZS field. In that case, you would select the WEIGHT_OZS field in the **Available** area, and then click the single right arrow to move that field to the **Selected** area. You could also specify a file name and a title for the chart, as shown in the next display.



5 Click **OK** to save your changes. Add other output nodes as desired.

After the flow for a data job is complete, you can run the job. See Previewing, Running, and Reviewing a Data Job on page 191. You can also right-click in the data

flow to see a pop-up menu. The options in the menu can help you print, run, and manage the flow.

Manage Data Types

The **Manage Data Types** presents a list of the data types that are available to the locale that you have selected in the tree for your QKB. To view the properties for a data type, click on the data type in the list. The name of the data type is shown in the **Properties** area, along with data type's tokens and the locale with which the data type is associated.

To rename a data type, enter a new name in the **Name** edit box. To edit the data type's tokens, use the **Add a token**, **Edit a token**, and **Delete a token** buttons on the right side of the **Tokens** list. Be aware that if you rename a token, the new name for that token is used in every definition that belongs to the data type you are editing.

Note that you cannot change the locale with which a data type is associated. If you need to associate a data type with a locale, you must create a new data type for that locale. Select the locale in the tree for your QKB, open the **Manage Data Types**, and create a new data type.

To create a new data type, select **Global**, a language, or a locale in the **Quality Knowledge Bases** tree on the left, select **Manage Data Types** in the menu on the lower toolbar on the right. In the **Manage Data Types**, click **New data type** in the toolbar, or select **New data type** in the menu. A new data type is added to the list. The new data type is selected so that its properties are displayed.

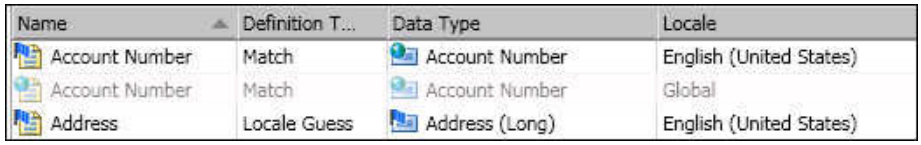
You must add at least one token in order to save the new data type. When creating a new data type, you can choose for the data type to be associated with the currently selected locale or with any of the locale's ancestors . You can associate the new data type with an ancestor if you want for the data type to be available in other locales via inheritance on page 163. Note that after you save the new data type, you cannot change the locale property.

To make a copy of a data type, select a data type and then click the **Duplicate** icon in the toolbar, or select **Duplicate** in the menu. A copy of the data type is added to the list, along with a number added to the end of the data type name.

To delete a data type, select the data type and click the **Delete** icon in the toolbar or select **Delete** in the menu. If any definitions belong to the selected data type, you see a list of those definitions. When you try to delete the data type, you receive a warning that all of the definitions are deleted. Click **Delete** to confirm the deletion, or click **Cancel** to cancel the operation.

Overriding a Definition or Data Type

If If a definition or data type exists in an ancesting on page 163 and you create a new definition or data type with the same name, it overrides the existing definition or data type. For example, an **English (United States)** definition has been added for **Account Number**. The locale definition overrides the **Global** definition of the same name. The **Global** definition is automatically overridden and appears dimmed.



Name	Definition T...	Data Type	Locale
Account Number	Match	Account Number	English (United States)
Account Number	Match	Account Number	Global
Address	Locale Guess	Address (Long)	English (United States)

If a data type is overridden, then any definitions associated with the selected locale that previously belonged to the data type associated with the ancestor belongs to the newly created data type.

Note: Differences in the tokens that make up the new data type and the ancestor data type, cause problems when you try to use definitions that belonged to the ancestor data type.

Accordingly, it is best not to override a data type unless your locale does not already contain any definitions that use that data type.

If a definition is overridden, then the definition associated with the locale is available for selection in a node in a data job, but the definition associated with the ancestor is not available. Remember that the data job user does not notice this change. Inheritance is transparent to a data job user. The data job user sees only a single listing for the definition.

7

QKB Usage Notes

QKB Usage Notes

General QKB

Locating QKB documentation

To access the documentation for a QKB, see Accessing Your QKB Documentation on page 8.

Compatibility of QKB and DataFlux Data Management Studio releases

When a QKB is released, it might contain definitions that require a newer version of DataFlux Data Management Studio. Accordingly, to check for compatibility, review the System Requirements for your QKB to see what version of DataFlux Data Management Studio it requires.

Data Quality warning for incompatible version

If you reference a QKB definition in DataFlux Data Management Studio and the version is incompatible with the version that you are using, you get an error like the following example.

```
Failure loading the Locale ENUSA with error(s):
|[117] Data Quality - Warning: 'Match' definition 'Name
(Probabilistic)' has an incompatible version '9.1'. This definition
may be unavailable at run-time."
```

To check for compatibility, review the System Requirements for your QKB to see what version of DataFlux Data Management Studio it requires.

Updating QKBs that are referenced in a job that is being edited

As described in Previewing, Running, and Reviewing a Data Job on page 191, , you can use the **Preview** tab to see the output of a given node in the job flow. If one of the nodes in the job references a QKB, then the output for that node in the **Preview** tab reflects any QKB processing that is specified for the node. While you are editing a job, it is possible for you or someone else to use the customize features in DataFlux Data Management Studio to edit the QKB that is referenced by a node. The impact of doing this varies, depending on whether you or someone else edits the referenced QKB.

Suppose that you want to edit the QKB that is referenced by the selected node in the **Preview** tab. You could do that in the same instance of DataFlux Data Management Studio. After modifying and saving the QKB, if you return to the job, the **Preview** tab for the selected node will not be automatically updated to reflect the changes made to the QKB. You could select a different node in the job then select the node that references the QKB in order for the **Preview** results to be updated and reflect the changes made to the QKB.

It is possible, but not recommended, to use a separate instance of DataFlux Data Management Studio to edit and save a QKB that is referenced by the selected node in the **Preview** tab. If this situation does arise, the person who edits the job must select **Tools** > **Reload Active QKB** to see the effects of modifications to the referenced QKB.

QKB Definition

Suggestions Node depends on the locale of the QKB definition

The system does not permit you to add a **Suggestions** node to a match definition when the match definition is defined within a locale which the suggestion-based matching feature does not support.

QKB Merge

Merging CI 2010B and later to CI 2010A and earlier versions

The QKB file formats were changed in CI 2010B. It is not possible to merge a QKB whose version is CI 2010B or later (source QKB) into a QKB that has an earlier version than CI 2010B (destination QKB). It is possible to merge a QKB that has a version earlier than CI 2010B into a QKB of version CI 2010B and later (destination QKB).

Merge behavior

The following table explains the merge results when a definition, data type, or library file exist in both the source QKB and the destination QKB.

Has the QKB been customized?		Merge Behavior	
Source QKB	**Destination QKB**	**Definitions and Data Types**	**QKB Library Files**
NO	NO	No change to the destination QKB.	No change to the destination QKB.
YES	NO	■ The tag '`(backup xxx)`' is appended to the name of the definition or data	■ The tag '`(backup xxx)`' is appended to the name

		type in the destination version. ■ The source version of the definition or data type is moved into the destination QKB.	of the QKB library file in the destination version. ■ The source version of the QKB library file is moved into the destination QKB.
NO	YES	■ The tag '(backup xxx)' is appended to the name of the definition or data type in the destination version. ■ The source version of the definition or data type is moved into destination QKB.	■ The destination version is persevered and not renamed. ■ The source version of the QKB library file is not moved into the destination QKB and therefore does not exist in the destination QKB.
YES	YES	■ The tag '(backup xxx)' is appended to the name of the definition or data type in the destination version. ■ The source version of the definition or data type is moved into the destination QKB.	■ The tag '(backup xxx)' is appended to the name of the QKB library file in the destination version. ■ The source version of the QKB library file is moved into destination QKB.

Note: When '(backup xxx)' is appended to a name, the 'xxx' is a numerical value starting with '001'. It increases in value to ensure no duplication in names occurs.

QKB Difference Viewer

Viewing QKB Differences can take a significant amount of time

Viewing QKB differences can take a significant amount of time, from several minutes to several hours. Many factors can cause the viewer to take a significant amount of time. Some of these include machine speed, programs and services running, amount of memory available, and where the QKB is located. When the difference calculations run, an indicator is displayed that shows which file the Diff Viewer is viewing. When the calculation is complete, "Completed" appears at the bottom.

Impact of Gender Analysis enhancements

The Gender Analysis definition was enhanced in DataFlux Data Management Studio 2.6 to allow for different chop tables, regex libraries, and vocabularies to be used within the token groups. Compare an earlier version of a Gender Analysis definition to a definition modified in version 2.6 (or later). The **QKB Difference Viewer** shows that the chop tables, regex libraries, and vocabularies have been removed from the definition and added to each token.

Parse Definition Quick Editor

General

Why would I use the Parse Definition Quick Editor

The Parse Definition Quick Editor enables you to create a new parse definition using a step-by-step process. When you view the results of the new parse definition you can determine the changes you need to make. You can also use the Parse Definition Quick Editor to apply existing parse definitions to your data and edit using the chop table, regex, grammar, or vocabulary components.

Once I have created a Parse Definition using the Parse Definition Quick Editor, what else should I configure to complete the parse definition process?

You need to open Customize and set your Preprocessing and Token Mapping to have your new parse definition available.

Note: Both Customize and the Parse Definition Quick Editor need to access the QKB.

What data sources are available in the Parse Definition Quick Editor for incoming data?

You can access an ODBC data source, SQL Query, Delimited Text File, Fixed Width Text File, SAS Data Sets, and Profile Reports.

How are data sources configured?

You configure your data sources the same way as you configure them for the DataFlux Data Management Studio data job input nodes.

What are Basic and Derived Categories?

A **basic category** is a category that represents a single word. Basic categories are the basic building blocks of Grammar rules. Every basic category in a Grammar corresponds to a category in an ordered word list. A **derived category** is a category composed of one or more other categories. The makeup of a derived category is described using rules.

Are word entries automatically re-parsed and updated with the changes in a parse definition?

To include all parse changes, click **Options** > **Refresh Parse Results** (from the main menu). This enables you to re-parse the data content with all the changes. You are prompted to save the parse definition as well.

When I have the Parse Definition Quick Editor open, can I have other vocabularies, chop tables, or regex libraries open within the parse definition?

The **Parse Definition Quick Editor** does not allow you to make changes directly to the vocabularies, chop tables, or regex libraries within a parse definition. To do this, use Customize.

What is a root category?

The root category is where all other rules (basic and derived) begin.

Common Errors

Error	Description
Please select a locale before creating a new definition	You must install a Quality Knowledge Base (QKB) and select the locale that you want available in the **Parse Definition Quick Editor** prior to creating a new definition.
Unable to load locale	You cannot have more than one instance of the **Parse Definition Quick Editor** running at one time. You also cannot have Customize and the **Parse Definition Quick Editor** open at the same time. Both tools require access to the QKB.

Error	Description
You cannot create left recursive rules	This error message appears when you try to assign a word to a derived category to itself under the **Categorization and Rule Building** tab.
No Parse Definition Solution	This message appears when the **Categorization and Rule Building** tab cannot apply a grammar rule to an entry.
An error occurred while setting the Parse Definition sought category: Unable to determine sought category index	Check your grammar abbreviation, you might have incorrectly entered the abbreviation.

Categorization and Rule Building

What is Categorization and Rule Building?

Click the **Categorization and Rule Building** tab to view the result after your new parse definition is applied to your data. This table provides:

- **Word Count** - the number of words in the sentence
- **Categorized Words** - the words matched with words in the vocabulary
- **Categorization Status** - the percentage of words from the total number of words found to have a category in the vocabulary
- **Root Category Rule** - whether a particular sentence matched a specified pattern rule or not

What root category rule is assigned when there is no rule matched?

When there is no rule matched, you see the message, "No Parse Solution" next to the entry.

What does "No Parse Definition Solution" mean?

This message appears when the **Categorization and Rule Building** tab cannot apply a grammar rule to an entry.

What does the Word/Category fields listed at the bottom of the Parse Definition Quick Editor mean when I select an entry in the Categorization and Rule building tab?

This illustrates how the **Parse Definition Quick Editor** parsed entry finds the correct grammar rule for the selected entry. Each step displays how the final definition arrived at the results from the previous definitions and processed components.

How can I narrow down the number of entries under Categorization and Rule Building?

Create a filter, click **Edit** > **Filter List**.

Sometimes there is no category listed for words that are defined in a vocabulary, Why?

When an entry does not fall under a particular rule category, it is difficult to select the correct category. When this happens there is no category associated with the entry.

Does the Parse Definition Quick Editor Have support for categorization regexlibs?

The **Parse Definition Quick Editor** does not support categorizing regexlibs.

When I remove a rule from the Root Category the derived rule becomes unlisted. Is this expected behavior?

Yes, this is expected. When you remove a rule from Root Category, only the basic types remain in the list.

The Parse Definition Quick Editor Categorization and Rule tab does not display derived rules. Is this expected behavior?

Parse Definition Quick Editor does not display the derived rules unless the derived rule falls into a root category.

Element Analysis

What is Element Analysis?

Element Analysis shows how often each word is listed along with the category, based on the parse definition vocabulary.

