Analyzing and Cleansing Data for a Master Index

Part No: 821–2658
March 2011

ORACLE

120126@25097

Contents

Analyzing and Cleansing Data for a Master Index

The topics listed here provide task, conceptual, and reference information for generating the Data Profiler and Data Cleanser from a Master Index Project and using those tools to analyze and cleanse legacy data.

Note that Java CAPS includes two versions of Oracle Java CAPS Master Index. Oracle Java CAPS Master Index (Repository) is installed in the Java CAPS repository and provides all the functionality of previous versions in the new Java CAPS environment. Oracle Java CAPS Master Index is a service-enabled version of the master index that is installed directly into NetBeans. It includes all of the features of Oracle Java CAPS Master Index (Repository) plus several new features, like data analysis, data cleansing, data loading, and an improved Data Manager GUI. Both products are components of the Java CAPS Master Data Management (MDM) Suite. This document relates to Oracle Java CAPS Master Index only.

What You Need to Know

These topics provide information you should to know before you start working with the Data Profiler and Data Cleanser.

What You Need to Do

These topics provide instructions on how to generate the Data Profiler and Data Cleanser, and how to configure and use the tools to analyze and transform legacy data.

More Information

These topics provide additional information you should know when working with the Data Profiler and Data Cleanser.

Related Topics

Several topics provide information and instructions for implementing and using a master index application. For a complete list of topics related to working with Oracle Java CAPS Master Index, see "Related Topics" in *Oracle Java CAPS Master Index User's Guide*.

Data Cleansing and Analysis Overview

Data analysis and cleansing are essential first steps towards managing the quality of data in a master index system. Performing these processes early in the project helps ensure the success of the project and can eliminate surprises down the road. Oracle Java CAPS Master Index provides the tools you need to analyze, profile, cleanse, and standardize legacy data before loading it into a master index database. The Data Profiler and Data Cleanser are generated from a master index application, and they use the object definition and configuration of the master index application to validate, transform, and standardize data.

The Data Profiler examines existing data and provides statistics and information about the data. It provides metrics on the quality of your data to help determine the risks and challenges of data integration. The Data Cleanser detects and corrects invalid or inaccurate records based on rules

you define to provide a clean and consistent data set. Together, these tools help ensure that the data you load into the master index database is standard across all records, that it does not contain invalid values, and that it is formatted correctly.

The following topics provide information about the Data Profiler and Data Cleanser as well as the overall process of analyzing and cleansing data.

About the Data Profiler

The Data Profiler analyzes the frequency of data values and patterns in your existing data based on predefined rules and rules you define. Rules are defined using a Rules Definition Language (RDL) in XML format. The RDL provides a flexible and extensible framework that makes defining rules an easy and straightforward process. Use the Data Profiler to perform an initial analysis of existing data to determine which fields contain invalid or default values, invalid formats, incorrect dates, and so on. This analysis spotlights values that need to be validated or modified during the cleansing process. For example, if you find you have several dates in the incorrect format, you can reformat the dates during cleansing. You might also find cases where the postal code extension is appended to the postal code, making the field value nine characters. You can truncate those fields to five characters during the cleansing phase, leaving the actual zip code.

The final data analysis is performed after the data is cleansed using the Data Cleanser. Use this analysis to verify the blocking definitions for the blocking query used in the master index match process. This analysis indicates whether the data blocks defined for the query are too wide or narrow in scope, which would result in an unreliable set of records being returned for a match search. This analysis can also show how reliably the field indicates a match between two records, which indicates how much relative weight should be given to each field in the match string.

The Data Profiler performs three types of analysis and outputs a report for each field being profiled. A *simple frequency analysis* provides a count of each value in the specified fields. You can specify a sort order and minimum frequency to display in the reports. A *constrained frequency analysis* provides a count of each value in the specified fields based on the validation rules you define. You can validate against a range of values, field length, or pattern to define the rules for the validation. A *pattern frequency analysis* provides a count of the patterns found in the specified fields. You can specify a sort order and the number of frequencies to display on the report.

About the Data Cleanser

The Data Cleanser validates and modifies data based on predefined rules and rules you define. Rules are defined using the same Rules Definition Language (RDL) as the Data Profiler, which provides a flexible framework for defining cleansing rules. The Data Cleanser not only validates and transforms data based on the rules you define, but it also parses, normalizes, and phonetically encodes data using the standardization configuration in the `mefa.xml` file of the master index project. You can define rules that validate or transform data during the cleansing process, and you can include conditional rules and operators. If you need to perform custom processing, you can define Java classes to extend the functionality of the rules.

The output of the Data Cleanser is two flat files; one file contains the records that passed all validations and was successfully transformed and standardized, and the other contains all records that failed validation or could not be transformed correctly. The bad data file also provides the reason each record failed so you can easily determine how to fix the data. This is an iterative process, and you might run the Data Cleanser several times to make sure all data is processed correctly. The final run of the Data Cleanser should produce only a good data file, with no records failing the process.