Can I assign a basic category to a word?

Yes, right-click on an entry and select **Assign Basic Category** > **Create Category**.

How do I unassign an existing basic category?

To unassign a basic category, right-click on an entry and select **Unassign Basic Category**.

Why is there no category next to an entry, but the entry falls within a grammar?

Some word entries are not included in the vocabulary associated with a grammar. However, for these entries the grammar assigns a default guess category. Based on that default category the grammar rule is still applied. The entries that are not listed in the vocabulary are not assigned a category in **Element Analysis**.

Why are some numbers listed under the NUM category in Element Analysis while others do not include numbers?

The category list on the **Element Analysis** tab is built from the vocabulary. If the number does not appear in the vocabulary with the NUM category assigned, then the NUM category does not appear on the **Element Analysis** tab. Some numbers appear correctly categorized as NUM on the **Categorization and Rule Building** tab, even though they are not in the vocabulary. This happens because some parse definitions have a categorization regexlib that can match numbers with the NUM category.

Can I unassign basic categories from multiple entries at one time, especially when the entries have the same type?

No, you must unassign basic categories one at a time.

Filters

What are filters?

Filters enable you to narrow down entries shown in the **Categorization and Rule Building**, **Element Analysis**, and **Vocabulary** tabs.

How much can I apply filters?

With filters you can build "AND" logic with conditions on any of the available columns within each tab. The filter operations change depending on the data type of the content, string, numeric, or date type.

Can I use regular expressions in filtering?

Yes, to use a regular expression when creating filters use the "LIKE" operation, which enables you to apply a regular expression.

Can I use "OR" logic in filters?

This functionality is not available at this time.

Which Parse Definition Quick Editor tabs use filtering?

You can filter your content on the **Categorization and Rule Building**, **Element Analysis**, and **Vocabulary** tabs.

Is filtering case sensitive?

By default, filtering is case insensitive unless you select **Match Case**.

Chop Table

What is Chop Table?

The **Chop Table** in **Parse Definition Quick Editor** provides a similar function as the separate **Chop Table Editor**. You can edit and create chop tables within the parse definition.

Does the Chop Table option in the Parse Definition Quick Editor provide the same functionality available in Chop Table Editor?

Chop Table in the **Parse Definition Quick Editor** does not provide the ability to test the chop table like the **Chop Table Editor**.

Do changes to the Chop Table appear immediately on the Categorization and Rule Building tab?

Yes, the changes immediately appear on the **Categorization and Rule Building** tab. You can also click **Options** > **Refresh Parse Results**.

I can select only one item at a time in Chop Table Editor. Is this by design?

Yes, you can make only changes to one item at a time.

How do I make the font larger or smaller?

Use the toolbar options, Increase Font Size and Decrease Font Size to adjust the font.

Can I locate a character based on the decimal or hexadecimal value?

Yes, to locate a character using the decimal or hexadecimal value, click **Search** > **Go To**.

What is implicit separation?

Implicit separators are an optional feature of string chopping. For each parse definition, you can enable or disable implicit separators. If implicit separators are enabled, a separator mark is placed wherever the classification of a character differs from the classification of the previous character in a string. See Chop Table Editor - Character Level Options on page 24 for additional information.

Normalization Regexlib

What is Normalization Regexlib?

The **Normalization Regexlib** in the **Parse Definition Quick Editor** provides similar functionality as the separate **Regex Editor**. You can edit and create new regex libraries within the parse definition.

Does the Normalization Regexlib option in the Parse Definition Quick Editor provide the same functionality available in the Regex Editor?

Normalization Regexlib in the **Parse Definition Quick Editor** does not provide the ability to test your regex libraries like the **Regex Editor**.

Do changes to the Normalization Regexlib appear immediately on the Categorization and Rule Building tab?

Yes, the changes immediately appear on the **Categorization and Rule Building** tab. You can also click **Options** > **Refresh Parse Results**.

Where can I find more information about using regular expressions in SAS products?

For additional information about creating regular expressions, refer to the *SAS Expression Language: Reference Guide*.

Grammar

What is Grammar?

The **Grammar** tab in the Parse Definition Quick Editor provides the same functionality as the separate Grammar Editor. You can edit and create new grammars within the parse definition.

Does the Grammar option in the provide the same functionality as the separate Grammar Editor?

Yes, both Grammar options provide the same functionality.

Vocabulary

What is vocabulary?

The **Vocabulary** tab in the **Parse Definition Quick Editor** provides similar functionality as the separate **Vocabulary Editor**. You can edit and create new vocabularies for the parse definition.

Does the Vocabulary tab in the Parse Definition Quick Editor provide the same functionality as the Vocabulary Editor?

Vocabulary does not provide the option to import vocabularies like the Vocabulary Editor.

How can I see all the different categories that I have for vocabularies?

Refer to your Grammar basic categories, which includes your vocabulary categories.

If you have two vocabularies and the same word is in each vocabulary with different categories, does it assign both categories?

Yes, assuming the "stop if found" flag is not set on the first vocabulary.

What if the word appears in two vocabularies with the same category but different likelihoods?

That category appears one time and the last likelihood encountered is used (that is, the duplicate overwrites the original).

Note: This situation tends to cause confusion; it should be avoided, if possible.

Creating a New Grammar and Rule

What components can I reuse or start with when building a new rule?

You have an option to begin building a new rule with an existing parse definition and regex library.

Can I start creating a new parse definition without using any regex libraries or chop definitions?

You can create a parse definition without a regex library but you must have at least a chop table definition.

How do I start creating grammars?

From the **Categorization and Rule Building** tab, click an entry to select. At the bottom of the screen, you should see a Word/Category listing for the entry. Select a word, right-click and select **Assign Basic Category**. You can follow these steps to create a derived category and assign that derived category to a root category.

How do I add a basic category to a word?

To assign a word to a basic category from **Categorization And Rule Building** tab and **Element Analysis** tab, select the word, right-click and select **Assign Basic Category**.

How do I create derived categories from Categorization and Rule Building?

You can create derived categories after you have assigned basic categories to your words. Select a basic category or multiple basic categories, right-click and select **Add Rule To Derived Category**.

Can I create recursive rules to avoid repetition?

You can create right recursive rules like the following rules.

- Address
- Street Address

But not left recursive rules like the following rules.

- Address
- Address Street

Left recursive rules yield an infinite loop.

What does the message "You cannot create left recursive rules" mean?

This error message appears when you try to assign a word to a derived category to itself under the **Categorization and Rule Building** tab.

Can I change the chopping of a word from the Categorization and Rule Building tab?

Yes, under Word/Category, select the word then right-click and select **Adjust Chopping**. Click the operation drop-down list and select USE, SUPPRESS, or TRIM for this word.

Import and Export QKB Definitions

Avoid using the option, "Replace existing items of same name if possible"

In the current release, avoid using the Replace existing items of same name if possible option on the second page of the Import Definitions wizard. In some cases with the option enabled, the import can fail with no errors reported.

Exporting from CI 2010A or earlier

If you export from QKB CI 2010A (or earlier) and then import into a QKB CI 2010B (or later), then the exported libraries (in the L1, L2, LS, LC or L5 encoding) have mangled names. They are placed into the Global locale of the imported QKB.

Name mangling

If an imported definition has the same name as a definition in the target QKB, and the two definitions are incompatible, the name of the imported definition is "mangled" to prevent the imported definition from overwriting the target definition.

A mangled definition name has the following format:
`<original_name> (<unique_number>)`

Where:

- `<original_name>` is the original name of the imported definition.
- `<unique_number>` is a counter number that is incremented as needed to ensure that the resulting definition name is unique.

When you import a QKB definition, you also import its libraries, such as schemes or chop table libraries. As with definitions, if an imported library has the same name as a library in the target QKB, the name of the imported library is mangled to prevent it from overwriting the target library.

A mangled library name has the following format:
`<original_name> (<unique_number>) (<iso_code>)`

Where:

- `<original_name>` is the original name of the imported library.
- <unique_number> is a counter number that is incremented as needed to ensure that the resulting library name is unique.
- <iso_code> is the ISO code of the QKB locale with which the library is associated (for example, ENUSA for the English (United States) locale).

For example, suppose definition Foo is associated with the English (United States) locale, and Foo uses a scheme named Bar, which is also associated with the English (United States) locale. Assume Foo and Bar are exported and are later imported into a QKB that also contains Foo and Bar. If the imported items are determined to be incompatible with the target items, then the names of the imported items are changed to Foo (2) and Bar (2) (ENUSA), respectively.

How the Import Definitions wizard counts imported definitions

Status messages on the last page of the import wizard show the number of definitions that have been added, mangled, or not added. In these messages, each incoming definition is counted as one. Accordingly, if the export archive contains a single definition that is imported into six locales, it counts as one definition, not six.

If a definition was not added anywhere, it counts as one definition not added. If the definition was mangled in any of the locales, it counts as one definition mangled.

Otherwise, if the definition was added in any of the locales, it counts as one definition added.

Importing definitions

When you import definitions, you must ensure that the version of DataFlux Data Management Studio that you use to perform the import is the same or newer than the version of DataFlux Data Management Studio that was used to export the definitions. The version includes any fixes or patches.

Appendix 1

Appendix

Definition Head Nodes

Definition Head Node

Case Definition Head Node

The first node of a case definition shows the name of the definition.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Output

None.

Extraction Definition Head Node

The first node of an extraction definition shows the name of the definition. In this node you can choose to extract substrings into multiple tokens to resolve ambiguities in the input string.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Allow extraction into multiple tokens - Check this box to extract overlapping segments of the input string into multiple tokens when they are attributed to multiple patterns. This functionality helps you resolve ambiguities in the input string. For example, suppose the input string “gold ring” is passed to an extraction definition that searches for text to insert into the following tokens:

- Colors
- Materials
- Additional info

The input word “gold” could be associated with the tokens Colors and Materials. By default, the extraction definition finds the best single token for this word. It this case it could be beneficial to apply the word to two tokens.

Note: Selecting this check box can lengthen processing time.

Output

None.

Gender Analysis Head Node

The first node of a gender analysis definition shows the name of the definition and defines the available genders.

The following **Definition** nodes use this definition type.

- Gender Analysis Definitions on page 227

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Table with 2 columns:

- Value - the gender abbreviation
- **Description** - the meaning of the gender abbreviation

Default value - Use the menu to change the default gender.

Output

None.

Note: The new gender analysis definitions automatically have M (Male), F (Female), and U (Unknown) genders populated in this node, with U being the default.

Identification Analysis Definition Head Node

The first node of an identification analysis definition shows the name of the definition and lists the available classes of entities.

The following **Definition** nodes use this definition type.

- Identification Analysis Definitions on page 228

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications. The list of possible identities is entered here.

Table with 2 columns:

- **Value** - the abbreviation of the identity value
- **Description** - the meaning of the identity abbreviation

Default value - Use the menu to change the default identity.

Output

None.

Language Guess Definition Head Node

The first node of a language guess definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230

Properties

None.

Output

None.

Locale Guess Definition Head Node

The first node of a locale guess definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Locale Guess Definitions on page 231

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Select the last locale given if no score can be computed - Check this box to choose a response to inconclusive computations.

Output

None.

Match Definition Head Node

The first node in a match definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Weight matchcode scores by sensitivity - Select this check box to display all of the scores for this match definition with a weighted value: displayedScore = baseScore x (currentSensitivity/100).

Output

None.

Parse Definition Head Node

The first node of a parse definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Parse Definitions on page 236

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Output

None.

Pattern Analysis Definition Head Node

The first node of a pattern analysis definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Pattern Analysis Definitions on page 238

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Output

None.

Standardization Definition Head Node

The first node of a standardization definition. Shows the name of the definition.

The following **Definition** nodes use this definition type.

- Standardization Definitions on page 239

Properties

Surface this definition in applications - Check the box to allow the definition to be available to external applications.

Output

None.

Definition Nodes

Case Definitions

A case definition transforms a string by changing the case of any (zero or more) of its characters.

Input: a string

Example: `"john james mcdonald"`

Output: transformed string

Example: `"John James McDonald"`

Hierarchy	Node Group	Container Group	Count
1	Case Definition Head Node on page 220		1
2	Preprocessing Regex Library Group on page 267		1
2.1	Preprocessing Regex Library Node on page 267	Preprocessing Regex Library Group	at most 1
3	Chopping Group on page 243		1

Hierarchy	Node Group	Container Group	Count
3.1	Chop Table Node on page 243	Chopping Group	at most 1
4	Familiar Words Scheme Group on page 273		1
4.1	Familiar Words Scheme Node on page 273	Familiar Words Scheme Group	at most 1
5	Familiar Patterns Group on page 263		1
5.1	Familiar Pattern Regex Library Node on page 263	Familiar Patterns Group	at most 1
6	Default Casing Node on page 241		1
7	Concatenation Node on page 244		1
8	Postprocessing Group on page 268		
8.1	Postprocessing Regex Library Node on page 269	Postprocessing Group	at most 1

Extraction Definitions

An extraction definition extracts parts of the input string and assigns them to corresponding tokens of the associated data type.

Input: a string

Example: `"100 Slightly used green Acme XJF-100 raygun $100 c/w lots of shiny buttons"`

Output: mapping between tokens and substrings

Example:

- Quantity => `100`
- Brand => `"Acme"`
- Model => `"XJF-100"`
- Color => `"green"`
- Price => `"$100"`
- Description => `"Slightly used raygun c/w lots of shiny buttons"`

Hierarchy	Node Group	Container Group	Count
1	Extraction Definition Head Node on page 220		1
2	Preprocessing Regex Library Group on page 267		1
2.1	Preprocessing Regex Library Node on page 267	Preprocessing Group	0 or more
3	Chopping Group on page 243		1
3.1	Chop Table Node on page 243	Chopping Group	1
4	Table-Based Extraction Group on page 246		1
4.1	Extraction Scheme Node on page 245	Table-Based Extraction Group	0 or more
5	Pattern-Based Extraction Group on page 245		1
5.1	Morph Analysis Group on page 249	Pattern-Based Extraction Group	1 (*)
5.1.1	Lookup Normalization Group on page 257	Morph Analysis Group	1 (*)
5.1.1.1	Uppercasing Node on page 241	Lookup Normalization Group	1 (*)
5.1.1.2	Normalization Regex Libraries Group on page 270	Lookup Normalization Group	1 (*)
5.1.1.2.1	Normalization Regex Library Node on page 270	Normalization Regex Libraries Group	0 or more (*)
5.1.2	Vocabularies Group on page 286	Morph Analysis Group	1 (*)
5.1.2.1	Vocabulary Node on page 287	Vocabularies Group	1 or more (*)
5.1.3	Categorization Regex Libraries Group on page 271	Morph Analysis Group	1 (*)

Hierarchy	Node Group	Container Group	Count
5.1.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more (*)
5.1.4	Number Check Node on page 258	Morph Analysis Group	1 (*)
5.1.5	Default Categories Node on page 242	Morph Analysis Group	1 (*)
5.2	Pattern Recognition Group on page 264	Pattern-Based Extraction Group	1 (*)
5.2.1	Pattern Logic Node on page 259	Pattern Recognition Group	1 or more
6	Token Mappings Node on page 280		1

Gender Analysis Definitions

A gender analysis definition determines the gender of the individual in the input string.

Input: either a string or, optionally, preparsed input

Example: `"John James McDonald"`

Output: a gender

Example: `"Male"`

Hierarchy	Node Group	Container Group	Count
1	Gender Analysis Definition Head Node on page 221		1
2	Tokenization Node on page 279		1
3	Token Configuration Node on page 280		1
4	Gender Token Group on page 246		1 or more (*)
4.1	Chopping Group on page 243		1

Hierarchy	Node Group	Container Group	Count
4.1.1	Chop Table Node on page 243	Chopping Group	at most 1
4.2	Morph Analysis Group on page 249		1
4.2.1	Lookup Normalization Group on page 257		1
4.2.1.1	Uppercasing Node on page 241		1
4.2.1.2	Normalization Regex Libraries Group on page 270		1
4.2.1.2.1	Normalization Regex Library Node on page 270		0 or more
4.2.2	Vocabularies Group on page 286		1
4.2.2.1	Vocabulary Node on page 287	Vocabularies Group	0 or more
4.2.3	Categorization Regex Libraries Group on page 271		1
4.2.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more
5	Scoring Node on page 275		1

(*) up to maximum number of tokens in data type

Identification Analysis Definitions

An identification analysis definition identifies the input string as referring to a particular predefined class of entity; for example, an individual versus an organization, or type of vehicle (car versus truck versus motorcycle).

Input: a string

Example: `"John James McDonald"`

Output: a class

Example: `"Individual"`

Hierarchy	Node Group	Container Group	Count
1	Identification Analysis Definition Head Node on page 221		1
2	Table-Based Identification Group on page 248		1
2.1	Familiar Phrase Scheme Node on page 247	Table-Based Identification Group	0 or more
3	Preprocessing Regex Library Group on page 267		1
3.1	Preprocessing Regex Library Node on page 267	Preprocessing Regex Library Group	0 or more
4	Chopping Group on page 243		1
4.1	Chop Table Node on page 243	Chopping Group	1
5	Morph Analysis Group on page 249		1
5.1	Lookup Normalization Group on page 257	Morph Analysis Group	1
5.1.1	Uppercasing Node on page 241	Lookup Normalization Group	1
5.1.2	Normalization Regex Libraries Group on page 270	Lookup Normalization Group	1
5.1.2.1	Normalization Regex Library Node on page 270	Normalization Regex Libraries Group	0 or more
5.2	Vocabularies Group on page 286	Morph Analysis Group	1
5.2.1	Vocabulary Node on page 287	Vocabularies Group	1 or more
5.3	Categorization Regex Libraries Group on page 271	Morph Analysis Group	1
5.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more
5.4	Number Check Node on page 258	Morph Analysis Group	1

Hierarchy	Node Group	Container Group	Count
5.5	Default Categories Node on page 242	Morph Analysis Group	1
6	Pattern Recognition Group on page 264		1
6.1	Pattern Logic Node on page 259	Pattern Recognition Group	1
7	Scoring Node on page 275		1

Language Guess Definitions

A Language Guess definition guesses the language from which a string might originate. This might be used as a first step before applying other processing; for example, running definitions from the guessed language on the string. Logically, this type of definition should be created in a language rather than a locale.

At least one N-Gram Scheme Node on page 255 or Regex Library Node on page 266 must exist for a valid definition.

Input: a string

Example: `"28 Rue des Halles"`

Output: a confidence score, indicating how likely it is that the input belongs to the definition's language.

Hierarchy	Node Group	Container Group	Count
1	Language Guess Definition Head Node on page 222		1
2	Preprocessing Regex Library Group on page 267		1
2.1	Preprocessing Regex Library Node on page 267	Preprocessing Regex Library Group	0 or more
3	N-Gram Analysis Group on page 254		1
3.1	N-Gram Scheme Node on page 255	N-Gram Analysis Group	0 or more

Hierarchy	Node Group	Container Group	Count
4	Regex Search Group on page 272		1
4.1	Regex Library Node on page 266	Regex Search Group	0 or more
5	Scoring Node on page 275		1

Locale Guess Definitions

A locale guess definition guesses the locale from which a string might originate. It has applications similar to a language guess definition, but is more specific.

Input: a string

Example: `"17 gallons of anesthetic"`

Output: a confidence score, indicating how likely it is that the input belongs to the definition's locale.

Hierarchy	Node Group	Container Group	Count
1	Locale Guess Definition Head Node on page 222		1
2	Language Guessing Node on page 248		1
3	Regex Search Group on page 273		1
3.1	Regex Library Node on page 266	Regex Search Group	0 or more
4	Pattern Search Group on page 265		1
4.1	Chopping Group on page 243		1 (*)
4.1.1	Chop Table Node on page 243	Chopping Group	at most 1 (*)
4.2	Morph Analysis Group on page 249		1 (*)
4.2.1	Lookup Normalization Group on page 257	Morph Analysis Group	1 (*)

Hierarchy	Node Group	Container Group	Count
4.2.1.1	Uppercasing Node on page 241	Lookup Normalization Group	1 (*)
4.2.1.2	Normalization Regex Libraries Group on page 270	Lookup Normalization Group	1 (*)
4.2.1.2.1	Normalization Regex Library Node on page 270	Normalization Regex Libraries Group	0 or more (*)
4.2.2	Vocabularies Group on page 286	Morph Analysis Group	1 (*)
4.2.2.1	Vocabulary Node on page 287	Vocabularies Group	1 or more (*)
4.2.3	Categorization Regex Libraries Group on page 271	Morph Analysis Group	1 (*)
4.2.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more (*)
4.2.4	Number Check Node on page 258	Morph Analysis Group	1 (*)
4.2.5	Default Categories Node on page 242	Morph Analysis Group	1 (*)
4.3	Pattern Recognition Group on page 264		1 (*)
4.3.1	Pattern Logic Node on page 259	Pattern Recognition Group	1 or more (*)
5	Scoring Node on page 275		1

(*) up to maximum number of tokens in data type

Match Definitions

Match definitions generate match codes for input strings. The match codes represent the character content of the tokens in the input strings. If two input

strings generate the same match code, then those two strings are intended to represent the same entity.

The value of a match code is provided by the fact that two strings do not need to be exact character matches in order to generate the same match codes. For example, the string Washington DC should generate the same match code as Washington D.C. Groups of similar strings can be grouped by match code and standardized into a single, consistent representation.

The degree of similarity between strings that generates the same match code is set by a numeric sensitivity value. By changing the sensitivity value, you change the number of matches between match codes. In the range of 50-100, a low sensitivity value generates a simpler match code. Simpler match codes generate more matches, for strings that are less similar. Higher sensitivity values generate more complex match codes, for fewer matches, which reflects a higher degree of similarity between strings.

In a match definition, input strings are parsed into component substrings called tokens. The tokens are structured to remove unimportant content, then they are combined into a string. The token string is used to generate the match code.

The match definition is executed at a specified level of sensitivity. Internally, the match definition is configured by 10 sensitivity ranges. The input sensitivity maps into a range, and the range determines the content of the token string that generates the match code. For each sensitivity range, the match definition specifies how the tokens are assembled into the token string. At lower sensitivities, tokens can be truncated or removed to enable matches with less similarity.

The match code generation process is summarized as follows:

- receive input string
- parse tokens from string
- assemble token string based on sensitivity
- generate a match code

Using Customize, you can create or edit match definitions to fine-tune your match codes for your applications. One way to customize match definitions is to implement suggestion-based matching, which helps you detect and correct errors in spelling and typography. You can also use Customize to implement token-based matching, which helps you detect and correct tokens that are missing or out of position.

Input: a string, or preparsed input, and a sensitivity level

Example: `"John James McDonald"`

Output: one or more match codes (an encoded string of characters)

Hierarchy	Node Group	Container Group	Count
1	Match Definition Head Node on page 223		1

Hierarchy	Node Group	Container Group	Count
2	Preprocessing Regex Libraries Group on page 267		1
2.1	Preprocessing Regex Library Node on page 267	Preprocessing Regex Libraries Group	0 or more
3	Preprocessing Schemes Group on page 274		1
3.1	(Preprocessing) Transformation Scheme Node on page 275	Preprocessing Schemes Group	0 or more
4	Tokenization Node on page 279		1
5	Match Morph Analysis Token Group on page 250		1 or more (* and **)
5.1	Morph Analysis Group on page 249	Match Morph Analysis Token Group	1 (*)
5.1.1	Lookup Normalization Group on page 257	Morph Analysis Group	1 (*)
5.1.1.1	Uppercasing Node on page 241	Lookup Group	1 (*)
5.1.1.2	Normalization Regex Library Group on page 270	Lookup Group	1 (*)
5.1.1.2.1	Regex Library Node on page 266	Normalization Regex Library Group	0 or more (*)
5.1.2	Vocabularies Group on page 286	Lookup Group	1 (*)
5.1.2.1	Vocabulary Node on page 287	Vocabularies Group	0 or more (*)
5.1.3	Categorization Regex Libraries Group on page 271	Morph Analysis Group	1 (*)
5.1.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more (*)

Hierarchy	Node Group	Container Group	Count
5.1.4	Number Check Node on page 258	Morph Analysis Group	1 (*)
5.1.5	Default Categories Node on page 242	Morph Analysis Group	1 (*)
6	Token Combination Rules Group on page 282		1 (*)
6.1	Token Combination Rule Node on page 283	Token Combination Rule Group	0 or more (*)
7	Match Token Group on page 284		1 or more (* and **)
7.1	Suggestions Group on page 277	Match Token Group	1
7.1.1	Suggestions Node on page 277	Suggestions Group	0 or 1
7.2	Match Normalization Group on page 258	Match Token Group	1
7.2.1	Uppercasing Node on page 241	Match Normalization Group	1
7.2.2	Pre-Scheme Regex Libraries Group on page 270	Match Normalization Group	1
7.2.2.1	Regex Library Node on page 270	Pre-Scheme Regex Libraries Group	0 or more
7.3	Noise Word Removal Group on page 256	Match Token Group	1
7.3.1	Noise Word Vocabulary Node on page 257	Noise Word Removal Group	0 or more
7.4	Table-Based Transformations Group on page 284	Match Token Group	1

Hierarchy	Node Group	Container Group	Count
7.4.1	Transformation Scheme Node on page 285	Table-Based Transformations Group	0 or more
7.5	Post-Scheme Regex Libraries Group on page 270	Match Token Group	1
7.5.1	Regex Library Node on page 266	Post-Scheme Regex Libraries Group	0 or more
7.6	Phonetics Group on page 265	Match Token Group	1
7.6.1	Phonetics Library Node on page 266	Phonetics Group	0 or more
8	Matchcode Layout Node on page 248		1
9	Match Score Threshold Node on page 276		1
10	Match Encoding Node		1

(*) not displayed when there is no Tokenization definition

(**) up to maximum number of tokens in data type

Parse Definitions

A parse definition parses an input string. It attempts to understand which words or subphrases (if any) should be associated with each token of the associated data type.