The final result of the Data Cleanser is a file that contains records that conform to the master index object definition and that no longer contain invalid or default values. The fields are formatted correctly and any fields that are defined for standardization in the master index application are standardized in the file. This is the file to load into the master index database using the Initial Bulk Match and Load tool (for more information, see *Loading the Initial Data Set for a Master Index*).

Data Cleansing and Profiling Process Overview

The general process of cleansing data begins with analysis, followed by cleansing, followed by additional analysis. By the time you are ready to load your existing data into the master index database, you want it to be of the best possible quality. To ensure this, you might need to repeat some of the profiling and cleansing steps to be sure all errant data is caught and fixed. The steps below provide a general overview to the analysis and cleansing process.

1. Make sure the master index application is configured and generated.
2. Extract the data to either a flat file.
3. Generate the Data Profiler and Data Cleanser from the master index application.
4. Determine the fields to use for the initial analysis.
5. Define the rules for the initial analysis and perform the initial analysis. This results in a set of reports for you to review to determine the cleansing rules to define.
6. Define the rules for the cleansing process and cleanse the data. This is an iterative process and you might find you need to define additional rules once the initial cleansing process is complete.

7. Analyze the blocking and matching fields to determine how to best configure the match process for the master index application.

After you configure the master index, you can load the data into the master index database using the Initial Bulk Match and Load tool (see *Loading the Initial Data Set for a Master Index* for more information).

Required Format for Flat Data Files

Both the Data Cleanser and the Data Profiler are designed to read data from a flat file. You can extract your data to a flat file using the extractor of your choice. The data needs to be placed in a flat file a format the Data Profiler and Data Cleanser can read. If your data is in a different format, you can define a custom data reader to read the flat file into the Data Profiler and Data Cleanser. The analysis tools can read a flat file in the following format without any additional configuration:

```
GID|SystemCode|LocalID|UpdateDate|UserID|ObjectFields
```

where:

- GID is a global ID computed by the data analysis. This field can be empty in the flat file.
- SystemCode is the processing code for the system from which the record originated.
- LocalID is the object's local ID in the given system.
- UpdateDate is the most recent update date for the record. This field can be empty.
- UserID is the logon ID of the user who last updated the record. This field can be empty.
- ObjectFields includes the objects and fields that are defined in `object.xml`. Be sure to include every field defined in the object structure in the order they are defined (include standardized, phonetic, and parsed fields). Child object types are delimited by a pound sign (#) and multiple child objects of one type are delimited by a dollar sign ($).

Below is an example of a valid input record based on the standard master index Person template, which includes alias, address, and phone objects. Note the empty fields after the first and last names for the phonetic and standardized data that will be inserted by the Data Cleanser. There are also empty fields after the street address for the parsed street address components that will also be inserted by the Data Cleanser.

```
28|ORACLE|00160419|11/14/1999 08:41:10|GSMYTHE|P|ELIZABETH|||ANN|WARREN|||MRS
|554-44-55555|08/18/1977|Y|F|M|W|13|BAP|ENG|STEVE|ANN|MARCH|GEORGE|CAHILL|SHEFFIELD
|CT|USA|E|Y||C4411444|CA|07/21/2018||ENG|USA#$BETH||CAHILL$LIZ|ANN|CAHILL#$H|1519
BOARDWALK|||||Unit 5|SHEFFIELD|CT|09876|1075|CAPE BURR|USA$W|12500 EAST RIVER ST.
||||||Suite 1310|CAPE BURR|CT|09877||CAPE BURR|USA#$CH|9895557848|$CB|9895551500|19
```

Generating the Data Profiler and Data Cleanser

In order to use the data analysis tools, you need to generate the Data Profiler and Data Cleanser from the master index application for which they will be used. These tools are based on the information you specified in the Master Index Wizard and changes you made to the configuration files. You can generate and extract the Data Profiler and Data Cleanser to a Windows or UNIX machine.

▼ To Generate the Data Profiler and Data Cleanser

1 **In the NetBeans Project window, right-click the main project of the master index application and then select Generate Cleanser Zip.**

The file is generated and downloaded to *NetBeans_Projects*/*Project_Name*/`cleanser-generated`.

2 **After the Data Cleanser is downloaded, right-click the main project again and then select Generate Profiler Zip.**

The file is generated and downloaded to *NetBeans_Projects*/*Project_Name*/`profiler-generated`.

3 **On your computer, navigate to** *NetBeans_Projects*/*Project_Name*/**`cleanser-generated` and extract the contents of `cleanser.zip`.**

4 **After the file are extracted, navigate to** *NetBeans_Projects*/*Project_Name*/**`profiler-generated` and extract the contents of `profiler.zip`.**

Next Steps Continue to "Configuring the Environment" on page 10.