Input: a string

Example: `"John James McDonald"`

Output: mapping between tokens and substrings

- Prefix => `""` (no part of the input string is associated with Prefix)
- First name => `"John"`
- Middle name => `"James"`
- Family name => `"McDonald"`
- Suffix => `""` (no part of the input string is associated with Suffix)

Hierarchy	Node Group	Container Group	Count
1	Parse Definition Head Node on page 223		1
2	Preprocessing Regex Library Group on page 267		1
2.1	Preprocessing Regex Library Node on page 267	Preprocessing Regex Library Group	0 or more
3	Chopping Group on page 243		1
3.1	Chop Table Node on page 243	Chopping Group	1
4	Morph Analysis Group on page 249		1
4.1	Lookup Normalization Group on page 257	Morph Analysis Group	1
4.1.1	Uppercasing Node on page 241	Lookup Normalization Group	1
4.1.2	Normalization Regex Libraries Group on page 270	Lookup Normalization Group	1
4.1.2.1	Normalization Regex Library Node on page 270	Normalization Regex Libraries Group	0 or more
4.2	Vocabularies Group on page 286	Morph Analysis Group	1
4.2.1	Vocabulary Node on page 287	Vocabularies Group	1 or more
4.3	Categorization Regex Libraries Group on page 271	Morph Analysis Group	1
4.3.1	Categorization Regex Library Node on page 272	Categorization Regex Libraries Group	0 or more
4.4	Number Check Node on page 258	Morph Analysis Group	1
4.5	Default Categories Node on page 242	Morph Analysis Group	1
5	Pattern Recognition Group on page 264		1

Hierarchy	Node Group	Container Group	Count
5.1	Pattern Logic Node on page 259	Pattern Recognition Group	1
6	Token Mappings Node on page 280		1

Pattern Analysis Definitions

The pattern analysis definition is used to transform strings that fit a particular pattern. It can be used to determine the structure of a string.

Input: a string

Example: `"apples and pears but not oranges"`

Output: transformed string

Example: `"X and X but not X"`

Hierarchy	Node Group	Container Group	Count
1	Pattern Analysis Definition Head Node on page 223		1
2	Regex Library Node on page 266		1

Regex Library Node

The Regex Library node checks the input string matches for certain patterns that can indicate its language or locale of origin.

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230
- Locale Guess Definitions on page 231
- Standardization Definitions on page 239

The properties available differ according to the type of definition in which is library is used.

Properties Within Standardization Definitions

Regex Library - Select a Regex Library to use. Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Output Within Standardization Definitions

Input - Test input string.

Output - String after regex processing.

Properties Within Language Guess and Locale Guess Definitions

Regex Library - Select a Regex Library to use. Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Likelihood - This is the likelihood of the input belonging to the definition's language (or locale) if this regex Library matches. "Never" means that, if the regex Library matches, it is impossible for the input to have come from the language (or locale).

Output Within Language Guess and Locale Guess Definitions

Input - Test input string.

Count - The number of matches for this node.

Score - The likelihood score for this node.

Standardization Definitions

A standardization definition formats the input into a desired "standard" format.

Input: a string or preparsed input.

Example: `"10 Main Street, Boston, Mass."`

Output: the standardized string

Example: `"10 Main St, Boston, MA"`

Hierarchy	Node Group	Container Group	Count
1	Standardization Definition Head Node on page 224		1
2	Tokenization Node on page 279		1
3	Standardization Token Group on page 276		1 or more (*)
3.1	(Lookup) Normalization Group on page 257	Standardization Token Group	1
3.1.1	Trimming Node on page 286	Normalization Group	

Hierarchy	Node Group	Container Group	Count
3.1.2	Uppercasing Node on page 241	Normalization Group	1
3.1.3	Pre-Scheme (Normalization) Regex Libraries Group on page 270	Normalization Group	1
3.1.3.1	Regex Library Node on page 266	Pre-Scheme Regex Libraries Group	0 or more
3.2	Transformation Scheme Group on page 284		1
3.2.1	Transformation Scheme Node on page 285	Transformation Schemes Group	0 or more
3.3	Post-Scheme (Normalization) Regex Libraries Group on page 269	Standardization Token Group	1
3.3.1	Regex Library Node on page 266	Post-Scheme Regex Libraries Group	0 or more
3.4	Casing Node on page 241		1
4	Concatenation Node on page 244		0 or 1See Note
5	Postprocessing Regex Libraries Group on page 269		0 or 1See Note
5.1	Postprocessing Regex Library Node on page 269	Postprocessing Regex Libraries Group	0 or 1See Note
6	Casing Node on page 241		1See Note

Note: This node is included only when tokenization is configured, as in the standardization definition named Business Title in the English locale.

Definition Groups and Nodes

Case Definitions

Default Casing Node

Specifies the casing rule to apply to the output. The rule does not apply to words that were matched in the Familiar Words group or the Familiar Patterns group. Words matched in these groups display as specified by the group.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

Output case - Select the type of casing operation this definition performs:

- Custom - no casing is applied; the definition relies on casing rules specified in other nodes.
- Lowercase, for example, "john smith"
- Uppercase, for example, "JOHN SMITH"
- Propercase, for example, "John Smith"

Use Unicode casing rules - Check the box to use Unicode casing rules instead of the legacy casing.

Output

Results - The selected case has been applied to the words in the Input column and appears in the Output column.

Uppercasing Node

The Uppercasing node represents the uppercase step within lookup normalization. By default, the input string(s) are automatically converted to uppercase before proceeding with the rest of the normalization process.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227

- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236
- Standardization Definitions on page 239

Properties

Automatic uppercasing - This check box is selected by default. Click the check box to turn off the automatic uppercase option.

Output

Message - If automatic uppercasing is on, the message is "Changes applied", because uppercase is applied. Otherwise, the message is, "No changes applied".

Result - Input(s) after the uppercase process. It is possible for the input and output to be identical even if "Changes applied", if the input was converted to uppercase to begin with.

Default Categories Node

The Default Categories Node categorizes any word in the input that has not been categorized by other means.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

A table lists the default categories and likelihoods that are assigned when no other categories or likelihoods have been be assigned. Click the icons on the right to add, edit, or delete the rows in the table.

Output

Message - If any word was assigned a default category, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - Output is cumulative with the outputs of the Vocabulary Node on page 286, Categorization Regex Library Node on page 272, and Number Check Node on page 258..

Chop Table

Chopping Group

Holds a number of Chop Table Nodes. These nodes are optional in some definitions and mandatory in others. If mandatory, this group contains a single (initially invalid) node when the definition is created.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Parse Definitions on page 236

Properties

None.

Output

Results - The output of the last node in the group. If the group is empty (that is, Chop Nodes are not mandatory), the output shows the default chopping algorithm.

Note: The default chopping algorithm chops based on white space only.

Chop Table Node

Divides the input string into substrings, most commonly individual words. The boundaries are determined by the rules in the Chop Table, which determine whether a given character indicates the start or end of a substring.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Language Guess Definitions on page 230

- Locale Guess Definitions on page 231
- Parse Definitions on page 236

Properties

Select a Chop Table to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the pane on the right under **Library files**, click **Show** files for all locales. QKB files from all locales are included.

Click **Open chop table** to edit the selected file.

Output

Results - A list of chopped substrings.

Concatenation Node

Concatenate substrings together into a single string.

- In Case definitions, the chopped and processed substrings are reassembled in the same order and with the same inter-word (between words) whitespace character as before chopping.
- In Standardization definitions, the outputs of each token group are concatenated. The separating characters between each token are specified in the node.
- Within Standardization definitions, there is no output when the tokenization definition specified has no solutions.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Standardization Definitions on page 239

Properties

- **Case Definitions** - None.
- **Standardization Definitions** - A table with three columns:
  - The first column does not have a name. It specifies to use separator strings before the selected tokens. Select the check box on the row for the separator to be used before the respective token.
  - **Token** - specifies the name of the token to which the separator can be applied.
  - **Separator** - specifies the characters in the separator string, which appears before the token's output string when the row is checked.

Click the up and down arrows to change the order of the tokens in the concatenated string.

To modify a separator value, click on the column in the table and modify the value.

Output

Message - The message "Changes applied" always appears, since concatenation always happens.

Result - The string after concatenation.

Extraction

Extraction Scheme Node

Extract substrings from the input and assign them to tokens using scheme (table) lookup.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225

Properties

Scheme

Select a scheme to use.

Click **Open** to edit the selected files or create a new file. The appropriate editor opens.

Token

Any word that is found in the scheme is assigned to this token.

Case Sensitive Scheme Lookup

If this check box is selected, the case of the input must match the case of the word in the scheme.

Output

Message - If any part of the input was found in the scheme, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result

- 1st column: the word
- 2nd column: the token that it was assigned to (blank if not assigned to any token)

Pattern-Based Extraction Group

The Pattern-Based Extraction Group contains the following nodes:

- Morph Analysis Group on page 249

- Pattern Recognition Group on page 264

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225

Properties

Use within definition: Select this check box to enable the group and its contained nodes. When this box is not selected, the display changes to show an empty node, and the Extraction Definition relies solely on the Extraction Scheme Nodes in the Table-Based Extraction Group. Click **Apply** to enable or disable the Pattern-Based Extraction Group.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If the group is not used, the output here is blank.

Table-Based Extraction Group

Holds a number of Extraction Scheme Nodes.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225

Properties

None.

Output

Message: If any node in the group matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result: The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Gender Token Group

The Gender Token Group represents the processing for a single token and contains the following Nodes:

- Categorization Regex Libraries Group on page 271
- Chopping Group on page 243
- Lookup Normalization Group on page 257
- Vocabularies Group on page 286

The name of the node is the name of the token.

The following **Definition** nodes use this definition type.

- Gender Analysis Definitions on page 227

Properties

Used within definition - If this check box is not selected, the token corresponding to this group is not processed and the assignments are inaccessible.

Assignments - Lists the categories and associated gender values that are assigned by the node. Click the icons on the right to add, edit, and delete table rows.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Identification Analysis

Familiar Phrase Scheme Node

The Familiar Phrase Scheme Node assigns an identity to the input string using a scheme.

The following **Definition** nodes use this definition type.

- Identification Analysis Definitions

Properties

Scheme

Select a Scheme to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, on the right, under **Library files**, click **Show files for all locales**. QKB files from all locales are included.

Click **Open scheme** to edit the selected file.

Scheme lookup is case sensitive

If the **Scheme lookup is case sensitive** check box is selected, the input case must match the case of the entry in the scheme in order to match.

The **Scheme lookup is based on toggle** enables you to select whether the scheme is applied to substrings within the input or the full phrase.

Identity

The identity that is assigned if the scheme matches. This assignment is not final, but contributes to the final score.

Output

Message - If any entry in the scheme matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - The assigned identity, if any.

Table-Based Identification Group

The Table-Based Identification Group holds a number of Familiar Phrase Scheme Nodes.

The following **Definition** nodes use this definition type.

- Identification Analysis Definitions on page 228

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the output is as if there was a single node that had no effect.

Language Guessing Node

The Language Guessing Node represents the processing carried out by a Language Guess Definition.

The following **Definition** nodes use this definition type.

- Locale Guess Definitions on page 231

Properties

Language definition - This is the language guess definition to use, if one is desired. Any language guess definition in the Locale (and ancestor Locales) can be selected, regardless of the data type. To change the name of the definition, click **Edit**.

Output

Confidence score - The usual output of the language guess definition. This score is factored into the final Locale Guess score.

Matchcode Layout Node

The Matchcode Layout Node combines tokens based on sensitivity to produce a token string. The token string generates a match code.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Each of ten available sensitivity ranges (100-95, 94-90...54-50) can generate a different token string, and hence a different match code. Select a sensitivity range to view or change the content of its output token string.

Token - Select a check box to include a token in the output token string.

Sliders - The sliders indicate the character position and length of the tokens in the output string. The maximum character position is 255. To change the display range of the slider bar, click **Tools** > **Options** > **Display**.

If a token has fewer characters than its specified length in the output string, then "placeholder" characters are appended to reach the specified length. If a token has more characters than the specified length, the token is truncated.

The maximum slider value specifies the length of the output token string.

Output

Message - If any regular expression in the Regex Library matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - A concatenated string of tokens.

Morph Analysis

Morph Analysis Group

The Morph Analysis Group is purely cosmetic. This node corresponds to the set of processing steps performed in the Data Quality morphological analysis. This group contains the following nodes:

- Lookup Normalization Group on page 257
- Vocabularies Group on page 286
- Categorization Regex Libraries Group on page 271
- Number Check Node on page 258 (except in Gender Analysis definitions)
- Default Categories Node on page 242 (except in Gender Analysis definitions)

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232

- Parse Definitions on page 236

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Match Morph Analysis Token Group

Represents the Morph Analysis processing for a single token.

Holds the following nodes:

- Morph Analysis Group on page 249

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Include token in definition output - If this box is unchecked, the token corresponding to this group is not processed.

Output

Same as the group that it holds.

N-Gram

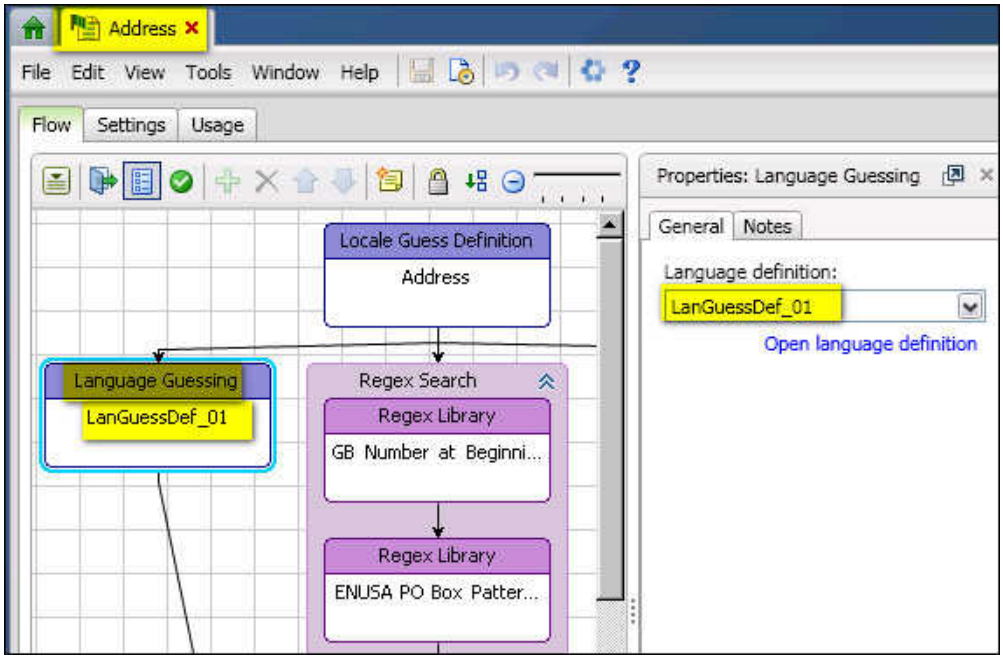
Using N-Grams in Your Language Guess Definition

Overview

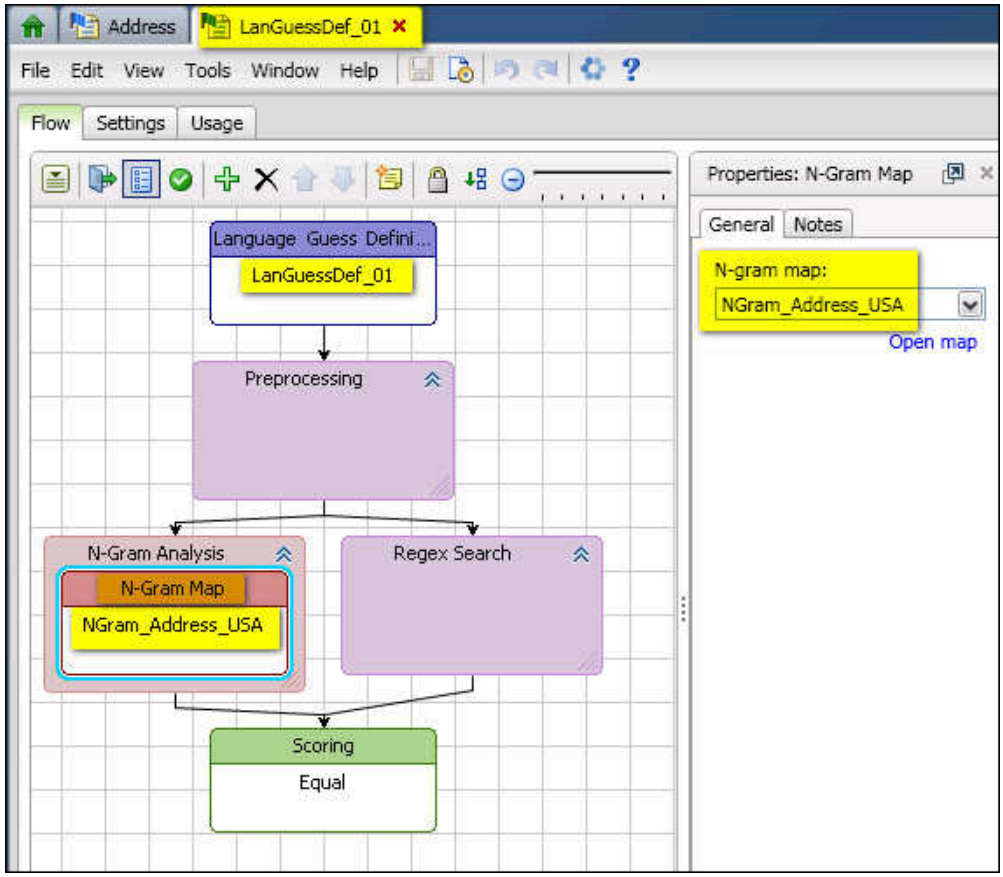
If you are using Locale Guess definitions to determine which QKB locale settings should be used to process data, you might want to use a Language Guess definition with n-gram analysis to improve the accuracy of your locale guess results.

A Language Guess definition is a QKB definition that analyzes an input string and computes a score indicating the confidence that the string is rendered in the language of the current QKB locale. If you embed a Language Guess definition in your Locale Guess definition, your Locale Guess definition uses the score returned by the Language Guess definition as a factor when determining the confidence that the input string belongs to the current QKB locale.

The next display shows a Locale Guess definition, Address, that contains a Language Guess definition, LangGuessDef_01.

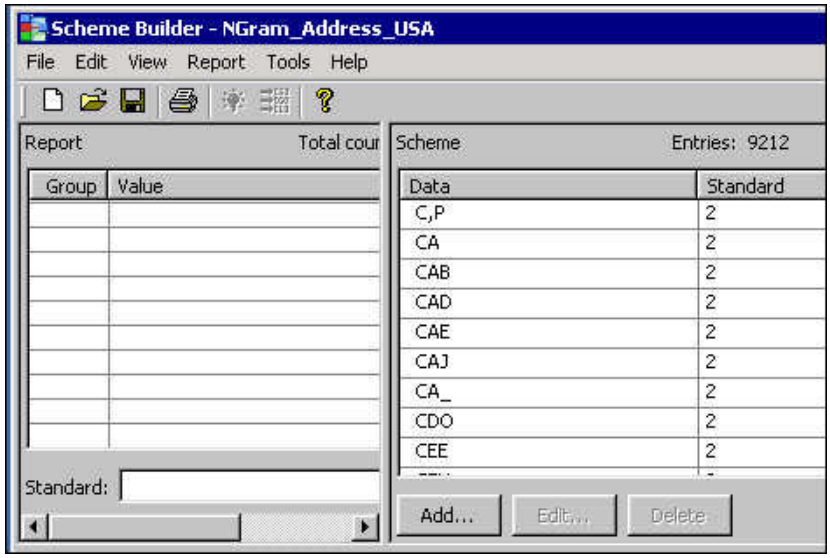


A Language Guess definition uses one or more n-gram schemes to analyze an input string. For example, if you click **Open** language definition in the right panel of the previous display, the editing window for the LangGuessDef_01 definition would open. The next display shows the editing window for the LangGuessDef_01 definition. This definition includes the n-gram scheme, NGram_Address_USA.



An n-gram scheme is a QKB scheme that contains patterns called n-grams that are derived from a body of text that is known to be in the language of the current QKB locale. The n-grams in the scheme are generated from a text file that is imported with Import N-Grams in the Scheme Builder.

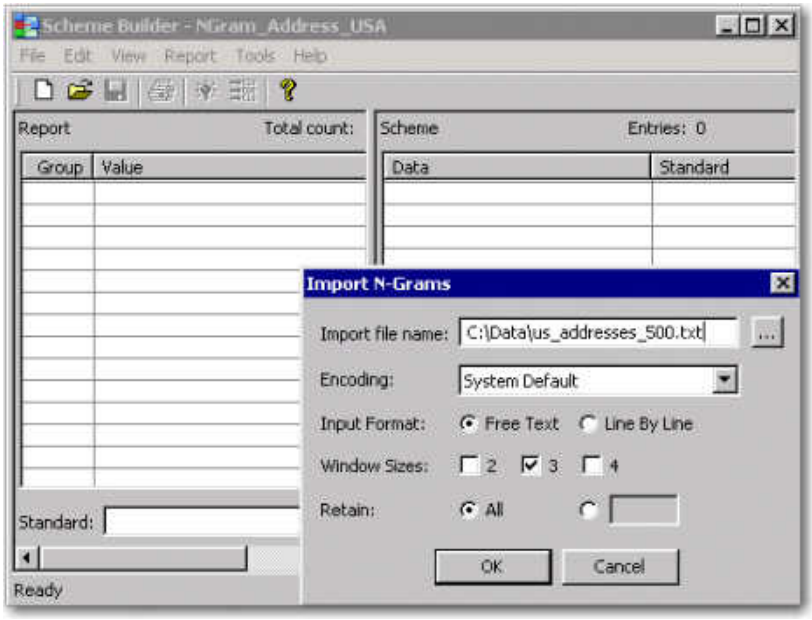
For example, if you were to click **Open** map in the right panel of the previous display, the Scheme Builder would open and display the contents of the n-gram scheme, NGram_Address_USA.



At run-time, the Language Guess definition generates a set of n-grams when it analyzes an input string. Each n-gram is then looked up in the n-gram scheme. If the n-gram exists in the scheme, the Language Guess definition adds points to the confidence score for the input string. This means that the confidence that the string belongs to the current QKB locale's language is increased.

Create an N-Gram Scheme

To create an n-gram scheme, use Import N-Grams to import an appropriate text file, as shown in the next display.



The text file must be in the language of the current locale. You get the best results if the file contains the same type of data that is analyzed by the Language Guess definition. In this example, assume that you want to create an n-gram scheme that are used to analyze address data in the English (United States) locale. The following steps describe one way to create such a scheme.

1. Select **Tools** > **Other QKB Editors** > **Scheme Builder** from the main menu in DataFlux Data Management Studio. The Scheme Builder is displayed. You are prompted to select a locale.
2. Select a locale that matches the data that you want to analyze. For this example, you would select the English (United States) locale.
3. Select **File** > **Import N-Grams** in the Scheme Builder. The **Import N-Grams** opens.
4. Select an appropriate text file in the Import file name field. For this example, you might select a file that contained address data from the English (United States) locale.
5. Specify the **Encoding of the data** or accept the default.

6 Specify the **Input Format for the data**. Select **Free Text** if each record in the file is not limited to one line. Select **Line by Line** if each record in the file is limited to one line.

7 Specify **Window Sizes**. A window size determines the length of an n-gram. For example, if you select a window size of 3, the Scheme Builder generates three-character n-grams, or 3-grams, from the text in the input file.

If you select more than one window size, the Scheme Builder generates n-grams for each selected window size. One window size can be used at a time in your Language Guess definition. Therefore, it might be useful to generate an n-gram scheme with multiple window sizes so that you can experiment with different window sizes in your Language Guess definition without needing to import the same text file multiple times.

Note: You might want to experiment with different window sizes before deciding which window size is best for your Language Guess definition. In principle, a larger window size (such as 3 or 4) might produce results with fewer false positives, a smaller window size (such as 2 or 3) produces more false positives but fewer false negatives.

8 In the **Retain** field, specify whether you want to retain all generated n-grams in your scheme or set a limit on the number to retain. If you want to retain all n-grams, select **All**. If you want to retain only a certain number of n-grams, enter the number in the edit field. The Scheme Builder retains only the most frequently occurring n-grams, up to the number of n-grams that you have specified. This option might be useful if you are importing a very large text file and you want to limit the size of your scheme.

9 Review your selections. When ready, click **OK**. The Scheme Builder scans the import text file and generate n-grams from the text that is stored in the file. The n-grams appear in the Data column in the Scheme Builder. The number of times the n-gram appeared in the import text file (the frequency of the n-gram) appears in the Standard column.

10 After importing, save the Scheme file.

Use an N-Gram Scheme

After an n-gram scheme has been saved, you can select the scheme in an n-gram scheme node in your Language Guess definition. In your Language Guess definition, be sure to select a window size that was used when you imported your n-grams. When analyzing strings, the Language Guess definition uses only n-grams of the specified window size, regardless of what window sizes were used to generate the n-grams contained in the n-gram scheme.

N-Gram Analysis Group

The N-Gram Analysis Group holds a number of N-Gram Scheme Nodes. See the Language Guess Definitions special constraints.

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230

Properties

N-Gram window size: The number of characters in an N-Gram.

Include word boundaries in n-gram lookup - Check this box to treat the beginning and end of a line as special characters.

Output

Unlike other groups, the N-Gram Analysis Group's output is not the output of the last node in the group. This is because the nodes in the group do not have individual outputs. Instead, all of the schemes are combined to produce a single output.

Results - Table with four columns:

- 1st column - each of the N-Grams in the input
- 2nd column - number of occurrences in the input
- 3rd column - index of first occurrence in the input
- 4th column - number of occurrences in the training data

Raw score - The score obtained from N-Gram analysis (between 0 and 1000).

Bias setting - The bias setting previously chosen.

Bias factor - The numerical weight that the bias setting translates to.

Bias score - The N-Gram score after applying the bias.

N-Gram Scheme Node

The N-Gram Scheme Node holds training data known to be in the language of interest. The data is stored in the form of N-Grams. Short segments of the input text produced by sliding a window of size N, move one character at a time, over the text. N refers to the number of characters in the segment (for example, 2, 3, or 4).