Configuring the Environment

Before you start working with the Data Profiler and Data Cleanser, make sure the environment is configured to handle the requirements of both processes.

▼ To Configure the Environment

Before You Begin Complete the steps under "Generating the Data Profiler and Data Cleanser" on page 10.

1 **Make sure the Java installation you are using matches the Java version used by the NetBeans installation that generated the Data Profiler and Data Cleanser.**

Tip – To find the Java version for NetBeans, select Tools from the main menu and then select Java Platforms. To find the Java version you are using, type `Java -version` at a command prompt. If they do not match, either locate or install the correct version and change your PATH variable to point to the correct version.

2. **Create an environment variable named JAVA_HOME, and set the path to your Java installation; for example, `C:/Java/jdk1.6.0_10`.**

3. **Make sure the PATH variable contains the path to the `bin` directory in your Java installation; for example, `C:/Java/jdk1.6.0_10/bin`.**

Next Steps Continue to "Extracting the Legacy Data" on page 11.

Extracting the Legacy Data

Use the data extraction tool of you choice to extract the data you want to cleanse and analyze into a flat file. Review the information presented in "Required Format for Flat Data Files" on page 9 before extracting the data to be sure it is in the correct format.

Once you have the data file, continue "Determining the Fields to Analyze" on page 11.

Determining the Fields to Analyze

Once you extract the data from your source systems (described in "Extracting the Legacy Data" on page 11), you should determine what you want to achieve from the initial pass-through with the Data Profiler before you run the Data Cleanser. You do not need to profile the data prior to cleansing, however running a profile first can help you determine which fields need to be validated or transformed by the cleanser and how the those fields need to be processed. The Data Profiler identifies common values and patterns for these fields and gives you information about how to configure the Data Cleanser.

Here are some examples of the types of fields you might want to analyze prior to cleansing. After reviewing your data processing requirements, you will likely come up with additional types of analysis to perform.

- Fields that are likely to contain default values. Default values can include invalid values such as "999–99–9999" for Social Security Numbers or "John Doe" for first and last names.
- Fields that must be presented in a specific format. Required formats can include hyphenated social security numbers or phone numbers with parentheses and a hyphen (for example, (780)555–1515).

- Fields whose values are restricted to a valid value list. This can include fields such as gender, where there is generally one abbreviation for Female, one abbreviation for Male, one abbreviation for Unknown, and so on. Analyzing these fields helps identify incorrect abbreviations that cannot be read correctly by the master index.
- Date fields, especially dates of birth. In a master person index, the date of birth is generally used for blocking and matching, so you should verify whether you have any that are obviously incorrect (such as birth dates prior to 1900 or later than 2008).

Once you determine the fields to profile, continue to "Defining the Data Analysis Rules" on page 12

Defining the Data Analysis Rules

In this step, you define the rules for the frequency and pattern analyses to perform prior to cleansing the data. Use the results of the initial run of the Data Profiler to learn more about your data so you know which fields need to be transformed or validated during the cleansing process. The Data Profiler runs standard frequency analyses, pattern analyses, and constrained frequency analyses, which allow you to specify validation rules.

You can define the rules using an XML or text editor, or you can access and edit the file from the Files window in NetBeans.

▼ To Define Data Analysis Rules

Before You Begin Determine the fields to profile, as described in "Determining the Fields to Analyze" on page 11.

1 Navigate to the location of the Data Profiler.

By default, the Data Profiler is generated and extracted to *NetBeans_Projects/Project_Name*`/profiler-generated/profile`.

2 Open `sampleConfig.xml`.

Note – You can rename the configuration file and you can create multiple configuration files, each defining a different set of rules. Use `sampleConfig.xml` as a template for any files you create.

---

3 In the *profilerVariable* element, enter values for the attributes defined in "Data Profiler Processing Attributes" on page 18.

4 In the *varList* element, define all of the field variables to use in the profiling rules.

This step is optional. For more information, see "Data Profiler Global Variables" on page 19.

5 **Define the rules for each pattern or frequency analysis.**

For information about the available rules and the syntax to use, see "Data Profiler Rules Syntax" on page 18. You can create multiple configuration files to define different sets of rules.

Next Steps Continue to "Performing the Initial Data Analysis" on page 13.

Performing the Initial Data Analysis

After you customize the configuration file for the initial analysis run, you can run the Data Profiler against the staging database or a flat file. The initial analysis provides information about the state of your data and how you need to cleanse the data prior to loading. It can also provide values to use for matching, query, or SBR exclusion lists for the master index application or the initial bulk match and load process.