For example: The N-Grams of size 3 in the string "Hello Bob" are:

- `Hel`
- `ell`
- `llo`
- `lo[space]`
- `o[space]B`
- `[space]Bo`
- `Bob`

If bookends are used, there are two additional 3-Grams:

- [bookend]He
- ob[bookend]

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230

Properties

Select an N-Gram Scheme to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Options**. Click **Display** and select **Show files for all locales** under the **Library file selection drop-down lists** to view QKB files from all locales.

Be sure to select an N-Gram Scheme that contains N-Grams that have the same Window Size as the value of the **Window Size** property in the Language Guess Definition Head Node. For more information about **Window Sizes**, see Using N-Grams in your Language Guess Definition.

Output

Individual N-Gram Scheme Nodes have no output, because all the schemes are combined to produce the output.

Usage Note

The scheme for the N-Gram contained within Language Guess definition is designed to always perform a case-insensitive match and also to trim multiple white spaces to a single white space. The options contained within the scheme that are accessible via the **Options** button on the lower right hand side of the Scheme Builder are not used in the context of the N-Gram processing.

Noise Word Removal

Noise Word Removal Group

The Noise Word Removal Group contains a number of Noise Word Vocabulary Nodes.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, the output is displayed as if there was a single node, which had no effect.

Noise Word Vocabulary Node

The Noise Word Vocabulary Node removes words that are considered "noise" from the input. These are usually words that form a legitimate part of the input but are not important for matching purposes. For example, if matching addresses, the word denoting the type of street (road, lane, drive, trail, and so on) is usually considered unimportant.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Vocabulary - Select a vocabulary to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the pane on the right under **Library files**, click **Show files for all locales**. QKB files from all locales are included.

Click **Open vocabulary** to edit the selected file.

Sensitivity - Select the sensitivity range for which this node has an effect.

Output

Message - If any word in the Vocabulary matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - The string after noise words were removed, if any.

"All noise" message - Set to true if every word in the input string is found to be a noise word. If this occurs, no filtering occurs and the string is left intact.

Filtered words - A list of the noise words that were removed.

Normalization

Lookup Normalization Group

The Lookup Normalization Group is purely cosmetic. This group includes the steps that are performed to normalize the input for vocabulary lookup. This group contains the following nodes:

- Uppercasing Node on page 241
- Normalization Regex Libraries GroupNormalization Regex Libraries Group on page 270
- Vocabularies Group on page 286

The Lookup Normalization Group is used in the following definitions.

- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

None.

Output

Message: If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result: The output of the last node in the group.

Match Normalization Group

The Match Normalization Group includes the steps that are performed to normalize the input for noise word removal, table-based transformation, and phonetics. This group contains the following nodes:

- Pre-Scheme Regex Libraries Group
- Uppercasing Node on page 241

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Number Check Node

The Number Check Node represents the processing step that checks for numbers in the input. Numbers are assigned the special category NUM.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Locale Guess Definitions on page 231

- Match Definitions on page 232
- Parse Definitions on page 236

Properties

None.

Output

Message - If a number was found, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - This node assigns only the NUM category, and has an effect only if a number exists in the input. Output is cumulative with the outputs of and Categorization Regex Library Nodes (if applicable).

Pattern Analysis

Pattern Logic Node

The Pattern Logic Node uses the categories previously assigned to input words by morphological analysis to generate possible parse solutions. At the end of the morph analysis step, each word (or substring) in the input has been categorized. However, it is likely that a single word can have more than one possible category, and categories by themselves, even if unambiguous, do not give the final answer expected.

The Pattern Logic Node solves both of these problems by considering each word and its categories in the context provided by the full string, and with the aid of a grammar that supplies information about the allowed structure of inputs.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Parse Definitions on page 236

Within Locale Guess definitions, chopped words found in a given pattern logic node is not considered in subsequent pattern logic nodes. For example, suppose there are two pattern logic nodes and the chopped words are: this, is, an, example. If the first pattern logic node finds a pattern match for the word ‘this’ and ‘is’, then these two words are not searched for in the second pattern logic node. If the second pattern logic node finds a pattern match on ‘this’ and ‘is’, this pattern match within the second pattern logic node never happens. The words are not considered since the first pattern logic node already found a pattern match.

Properties

All Definition Types

The following properties are common to all definitions that include Pattern Logic Nodes.

- **Grammar** - Select a grammar to use. A grammar describes all the categories that words might have within a particular context, such as names or addresses. A grammar also describes the relationships between categories. Some categories are derived from a combination of others, and there might be many different ways to derive a particular category.

  By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the pane on the right, under **Library files**, click **Show files for all locales**. QKB files from all locales appear.

  Click **Open grammar** to edit the selected file.

  For example, in the United States, a phrase that represents a person's family name (the derived category) can commonly be seen in the following constructions:

  - single family-name word (for example, "John SMITH")
  - two family-name words (for example, "Helena BONHAM CARTER")
  - two family-name words with a hyphen (for example, "Mary JONES-SMITH")
  - Multiple family-name words, with hyphens or without
  - Single initial (for example, "John S.", last name abbreviated to protect a person's identity), and the list expands greatly as different languages, countries, cultures, and customs are taken into account

  In the simplest case, the structure of a grammar looks like a single tree. Some grammars can be made of multiple, unconnected trees.

- **Root Category** - This is the category at the root of the desired sub-tree of the category hierarchy, as defined by the grammar. This can be anything from the true root of a single-tree grammar, the root of a tree in a multiple-tree grammar, or simply any category, depending on the intent.

- **Optimization Parameters**

  - **Parse resource limit** - The maximum amount of computational resources that the parser should use. If the parser requires more resources for a particular input than are allowed by this parameter, the parser abandons the attempt to parse that string. Choosing a high resource limit allows the parser to use more resources when parsing an input string, meaning there is less chance that parses are abandoned for complex strings. However, a high resource limit also means that jobs can take longer to execute.
  - **Solution tree depth** - The maximum depth within a solution tree to which the parser searches when computing scores to compare solutions for a single input. Choosing a higher value allows the parser to search deeper into solution trees, meaning that rules at lower levels of a tree impact the score for that tree.
  - **Input length** - Optimizes processing for different lengths of input string. A different method of generating solution trees is used for short strings than

for long strings. If Auto is selected, the parser automatically determines which method to use. Auto is the recommended value for this parameter.

Note: Changing parse optimization parameters is an advanced activity. It is recommended that you not change the values of these parameters in QKB definitions provided by SAS. If you want to learn more about the effects of changing these parameters, you might want to attend a SAS QKB Customization training course. For information about training courses available from SAS, contact SAS Technical Support.

- **Identification Analysis and Locale Guess Definitions only**
  - **Sought category** - This is the category of interest within that sub-tree whose root is the root category.
  - **Stop searching all pattern logic nodes if a match occurs in this node** - If this check box is selected, processing of patterns stops after this node if a pattern match is found. If there is a pattern match on one or more of the chopped words in the node when this check box is selected, then no subsequent pattern logic nodes are used.
  - Search for patterns in Substrings and Full phrase.
    - **Substrings** - The pattern is matched against all input substrings
    - **Full phrase** - The pattern is matched only against the entire input string
- **Identification Analysis Definition only**
  - **Identity** - The identity to be assigned if the pattern matches.
  - **Pattern weight** - This number weights the final calculation in the pattern analysis.
- **Locale Guess Definition only**
  - **Likelihood** - The likelihood of the input being in the definition's locale, if the pattern matches. Setting to "Never" implies that an input that matches this pattern could never belong to the definition's locale.
- **Extraction Definition only**
  - **Category token mappings** - This table displays an ordered list of categories that are sought to match with input substrings, using the specified grammar. The best category match in the list causes the specified token to be applied to the input string. The table can therefore include multiple rows for the same category, each with a different token value. In the table that you can reposition rows vertically for easier ready by clicking the up and down arrows. To add and remove rows, use the plus and minus arrows.
  - **Pattern weight** - This number weights the final calculation in the pattern analysis.
  - **Maximum matches per pattern** - The maximum number of matches that are processed for this pattern.
- Search for patterns in Substrings and Full phrase.
  - **Substrings** - The pattern is matched against all input substrings
  - **Full phrase** - The pattern is matched only against the entire input string

Output

- **Message** - One of the following:
  - **Changes applied** - if the pattern was found and generated some solutions.
  - **Substring matched basic category** - if no solutions were found.
  - **Parse abandoned** - if computational resources were insufficient to complete the parsing operation (Parse Definition only).
- **Solution trees** - If any solutions were found for a test string, a number of solution trees appear in the output pane. Depending on the definition, they are organized in different ways.
  - each pattern up to and including the pattern of the currently selected node can generate its own set of solutions
  - solution display is cumulative (for example, solutions found by previous patterns are displayed along with solutions for the current node, if any)
  - for each pattern, the solutions are sorted from left to right in decreasing score order
  - the number of solutions shown can be configured under **Tools** > **Options** > **Display**
- **Solution tree structure**

```
--+ ROOT CATEGORY (Likelihood)
     |
     |--o string
     |
     |--o SUBCATEGORY1 SUBCATEGORY2...
```

The top of the tree shows the root category and the likelihood associated with it. The first bullet (blue) shows the string. The second bullet (yellow) shows the subcategories that form the root category when combined according to the rules of the grammar. Following are similar sub-trees for each of the subcategories. Then structure then recurses into each category until only basic (non-derived categories) are a part of the tree, or until the maximum solution tree depth specified by the node is reached.

If there are multiple solutions generated for a pattern, the one with the highest score is used to determine the final result.

- **Parse Definition** - The Parse Definition has only one pattern. Therefore, only one set of solutions appear, at most.
- **Identification Analysis Definition** - Each pattern that generated solutions (up to and including the currently selected Node) has a row of solution trees. For viewing convenience, some information about the pattern is shown to the left of its solution tree row.
- **Extraction Definition** - The Extraction Definition processes the input string in parallel across all Pattern Logic Nodes. If a node detects a word that matches its category, that word is extracted from the string. A second iteration then examines the new, shorter, substring. Iterations continue until all words have been extracted. In the Testing area, the left side of the output pane displays a substring tree. Each node in the tree represents an iteration. Iterations that expand display the words that were extracted in that

iteration. The same word or substring can appear more than once in the tree if multiple-token extraction is selected in the Extraction Definition Head Node.

Click a word to display its solution tree on the right side of the output pane. Included in the solution tree is the number of the Pattern Logic Node that generated the solution. Note that the test output for the Pattern Logic Summary Node combines the output from all Pattern Logic Nodes.

- **Locale Guess Definition** - The left output pane in the Test: area contains a tree representing the pattern logic node that was selected. The tree contains a node for each substring of the test value that was extracted by a pattern. Next to the substring is the number of solutions found. When you click on the top node of the tree, the right output pane contains a summary of the confidence values. The summary includes the confidence values for the selected pattern logic node and all pattern logic nodes that appear before the selected one in the Flow diagram. If you click on a substring, the solution trees for that substring are displayed in the right pane. The absence of nodes in the pattern logic tree means that no patterns were found.

Note: Locale Guess definitions created using older versions of DataFlux Data Management Studio or DataFlux dfPower Studio do not select a grammar in the Pattern Logic node. For these definitions a pattern can still be found due to a category match. When a pattern is found for these definitions, the number of solutions found is zero. Clicking on a substring displays the message "Substring matched basic category" in the right output pane.

Familiar Patterns Group

Holds a number of Familiar Pattern Regex Library Nodes. These nodes are always optional, so this group is empty when the definition is created.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Familiar Pattern Regex Library Node

This node enables you to identify exceptions for which the casing rules specified in the Casing node should not be applied. The chopped words that match any of the regular expressions from the specified Regex library is substituted according to the matched regular expression defined in the Regex library and the substituted value is excluded from having the casing rules applied.

The following **Definition** nodes use this definition type.

- Case Definition on page 224

Properties

Regex Library - Select a Regex Library to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the pane on the right, under **Library files**, click **Show file for all locales**. QKB files from all locales are included.

Click **Open regex library** to edit the selected file.

Do not apply if the value is transformed by a scheme - Select this check box to skip the Regex processing in this node, if the scheme in the previous group caused a transformation. If there is no scheme, this check box has no effect.

Output

Message - If any regular expression in the Regex Library matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Results

- 1st column: input word
- 2nd column: output word after transformation

Pattern Recognition Group

The Pattern Recognition Group holds a number of Pattern Logic Nodes. If the definition requires at least one Pattern Node, then this group contains a single (initially invalid) node when the definition is first created.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Parse Definitions on page 236

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Pattern Search Group

The Pattern Search Group contains the following nodes:

- Chopping Group on page 243
- Morph Analysis Group on page 249
- Pattern Recognition Group on page 264

The following **Definition** nodes use this definition type.

- Locale Guess Definitions on page 231

Properties

Use within definition - Select this check box to enable the group and its contained nodes. When not selected, the group node is displayed as empty and collapsed. In that case, the Locale Guess Definitions on page 231 relies solely on the Regex Library Nodes on page 266 in the Regex Search Group on page 272. Click **Apply** to enable or disable the Pattern Search Group.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Phonetics Library

Phonetics Group

The Phonetics Group contains a number of Phonetics Library Nodes.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - The output of the last node in the group. If there are no nodes, the output is displayed as if there was a single node that had no effect.