▼ To Perform the Initial Data Analysis

Before You Begin Before performing this step, make sure you have completed the following procedures:

- "Generating the Data Profiler and Data Cleanser" on page 10
- "Extracting the Legacy Data" on page 11
- "Defining the Data Analysis Rules" on page 12

1 **Navigate to *NetBeans_Projects/Project_Name*`/profiler-generated/profiler`.**

2 **If you changed the name of the configuration file from `sampleConfig.xml` or created new configuration files, do the following:**

   a. **Open `run.bat` (or `run.sh` on UNIX) for editing.**

   b. **Change "sampleConfig.xml" to the new name of the configuration file to use for the current run.**

3 **Do one of the following:**

   - **On Windows, navigate to the profiler home directory and then double-click `run.bat` or type `run.bat` on a command line.**
   - **On UNIX, navigate to the profiler home directory and then type `run.sh`.**

4 **If you created multiple configuration files to define the analysis rules, repeat the above steps until all configuration files are processed.**

Next Steps Continue to "Reviewing the Data Profiler Reports" on page 14.

Reviewing the Data Profiler Reports

When you run the Data Profiler, as described in "Performing the Initial Data Analysis" on page 13, the Data Profiler creates one report for each rule you defined and stores them in the location you specified in the configuration file. Review each report to help determine which fields need to be cleansed and how data needs to be validated or modified. When you profile prior to cleansing, you are looking for occurrences of invalid values or patterns, default values, missing values, and so on. This information translates into the rules you will write for the Data Cleanser.

Reports are written in CSV format so you can import them into a spreadsheet or reporting tool. Data patterns listed in the pattern frequency reports appear as regular expressions. See the Javadoc for `java.util.regex` for more information. The Data Profiler names each report based on the type of frequency performed, the order in which the rules appear in the configuration file, and the records on which it was performed. The naming syntax for the report file names is:

```
UD_FreqType_Order_Records.csv
```

where:

- *ID* identifies the type of frequency analysis performed. SF indicates simple frequency, CF indicates constrained frequency, and PF indicates pattern frequency.
- *FreqType* is a description of the type of frequency analysis performed; for example, PROFILE_PATTERN_FRQ.
- *Order* is the order in which the rule that generated the report appears in the configuration file. The profiler numbers each type of frequency report. For example, you might have constrained frequency reports 1, 2, and 3, and also have pattern frequency reports 1 and 2.
- *Records* is the range of records on which analysis was performed. If you did not specify a profile size, the ending number in the range is "0" (zero).

For example, `SF_PROFILE_SIMPLE_FRQ_3_1-100000.csv` is the third simple frequency analysis report defined in the configuration file and performed against the first the first 100,000 records. `CF_PROFILE_CONSTRAINED_FRQ_1_1-0.csv` is the first constrained frequency analysis report defined in the configuration file and performed against all records.

For examples of different types of frequency analysis reports, see "Data Profiler Report Samples" on page 41. When you finish analyzing the reports, continue to "Configuring the Data Cleansing Rules" on page 15.

Configuring the Data Cleansing Rules

After you review the Data Profiler reports, you should be able to determine which fields need to be validated or modified before the records can be loaded into the master index database. Before you begin this step, have a clear outline of the fields to validate, the actions to take if data fails or passes validation, and valid and invalid values to compare against. Also define any values that will be replaced, deleted, or truncated by the Data Cleanser, as well as any values that are simply rejected.

▼ To Configure the Data Cleansing Rules

Before You Begin Review the Data Analysis reports to determine the cleansing rules (see "Reviewing the Data Profiler Reports" on page 14 for more information).

1. **Navigate to the location of the Data Cleanser.**

   By default, the Data Cleanser is generated and extracted to *NetBeans_Projects*/*Project_Name*/`cleanser-generated/cleanser`.

2. **Open `sampleConfig.xml`.**

   Note – You can rename the configuration file and you can create multiple configuration files, each defining a different set of rules. Use `sampleConfig.xml` as a template for any files you create.

3. **In the *cleansingVariable* element, enter values for the attributes defined in "Data Cleanser Processing Attributes" on page 25.**

4. **In the *varList* element, define all of the variables to use in the cleansing rules.**

   For more information, see "Data Cleanser Global Variables" on page 26.

5. **Define the rules for the Data Cleanser.**

   For information about the available rules and the syntax to use, see "Data Cleanser Rules Syntax" on page 25.

6. **Save and close the file.**

7. **If you specified a path that does not exist for the output files, create the path you specified.**

Next Steps Continue to "Cleansing the Legacy Data" on page 16.

Cleansing the Legacy Data

After you customize the configuration file for the Data Cleanser, you can run the Data Cleanser against the staging database or a flat file. This step generates two files, one containing the records that passed all validation and was successfully cleansed and one containing records that failed validation along with an error message for each.

▼ To Cleanse the Data

Before You Begin Before performing this step, make sure you have completed the following procedures:

- "Generating the Data Profiler and Data Cleanser" on page 10
- "Extracting the Legacy Data" on page 11
- "Configuring the Data Cleansing Rules" on page 15

1. **Navigate to *NetBeans_Projects/Project_Name*/`cleanser-generated/cleanser`.**
2. **If you changed the name of the configuration file from `sampleConfig.xml` or created multiple configuration files, do the following:**
   a. **Open `run.bat` (or `run.sh` on UNIX) for editing.**
   b. **Change "<Rule_Config_File>" to the name of the configuration file to use for this run.**
3. **Do one of the following:**
   - **On Windows, navigate to the cleanser home directory and then double-click `run.bat` or type `run.bat` in the command line.**
   - **On UNIX, navigate to the cleanser home directory and then type `run.sh`.**
4. **Review the output files.**
5. **If there are any records in the bad data file, do one of the following:**
   - **If there are common errors for several records, define new cleansing rules in `sampleConfig.xml` to transform the bad data, delete the output files from the previous run, and then rerun the Data Cleanser against the staging database or flat file.**
   - **If there are unique errors for few records, fix the errant records in the bad data file, rename the bad data file, update the *DBConnection* and *startcounter* properties, and rerun the Data Cleanser against the updated file.**

Caution – Be sure to change the *DBConnection* attribute in the configuration file to point to the renamed file and change the *startcounter* value to the next record to be processed. For example, if the original run processed 100 good records, change the value to "101" to start processing the bad records. Any records cleansed from the fixed file are appended to the good data file.

6 **Repeat the previous steps until there are no records being written to the bad data file.**

Note – The final output to the good file can be loaded into the master index database using the Initial Bulk Match and Load tool (see *Loading the Initial Data Set for a Master Index*). The Data Cleanser automatically places the data in the correct format based on the `object.xml` file.

Next Steps Continue to "Performing Frequency Analyses on Cleansed Data" on page 17 to perform frequency analyses on the cleansed data.

Performing Frequency Analyses on Cleansed Data

After the data is cleansed (see "Cleansing the Legacy Data" on page 16), you can perform additional analyses against the data to help you determine how to configure query blocks and matching rules. Typically you would analyze the fields that are included in the block definitions for the query that is used for matching, and you could also analyze fields used for matching. The frequencies of these fields indicate how reliable they might be in the matching process and also indicate whether the blocking definitions are too broad or narrow to retrieve a reliable group of records for matching.

For this process, you might want to run frequency analysis against groups of records to find the frequencies for the unique values of the fields in the blocking definitions. After the data has been cleansed, you can run frequencies on standardized and phonetically encoded fields. The input for this process is the good data file to which the Data Cleanser wrote all the corrected and validated records.

Note – The blocking query is defined in the master index project in `query.xml`, and the match fields are defined in `mefa.xml`.

To perform frequency analyses on the cleansed data, repeat these procedures:

- "Defining the Data Analysis Rules" on page 12
- "Performing the Initial Data Analysis" on page 13
- "Reviewing the Data Profiler Reports" on page 14

Adjusting the Master Index Configuration

Based on the results of the final frequency analyses (see "Performing Frequency Analyses on Cleansed Data" on page 17), you might need to adjust the configuration of the master index application by adjusting the block fields if the frequencies are too high and by setting the relative match weights based on how unique each match field is. The results could also indicate that you might need to define exclusion files for the Initial Bulk Match and Load tool or filters for the SBR filter so certain values are not used for matching. For example, if there are a large number of SSN fields with the default value "000–00–0000", you can exclude that values from the blocking process, the match process, or the survivor calculation for the single best record.

Data Profiler Rules Syntax

Data Profiler rules are expressed using a Rules Definition Language (RDL) in an XML configuration file. Using a markup language allows you to easily define and change rules without having to code them. The configuration file is divided into two sections. The first section, *cleansingRules*defines rules for the Data Cleanser and the second section, *profilingRules*, defines rules for the Data Profiler.

The profiling section includes five subsections:

Data Profiler Processing Attributes

The following table lists and describes the attributes for the *profilerVariable* element in the configuration file. These attributes define the data source and path names for the Data Profiler as well as batch size. Below is a sample of the profiler attributes.

```
profilerVariable objectdefFilePath="../../src/Configuration"
DBconnection="../StagingDB" startFrom="50001" profileSize="50000"
reportFilePath=/Reports
```

Attribute	Description
objectdefFilePath	The path and filename to the `object.xml` file to use to profile the data.
DBconnection	The path to the staging database or the path and name of the flat file containing the data to be profiled. In this path, use forward slashes rather than back slashes.