Phonetics Library Node

The Phonetics Library Node transforms an input using a Phonetics Library.

A Phonetics Library is a list of character combinations and their transformations, based on phonetic equivalents. For example, a doubled consonant ("LL") might be replaced by a single consonant ("L"). This is useful when dealing with words that are subject to spelling errors or variant forms, usually proper names. The underlying principle is that names that are written to sound alike, even though they might not be written the same way, are likely to refer to the same entity.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Phonetics library - Select a Phonetics Library to use.

Click **Open phonetics library** to edit the library in the Phonetics Editor.

Sensitivity range - Select a sensitivity range for which this node has an effect.

Output

Message - If any entry in the Phonetics Library matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - The string after phonetic transformation.

Regex Library

Regex Library Node

The Regex Library node checks the input string matches for certain patterns that might indicate its language or locale of origin.

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230
- Locale Guess Definitions on page 231
- Standardization Definitions on page 239

The properties available differ according to the type of definition in which is library is used.

Properties

Within Standardization Definitions

Regex Library - Select a Regex Library to use. Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Output Within Standardization Definitions

Input - Test input string.

Output - String after regex processing.

Within Language Guess and Locale Guess Definitions

Regex Library - Select a Regex Library to use. Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Likelihood - This is the likelihood of the input belonging to the definition's language (or locale) if this regex Library matches. "Never" means that, if the regex Library matches, it is impossible for the input to have come from the language (or locale).

Output Within Language Guess and Locale Guess Definitions

Input - Test input string.

Count - The number of matches for this node.

Score - The likelihood score for this node.

Preprocessing Regex Library Group

This group contains a number of Preprocessing Regex Library Nodes. In certain definitions, this group is empty until you add a node.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Parse Definitions on page 236

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the output appears as if there was a node that did not transform anything.

Preprocessing Regex Library Node

This node preprocesses the input string, removing or changing certain parts of the string using a set of regular expressions contained in a Regex Library. This is

generally done to remove irrelevant information or format the input to facilitate downstream processing.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Extraction Definitions on page 225
- Identification Analysis Definitions on page 228
- Language Guess Definitions on page 230
- Parse Definitions on page 236

Properties

Regex Library

Select a Regex Library to use.

By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the same pane on the right, under **Library files**, click **Show** files for all locales. QKB files from all locales are included.

Click **Open** regex library to edit the selected file.

Output

Message - If any regular expression in the Regex Library matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - The input string after processing.

Note: It is possible for the input and output to be identical even if "Changes applied".

Postprocessing Group

Holds a number of Postprocessing Regex Library Nodes. This group is empty until you add a node.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

None.

Output

Message - If any node in the group matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Postprocessing Regex Libraries Group

This group enables you to add one Regex Library node after the final Concatenation node in a Standardization definition. The library would enable you to perform regex operations on the input string as a whole, after it is standardized and concatenated. For example, you could always uppercase the first word in a string, or add a period onto the end of a sentence.

The following **Definition** nodes use this definition type.

- Standardization Definitions on page 239

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Postprocessing Regex Library Node

Apply a transformation to the string, removing or changing certain parts of the string using a set of regexes contained in a Regex Library. The postprocessing node is identical in function to a Preprocessing Regex Library Node.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224
- Pattern Analysis Definitions on page 238
- Standardization Definitions on page 239

Properties

Regex Library

Select a Regex Library to use.

Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Output

Input - If any regular expressions in the Regex Library match the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Output - The input string after processing.

Note: It is possible for the input and output to be identical even if "Changes applied", if the node happens to return output identical to the input.

Normalization Regex Libraries Group

The Normalization Regex Libraries Group contains a number of Normalization Regex Library Nodes.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236
- Standardization Definitions on page 239

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result -The output of the last node in the group. If there are no nodes, then the input of the group is the output. When you process a standardization result, the data is trimmed of whitespace character after you apply the Pre-Scheme Regex Libraries and before you apply the Transformation Schemes. This trimming occurs because regexes could possibly add spaces.

Normalization Regex Library Node

The Normalization Regex Library Node normalizes the input string using a set of regular expressions contained in a Regex Library. This is generally done to format the input prior to vocabulary lookup.

In most definitions the input to this node is a list of substrings or words. The normalization is applied to each substring independently.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228

- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236
- Standardization Definitions on page 239

Properties

Regex library

Select a Regex Library to use.

Click **Open** regex library to edit the selected file.

Output

If any regular expressions in the Regex Library match the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Input - The input before processing.

Normalized Form - The normalized form of the input. It is possible for an input and output to be identical even if "Changes applied".

Categorization Regex Libraries Group

The Categorization Regex Libraries Group includes a number of Categorization Regex Library Nodes.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Categorization Regex Library Node

The Categorization Regex Library Node searches a Regex Library to assign a category and likelihood. For more information about this topic, see Vocabulary Node.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

Regex library - Select a Regex Library to use.

Click **Open** regex library to edit the selected file in the Regex Library Editor.

Stop searching libraries if a matching pattern is found in this library - Check this box to stop the category search after the first match. Otherwise, the search continues through the library to find the best match.

Category - The category that is assigned to the word if a match is found in the Regex Library.

Likelihood - This is the likelihood of the category assignment.

Output

Message - If any regular expression in the Regex Library matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - See Vocabulary Node on page 287. Output is cumulative and includes categories assigned by vocabularies, if applicable.

Regex Search Group

The Regex Search Group holds a number of Regex Library Nodes. In certain definitions, this group node is empty until you add a node. See the Language Guess Definition on page 230 for special constraints.

The following **Definition** nodes use this definition type.

- Language Guess Definitions on page 230
- Locale Guess Definitions on page 231

Properties

None.

Output

Regex score - The total score for all Regex Libraries.

Language Guess Definitions Only

Bias setting - The bias setting previously chosen.

Bias factor - The numerical factor that the bias setting translates to.

Bias score - The regex score after weighting by the bias.

Scheme

Familiar Words Scheme Group

Holds a number of Familiar Word Scheme Nodes. These nodes are always optional, so this Group is empty when the Definition is created.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Familiar words - The output of the last node in the group. (If there are no nodes, then the output appears as if there was a node that did not transform anything.)

Familiar Words Scheme Node

This node enables you to identify exceptions for which the casing rules specified in the Casing node should not be applied. The chopped words that match familiar words contained within the specified scheme is transformed to their respective standard defined in the scheme and the transformed value is excluded from having the casing rules applied.

The following **Definition** nodes use this definition type.

- Case Definitions on page 224

Properties

Scheme - Select a scheme to use. By default, only files from the locale and its ancestors appear in the drop-down. If you do not see the desired library, click **Tools** > **Data Management Studio Options**. From the column on the left, select **QKB Definition Editor**. Then, in the pane on the right under **Library files**, click **Show files for all locales**. QKB files from all locales are included.

Click **Open** to edit the selected file or create a new file. The appropriate editor opens.

Case-sensitive scheme lookup Select this check box if the case of the input word must match the case of the word in the scheme in order to be transformed.

Output

Message - If any word was found in the scheme, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Results

- 1st column: true if the input word was found in the scheme
- 2nd column: input word
- 3rd column: transformed word (if the input was found in the scheme), or input word (if it was not)

Note: It is possible for the input and output word to be identical even if the word was found in the scheme.

Preprocessing Schemes Group

This Preprocessing Schemes Group holds a number of Transformation Scheme Nodes. This group is empty until you add a node.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output. If parsed input is used, this group is skipped, so there is no output.

(Preprocessing) Transformation Scheme Node

In a Preprocessing group, the Transformation Scheme Node performs preprocessing on the input string prior to parsing.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Scheme - Select a scheme to use. Click **Open scheme** to edit the selected scheme in the Scheme Editor.

Scheme lookup is case-sensitive - Select this check box if the case of the input string must match the case of the entry in the scheme in order for the input string to be transformed.

Scheme lookup is based on - Select **Substring** to apply the scheme to all substrings of the input. Select **Full Phrase** to apply the scheme to the input as a whole.

Output

Message - If the scheme matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - The input after transformation.

Note: The result can be the same as the input even though "Changes applied" appears in the message.

Scoring

Scoring Node

The Scoring Node represents the processing step that determines the final outcome.

The following **Definition** nodes use this definition type.

- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Language Guess Definitions on page 230
- Locale Guess Definitions on page 231

Properties

Language Guess Definition

Scoring bias - Select whether N-Gram or Regex searches have more weight in the final score.

Other Definitions

None.

Output

Gender Analysis Definition

Result - The final gender determination.

Scores - The scores for each possible gender.

Assignments - The possible gender assigned to each word.

Identification Analysis Definition

Result - The final gender determination.

Scores - The scores for each possible gender.

Language Guess Definition and Locale Guess Definition

Confidence score - A number ranging from 0 to 1000. The higher the score, the more likely it is the input string comes from the language or locale. Note that the score values are rounded, so the weighted and cumulative scores might not always add up to 1000.

Match Score Threshold Node

The Match Score Threshold Node is used for suggestion-based matching. It specifies a Minimum Score Threshold value, which is applied to the combined score for the entire match code as specified in the Matchcode Layout node. Any match codes that have scores below this value are discarded.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

For more information about this node type, see Suggestion-Based Matching on page 179 and Configuring the Match Score Threshold Node on page 189.

Standardization Token Group

The Standardization Token Group represents the processing for a single token. In the diagram of a Standardization Definition, the name of this group is the name of the token. The group contains the following nodes:

- Casing Node on page 241
- Lookup Normalization Group on page 257

- Transformation Scheme Group on page 285

The following **Definition** nodes use this definition type.

- Standardization Definitions on page 239

Properties

Used within definition - If this check box is not selected, the token corresponding to this group is not processed. Click **Apply** to change the selection.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Suggestion-Based Matching

Suggestions Group

The Suggestions Group enables you to add a suggestion-based matching mechanism to a token in a match definition. Suggestion-based matching is optional. The Suggestions Group is empty, unless you define a Suggestion Node. Suggestion-based matching helps detect data entry errors.

The Suggestions group contains the following nodes:

- Suggestions Node

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

None.

Output

If the group contains a node, the test output for the group is the same as the test output for the node. If there are no nodes, the test output shows the input to the group, unchanged, with default scores.

Suggestions Node

The Suggestions Node is used to configure the suggestion-based matching mechanism for the tokens in a match definition. The matching rules that you configure apply only to the token in which the Suggestion Node is defined. Each token in a match definition can have their own Suggestions Node.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

For more information about this node type, see Suggestion-Based Matching

Properties

Sample pronunciations scheme - Select a pronunciations scheme to use. The pronunciations scheme defines the pronunciation of some words so that suggestions can be generated with phonetically equivalent characters. A sample pronunciations scheme is required.

Click **Open scheme** to edit the selected scheme in the Scheme Builder.Known words scheme.

Known words scheme - Select a known words scheme to use. The known words scheme defines the words or terms that represent possible suggestions that can appear in the results.

Click **Open scheme** to edit the selected scheme.

Fuzziness - Select the Fuzziness level for the Suggestion Node, one of High, Medium, Low, or Custom. Each setting encompasses a set of values for many individual configuration parameters. The High level provides the highest accuracy and the Low level requires the least computation time. The Medium level is a reasonable compromise. These settings are not guaranteed to perform well, or even sensibly, for arbitrary data. For better results, it might be necessary to configure the individual parameters. To configure individual configuration parameters, select **Custom**. The **Fuzziness** editor opens. The **Fuzziness** editor enables you to view parameter settings for the High, Medium, and Low fuzziness levels, in addition to selecting customized values.

Sensitivity range - Select the upper and lower values of the sensitivity range in which suggestions are shown by this node.

Suggestions are case sensitive - If this check box is not selected, the incoming token is converted to uppercase at the input to the **Suggestions** node.

Do not use phonetics analysis if suggestions are found - If this check box is selected, the Phonetics Libraries farther down in the match definition flow does not take effect whenever there is at least one suggestion output by the Suggestions node (other than the input token itself). This is useful when converting an existing match definition that uses Phonetics Libraries because those phonetics rules tend to overly reduce the string.

Output

Message - If any node in the group matched the input, the message, "Changes applied", is displayed. Otherwise, the message is "No changes applied".

Rule menu - Use the menu to choose a rule (from among all the Token Combination Rules, plus the default rule) for which to display results. The result changes depending on which rule is selected.

Input - The string before entering this node.

Output - The string after the rules defined in the node were applied.

Raw score - The score when no rules were applied.

Rule-Weighted Score - The score after the rules were applied.

Tokenization

Tokenization Node

The Tokenization Node represents the tokenization processing carried out by a definition. This is necessary for definitions that treat individual tokens differently, to determine which substrings in an input string correspond to the tokens.

When the tokenization definition finds no solutions, there is no results for the Tokenization node or for any of the groups or nodes within the following token groups, except when the Tokenization node is used in a Match definition.

The following **Definition** nodes use this definition type.

- Gender Analysis Definitions on page 227
- Match Definitions on page 232
- Standardization Definitions on page 239

Input Properties

Within Gender and Standardization Definitions

Parse definition - This is the name of the parse definition. All available parse definitions that belong to the same data type are shown here.