Attribute	Description
startFrom	The record number at which the Data Profiler will start analyzing data. Use this attribute, along with the *profileSize* attribute, if you are running the process in batches.
profileSize	The number of records to process in one batch. This attribute is optional.
reportFilePath	The path where the Data Profiler reports will be stored. The profiler generates three different types of reports: Simple Frequency Reports, Constrained Frequency Reports, and Pattern Frequency Reports. One report is generated for each frequency rule you define. To see examples of these reports, see "Data Profiler Report Samples" on page 41

Data Profiler Global Variables

You can define global variables for the fields used in the Data Profiler rules so you do not need to use the qualified field name for each field. When defining variables, the qualified field name syntax is used. For more information about qualified field names, see "Master Index Field Notations" in *Oracle Java CAPS Master Index Configuration Reference* . Variables are defined in the *varList* element of the profiling rules section, and each variable is defined by a *var* element.

The *var* element has these attributes:

- *name* – The name of the variable.
- *default* – The qualified field name to the field defined by the *name* attribute.

Here is an example of a variable list for a master index application storing person data.

```
<varList>
   <var name="fname" default="Person.FirstName"/>
   <var name="lname" default="Person.LastName"/>
   <var name="ssn" default="Person.SSN"/>
   <var name="zip" default="Person.Address.PostCode"/>
   <var name="state" default="Person.Address.State"/>
</varList>
```

When you reference a variable in a rule, use the format ":[*variable_name*]" (where *variable_name* is the name you assigned to the variable). For example, to reference the FirstName field as defined above, it would look similar to this in the rules:

```
<field fieldName=":[fname]"/>
```

Simple Frequency Analysis Rules

A simple frequency analysis compares the values of the fields you specify and creates a report that lists each value for those fields along with the number of times each value occurs. You can

perform the analysis on a single field or multiple fields, and you can sort the resulting report by any of the fields in the report. Each simple frequency analysis rule is defined within *SimpleFrequencyAnalysis* tags that include the elements and attributes listed in the following table.

TABLE 1 Simple Frequency Analysis Rules

Element	Attribute	Description
fields		A list of fields to include in the frequency analysis.
field		One field definition in the list of fields.
	fieldName	The name of the field. If you defined a variable for the field, the syntax for this attribute is `fieldName=":[var_name]"`, where var_name is the name you gave the variable. If you did not define a variable, enter the qualified field name within double quotes. For example, `fieldName="Person.FirstName"`.
sortOrder		If defined, a field on which to sort in order of frequency. If multiple fields are defined, all of their frequencies will be sorted in this order.
	fieldName	The name of the field on which to sort. Use the syntax described for *fieldName* above.
	increasing	An indicator of whether to sort in increasing or decreasing order. Specify "true" to sort in increasing order, or specify "false" to sort in decreasing order.
threshold		If defined, a frequency threshold above which or below which field values will be listed on the report. If multiple fields are defined, the threshold setting applies the combination of the fields.
	value	The frequency threshold. This is a cutoff value to help limit the results of the report.
	more	An indicator of whether the threshold is an upper or lower threshold. Specify "true" to return field values with a frequency greater than or equal to the threshold. Specify "false" to return field values with a frequency less than the threshold.
topNpattern		If defined, the number of top frequencies to display. For example, you can restrict a report to just the top 10 frequencies of a field. If multiple fields are defined, this setting applies to the combination of fields.
	value	The number of top frequencies to display.

TABLE 1 Simple Frequency Analysis Rules *(Continued)*

Element	Attribute	Description
	showall	An indicator of whether to display more than the specified number of frequencies if there are multiple values tied at the lowest frequency to display. Specify "true" to show all values that are tied for the top frequencies. Specify "false" to only display the number of frequencies specified by the `value` element. If there is a tie, the displayed value is selected randomly.

EXAMPLE 1 Sample Simple Frequency Analysis Rules

The following sample defines an analysis of unique values for first and last names, and the resulting report displays only those values that occur 25 or more times and only those with the top 10 frequencies. If more than one value has the same frequency at the 10th frequency, all of those values are displayed. The results appear in increasing order of frequency.

```
<SimpleFrequencyAnalysis>
  <fields>
    <field fieldName=":[fname]"/>
    <field fieldName=":[lname]"
  </field>
  <sortOrder fieldName=":[fname]"/>
  <threshold value="25" more="true"/>
  <topNpatterns value="10" showall="true"/>
</SimpleFrequencyAnalysis>
```

The following sample defines an analysis of social security numbers and lists any duplicate values (that is, values that appear two or more times).

```
<SimpleFrequencyAnalysis>
  <fields>
    <field fieldName="Person.SSN"/>
  </field>
  <sortOrder fieldName="Person.SSN"/>
  <threshold value="2" more="true"/>
<SimpleFrequencyAnalysis>
```

Constrained Frequency Analysis Rules

A constrained frequency analysis compares the values of the fields you specify based on validation rules you define. It creates a report for each rule that lists each value for the fields along with the number of times each value or combination of values occurs. You can perform the analysis on a single field or multiple fields, and you can sort the resulting report by frequency in increasing or decreasing order. Constrained frequency analysis rules are defined within *ConstrainedFrequencyAnalysis* tags that include the elements and attributes listed in the following table.

TABLE 2 Constrained Frequency Analysis Rules

Element	Attribute	Description
fields		A list of fields to include in the frequency analysis.
field		One field definition in the list of fields.
	fieldName	The name of the field. If you defined a variable for the field, the syntax for this attribute is `fieldName=":[var_name]"`, where var_name is the name you gave the variable. If you did not define a variable, enter the qualified field name within double quotes. For example, `fieldName="Person.FirstName"`.
sortOrder		If defined, specifies a field on which to sort in order of frequency.
	fieldName	The name of the field to sort on. Use the syntax described for *fieldName* above.
	increasing	An indicator of whether to sort in increasing or decreasing frequency. Specify "true" to sort in increasing order, or specify "false" to sort in decreasing order.
threshold		If defined, specifies a frequency threshold above which or below which field values will be listed on the report.
	value	The frequency threshold. This is a cutoff value to help limit the results of the report.
	more	An indicator of whether the threshold is an upper or lower threshold. Specify "true" to return field values with a frequency greater than or equal to the threshold. Specify "false" to return field values with a frequency less than the threshold.
ruleList		A list of rules to apply to the frequency analysis.
rule		One rule definition to apply to the frequency analysis. You can define multiple rule definitions in a rule list. The following validation rules can be used in a constrained frequency analysis: "dateRange" on page 28, "patternMatch" on page 29, "dataLength" on page 27, and "range" on page 30. For descriptions and samples of these rules, see "Data Validation Rules" on page 27.

EXAMPLE 2 Sample Constrained Frequency Analysis

The example below performs a frequency analysis on the date of birth, but only for those dates that fall within a range too early to be valid. This an example of a profiling you might do prior to data cleansing in order to determine invalid values. This can also bring out invalid formats for the date, such as MM/DD/YY.

```
<ConstrainedFrequencyAnalysis>
  <fields>
    <field fieldName="Person.DOB"/>
  </fields>
  <ruleList>
    <rule>
      <dateRange fieldName="Person.DOB" min="01/01/0001" max="01/01/1900"/>
    </rule>
  </ruleList>
</ConstrainedFrequencyAnalysis>
```

Pattern Frequency Analysis Rules

A pattern frequency analysis compares the regular expression patterns found in the values for the specified field and performs a frequency analysis based on the patterns found. It creates a report for each field that lists each pattern along with the number of times each pattern occurs. You can perform the analysis on a single field or multiple fields, and you can sort the resulting report by frequency in increasing or decreasing order. Patterns are represented by regular expressions. For more information, see the Javadoc for `java.util.regex`.

Pattern frequency analysis rules are defined within *PatternFrequencyAnalysis* tags that include the elements and attributes listed in the following table.

TABLE 3 Pattern Frequency Analysis Rules

Element	Attribute	Description
topNpattern		If defined, the number of top frequencies to display. For example, you can restrict a report to just the top 10 frequencies of a field. If multiple fields are defined, this setting applies to the combination of fields.
	value	The number of top frequencies to display.
	showall	An indicator of whether to display more than the specified number of frequencies if there are multiple values tied at the lowest frequency to display. Specify "true" to show all values that are tied for the top frequencies. Specify "false" to only display the number of frequencies specified by the `value` element. If there is a tie, the displayed value is selected randomly.

TABLE 3 Pattern Frequency Analysis Rules *(Continued)*

Element	Attribute	Description
fields		A list of fields to include in the frequency analysis.
field		One field definition in the list of fields.
	fieldName	The name of the field. If you defined a variable for the field, the syntax for this attribute is `fieldName=":[var_name]"`, where var_name is the name you gave the variable. If you did not define a variable, enter the qualified field name within double quotes. For example, `fieldName="Person.FirstName"`.
sortOrder		If defined, specifies a field on which to sort in order of frequency.
	fieldName	The name of the field to sort on. Use the syntax described for *fieldName* above.
	increasing	An indicator of whether to sort in increasing or decreasing frequency. Specify "true" to sort in increasing order, or specify "false" to sort in decreasing order.
threshold		If defined, specifies a frequency threshold above which or below which patterns will be listed on the report.
	value	The frequency threshold. This is a cutoff value to help limit the results of the report.
	more	An indicator of whether the threshold is an upper or lower threshold. Specify "true" to return field values with a frequency greater than or equal to the threshold. Specify "false" to return field values with a frequency less than the threshold.