Allow input that is already parsed - If this check box is selected, you can choose to allow previously parsed input as well as a single string. This parsed input can come from external sources (for example, columns of a database table) or from a previous run of the parse definition. No matter where the data comes from, the input fields must correspond to the tokens on the selected parse definition.

Within Match Definitions

Definition type - Enables you to select the type of definition that should be used. The choices are: (None), Parse and Extraction.

Definition - Enables you to select the definition that should be used. Click **Open** definition to view and edit the selected definition's Flow diagram.

Allow input that is already tokenized - If this check box is selected, you can choose to allow previously tokenized input as well as a single string. This tokenized input can come from external sources (for example, columns of a database table) or from a previous run of the definition. No matter where the data comes from, the input fields must correspond to the tokens on the selected definition.

Output

The output of the definition, if a single string was entered. If parsed input was used, the parsing step is skipped, so there is no output.

Token Configuration Node

The Token Configuration Node configures the sequential order of processing for tokens.

The following **Definition** nodes use this definition type.

- Gender Analysis Definitions on page 227

Properties

Table consists of one row for each token, and two columns:

- **Token** - name of the token
- **Stop if Matched** - whether to skip any tokens that have not yet been processed, if this token has assigned a gender

Select a row and click the up and down arrows to reposition the row.

Output

None.

Notes

Unlike other definitions, the gender analysis definition processes the tokens sequentially, in the order set by you. It also enables you to permit certain tokens' outputs to be considered more definitive than others. For example:

Input "`Mrs John Smith`", referring to the wife of John Smith. In this case, "`Mrs`" is the definitive word.

If all the tokens are processed in the natural order, the outputs for the tokens are:

- Name Prefix - "Mrs" (Female)
- Given Name - "John" (Male)
- Family Name - "Smith" (not used in the definition, since Family Name does not provide information about gender)

Since there is one female and one male assignment, the overall final gender assignment is unknown. However, processing stopped after the first token (Name Prefix) that has assigned a gender, the correct result is (Female). If Given Name was processed before Name Prefix, this does not work.

Token Mappings Node

The Token Mappings Node receives pairs of substrings and categories. The categories determine the assignment of substrings to tokens.

The output of the node is a list of tokens and token strings. Token strings are concatenations of the substrings that have been assigned to each token by category.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 225
- Parse Definitions on page 236

Properties

Parse Definition

Start in the **Solution-Based** tab, and then add details as needed in the **No Solution** tab.

Solution-Based tab

Use the **Solution-Based** tab to add, edit, reorder, or delete a list of categories for each token. Click a token to display its categories. Use the icons on the right to make changes. All input substrings with matching categories are added to the output string for the selected token.

For example, in the context of name-parsing, a category called Middle Name Word (MNW) could be mapped to a token called Middle Name. A category called Middle Name Initial (MNI) could also be mapped to the Middle Name token. All substrings that have been classified as MNW or MNI are added to the token string that appears with the Middle Name token.

Click **Use Separator** to remove the separator that is applied between substrings. To specify a different separator, click a token and click the edit icon. The default separator string is a single blank space.

No Solution tab

Use the **No Solution** tab to define default mappings for the output of this node. These mappings are used only when the conditions that are specified on the **Solutions-Based** tab are not found.

Note: If a Parse definition is embedded in another definition, the No Solution mappings are ignored unless the definition in which the Parse definition is embedded in a Match definition.

Under **Map categories when no solution is found**, click a token and click the plus icon to specify a maximum number of words for a category. You can specify different numbers for different categories.

Under **Categories to exclude**, specify the categories that are prevented from appearing in the node output. No categories are excluded by default. Typically, excluded categories are those such as COMMA and DASH that apply to punctuation marks.

Click **Default Token** to change the token that receives all input substrings that are not assigned to tokens. The default value **(None)** discards substrings that are not assigned to tokens.

Extraction Definition

Use token separator - If the **Use token separator** check box is selected, the displayed token separator is inserted between substrings. The default separator is a semi-colon (;). Click in the separator field to change the separator.

Default Token - In the **Default Token** field, select a token that receives unassigned substrings. Available selections include tokens, an **Additional Info** token, and a value of **(None)**. Select **(None)** to discard unassigned substrings. **Additional Info** is selected by default.

Output

Results - Table with two columns:

- 1st column - token name
- 2nd column - substring assigned to that token (in the Extraction Definition, could be several substrings concatenated together)

Token Combination Rules Group

Contains any Token Combination Rule Nodes that you add to your match definition. At the group level, you can control the generation of match codes for the default rule, and assign a weight number to the match code of the default rule.

The default rule is defined by the default combination of tokens, as defined in the match code, and no actions taken on those tokens.

Until you add your first Token Combination Rule node, the group node is empty and its properties are disabled.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Weight of default rule - Specifies a number from 0-100 that appears in the output along with the match code for the default rule. Assign weight values to the default rule and to the token combination rules to resolve clusters.

Still evaluate default rule if any token combination rules are defined - Select this check box to generate a match code for the default rule, based on the existence of token combination rules (which can be inactive.)

Still evaluate default rule even if any token combination rules are applied - Select this check box to generate a match code for the default rule if a token combination rule has been applied, as opposed to a rule simply being defined.

Output

Message - If any node in the group matches the input, the message is, "Changes applied". Otherwise, the message is, "No changes applied".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Token Combination Rule Node

Add one or more of these nodes to a Token Combination Rules Group in order to generate one additional match code for each node, for each input string. You have the option of generating another match code for the default rule.

When enabled, each Token Combination Rule is applied if the tokens in the input string meet specified conditions. The conditions are categories and likelihoods that are applied to the token nodes.

Token combination rules move or remove one or more tokens in the default rule. You move tokens to account for input strings with tokens that are out of position. You remove tokens to account for input strings that lack tokens. The subset combination of tokens is combined with a sensitivity range to generate a match code.

The default rule consists of the ordered list of tokens, without actions, that is listed by default under Actions in the node's **Properties**.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Use this rule - When this check box is selected, properties become accessible and the rule is used by the match definition.

Weight - Optionally specifies a number from 1-100 that appears in the output for the rule, for use in resolving clusters

Sensitivity range - Specifies a range of sensitivity values that is used to generate a match for token combination rule.

Conditions - Lists all of the conditions that must be met before the specified actions are taken. If no conditions are specified, then the rule is always applied.

You can click icons to add, edit, and delete conditions. In **Condition**, select a token, category, likelihood, and optional negation status. Negation represents a logical NOT, in which case the condition must not be met in order to take the specified actions.

Categories and likelihoods are made available for inclusion in conditions by adding Vocabularies to tokens in the match definition. For example, for the match definition EN Date (MDY), you could add a vocabulary that tags the Day token with the category COULD_BE_MONTH and a likelihood of Medium. You could then apply that category and likelihood to a condition.

A condition is true if the token is tagged with the specified category and with a likelihood that is equal to or greater than the specified likelihood.

Actions - Specifies how tokens are combined when conditions are met. For each token, specify None to remove that token from the recombined output string. Specify a different token to indicate a rearrangement. Specify the same token to keep that token in the same position.

Output

Message - If any part of the input was found in the scheme, "Changes applied". Otherwise, the message is, "No changes applied".

Result - A string of recombined tokens that is used to generate a match code.

Match Token Group

The Match Token Group represents the processing for a single token. The name of the node is the name of the token. The group node contains the following nodes:

- Match Normalization Group
- Noise Removal Group
- Phonetics Group
- Suggestions Group
- Transformation Scheme Group

The following **Definition** nodes use this definition type.

- Match Definitions on page 232

Properties

Include token in definition output - If this check box is not selected, then the token corresponding to this group is not processed.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group.

Transformations

Table-Based Transformations Group

This group contains a number of Transformation Scheme Nodes. To add a new node, right-click and select **Insert new Transformation Scheme**.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232
- Standardization Definitions on page 239

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group."

Result - The output of the last node in the group. If there are no nodes, the output is displayed as if there was a single node that had no effect.

Transformation Scheme Node

The Transformation Scheme node transforms the input by table lookup. For example, you can use a Transformation Scheme to standardize variant spellings to a common form.

The following **Definition** nodes use this definition type.

- Match Definitions on page 232
- Standardization Definitions on page 239

Properties

Match Definitions

Scheme - Select a scheme to use. Click **Open** to edit the selected scheme in the Scheme Editor.

Scheme lookup is case sensitive - Select this check box if the case of the input string must match the case of the entry in the scheme in order for the input string to be transformed.

Do not use phonetic analysis if transformation occurs - Select this check box to disable phonetic analysis after transformation. You might want to do disable phonetic analysis if you want the standard from your scheme to appear in the matchcode without any additional transformations applied during phonetic processing. You might choose this option if your scheme contains a comprehensive set of values and you are confident that you can achieve the desired level of matching without using phonetic reduction on the token to containing this node.

Sensitivity range - Select the upper and lower sensitivity range in which this scheme is applied.

Scheme lookup is based on - Select **Substring** to apply the scheme to all substrings of the input. Select **Full Phrase** to apply the scheme to the input as a whole.

Standardization Definitions

Scheme - Select a scheme to use. Click **Open** to edit the selected scheme in the Scheme Editor.

Output case - Use this control to select the case of the node output. Selecting (None) leaves the case unchanged.

Scheme lookup is case sensitive - Select this check box if the case of the input string must match the case of the entry in the scheme in order for the input string to be transformed.

Scheme lookup is based on - Select **Substring** to apply the scheme to all substrings of the input. Select **Full Phrase** to apply the scheme to the input as a whole.

Output

Message - If the scheme matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - The input after transformation.

Note: The result can be the same as the input even though "Changes applied" appears in the message, if the scheme transforms the input to itself.

Trimming Node

Properties

Trim whitespace - Select to trim white space from the input.

Output

Message - If any node in the group matches the input, "Changes were applied". Otherwise, the message is, "No changes were applied".

Result - The output of the last node in the group. If there are no nodes, then the input of the group is the output.

Vocabularies

Vocabularies Group

The Vocabularies Group contains a number of Vocabulary Nodes. All Definitions that use this group require at least one Vocabulary Node, so this group contains a single (initially invalid) node when the definition is created.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

None.

Output

Message - If any node in the group matches the input, the message is, "Changes applied in this group". Otherwise, the message is, "No changes applied in this group".

Result - The output of the last node in the group. Since the outputs of successively applied vocabulary nodes are cumulative, this is the combined output of all vocabularies applied.

Vocabulary Node

The Vocabulary Node performs a lookup in the vocabulary file to retrieve the categories and likelihoods for each input substring.

The following **Definition** nodes use this definition type.

- Extraction Definitions on page 227
- Gender Analysis Definitions on page 227
- Identification Analysis Definitions on page 228
- Locale Guess Definitions on page 231
- Match Definitions on page 232
- Parse Definitions on page 236

Properties

Vocabulary - Select a Vocabulary to use.

Click **Open Vocabulary** to open the selected vocabulary in the Vocabulary Editor.

Stop searching vocabularies if the word is found in this vocabulary - If this check box is selected and the input substring is found in the vocabulary of the current node, subsequent Vocabulary Nodes do not process that word. This enables you to decide whether two (or more) vocabularies, which contain the same word should contribute all the categories, or if only the first vocabulary categories should be considered.

Perform fuzzy lookups for all categories (if no selection) or specified categories - A word must match an entry in a vocabulary in order to be called a match. When fuzzy lookups are activated, words that are somewhat similar to those in the specified vocabulary matches . This option lets you select a specific category or categories of words from the vocabulary to match instead of using all categories.

Threshold - The threshold specifies the degree of similarity that must exist for the word to match. The higher the number, the more similarity must exist for the match. If you want an exact match, select 100. This turns off fuzzy lookups.

Output

Message - If any word in the Vocabulary matches the input, the message is "Changes applied". Otherwise, the message is, "No changes applied".

Result - A table with four columns:

- 1st column - the word
- 2nd column - the assigned category
- 3rd column - the likelihood associated with that category assignment
- 4th column - the word actually found, which might differ from the input word due to fuzzy lookup

Vocabulary outputs are cumulative, so a single node's output includes the outputs from previous vocabularies (if applicable).

Notes

A vocabulary is a table containing a list of words. For each word, one or more categories are assigned, and a likelihood is attached to each assignment.

Categories - A category indicates the function of the word in the context for which the vocabulary is intended.

For example, in the context of people's names, some possible categories might be:

- Prefix Word (PW), for example, "Mr"
- Given Name Word (GNW) , for example, "John"
- Family Name Word (FNW), for example, "Smith"

Likelihoods - A likelihood indicates the presumptive probability of that word belonging to that category.

For example, in the context of people's names, assuming the English language, you could say that, given no other information than general knowledge of English, "Judy" might show the following output.

- a very high likelihood of being a GNW
- a very low likelihood of being an FNW
- no possibility of being a PW

Relationship to Grammars - In many definitions, vocabularies (using Vocabulary Nodes) are used in conjunction with Grammars (through various Pattern Nodes). For this reason, the categories of a vocabulary that is intended for use in morphological analysis generally correspond to the categories defined in a related grammar.

See:

QKB Usage Notes - Vocabulary on page 214