EXAMPLE 3 Sample Pattern Frequency Analysis

The examples below perform pattern frequency analyses on the date of birth and the social security number. This generates two reports, one for each field.

```
<PatternFrequencyAnalysis>
  <topNpatterns value='10' showall "true"/>
  <fields>
    <field fieldName="Person.SSN"/>
  </fields>
</PatternFrequencyAnalysis>

<PatternFrequencyAnalysis>
  <topNpatterns value='10' showall "false"/>
```

EXAMPLE 3 Sample Pattern Frequency Analysis *(Continued)*

```
  <fields>
    <field fieldName="Person.DOB"/>
  </fields>
</PatternFrequencyAnalysis>
```

Data Cleanser Rules Syntax

Data Cleanser rules are expressed using a Rules Definition Language (RDL) in an XML configuration file. Using a markup language allows you to easily define and change rules without having to code them, and you can also create custom Java classes to perform additional types of data validation or transformation. The first section of the configuration file, *cleansingRules*, defines rules for the Data Cleanser. The cleansing section includes three subsections. The first defines processing attributes, the second defines variables, and the third defines cleansing rules. The following topics provide information about each subsection.

Data Cleanser Processing Attributes

The following table lists and describes the attributes for the *cleansingVariable* element in the configuration file. These attributes define the data source and path names for the Data Cleanser as well as global validation rules. Below is a sample of the cleansing attributes.

```
cleansingVariable objectdefFilePath="../../src/Configuration" validateType="true"
validateNull="false" validateLength="true" DBconnection="../StagingDB"
goodFilePath="./Output/good.txt" badFilePath=./Output/bad.txt startCount="1"
standardizer="true"
```

Attribute	Description
objectdefFilePath	The path and filename for the `object.xml` file to use to cleanse the data.
validateType	An indicator of whether the cleanser should validate each field's data type against the type defined in `object.xml`. Specify **`true`** to validate field type; otherwise specify **`false`**. If you validate against type and the validation fails for any field in a record, the record is written to the bad file.

Attribute	Description
validateNull	An indicator of whether the cleanser should check for null values in each field that is configured to be required in `object.xml`. Specify **`true`** to check for null values; otherwise specify **`false`**. If you check for null values and any required field in a record is null, the record is written to the bad file.
validateLength	An indicator of whether the cleanser should validate each field's length against the length defined in `object.xml`. Specify **`true`** to validate field length; otherwise specify **`false`**. If you validate against length and the validation fails for any field in a record, the record is written to the bad file.
DBconnection	The path to the staging database or the path and name of the flat file containing the data to be profiled. Use forward slashes in this path rather than back slashes.
badDataFilePath	The path and name of the file that lists the records that are found to contain bad data during the cleansing process. This file includes an error message for each record describing the reason it was rejected. If you specify a path that does not exist, you need to create the path.
goodDataFilePath	The path and name of the file that lists the records that do not contain any bad data. These records can be processed through the Initial Bulk Match and Load tool into the master index database. If you specify a path that does not exist, you need to create the path.
startCounter	The starting number for the GID generator for the cleansed records. The GID is a unique value used by the Initial Bulk Match and Load tool, which takes the good data file created by the cleansing process as its input. Enter a non-negative long value. For the initial cleansing, set this to **`1`**.
standardizer	An indicator of whether the Data Cleanser should standardize the input data according to the standardization rules defined in the `mefa.xml` file in the master index project. Specify **`true`** to standardize the data. This populates the standardized values into the output file. Specify **`false`** to bypass standardization. If no value is specified or this property is missing, the default is true.

Data Cleanser Global Variables

You can define global variables for the fields used in the cleansing rules so you do not need to enter the qualified field name for each rule. When defining variables, the qualified field name syntax is used. For more information about qualified field names, see "Master Index Field Notations" in *Oracle Java CAPS Master Index Configuration Reference* . Variables are defined in the *varList* element, and each variable is defined by a *var* element.

The *var* element has these attributes:

- *name* – The name of the variable.
- *default* – The qualified field name to the field defined by the *name* attribute.

Here is an example of a variable list for a master index application storing person data.

```
<varList>
   <var name="fname" default="Person.FirstName"/>
   <var name="lname" default="Person.LastName"/>
   <var name="ssn" default="Person.SSN"/>
   <var name="zip" default="Person.Address.PostCode"/>
   <var name="state" default="Person.Address.State"/>
</varList>
```

When you reference a variable in a rule, use the format ":[*variable_name*]" (where *variable_name* is the name you assigned to the variable). For example, to reference the FirstName field as defined above, it would look similar to this in the rules:

```
<field fieldName=":[fname]"/>
```

Data Validation Rules

You can define rules to validate certain fields against specific values, a range of values, patterns, and length. You can define multiple rules, which are listed in the *ruleList* element. Each rule you define is contained within a *rule* element, and each *rule* element only defines one rule (though the rule can be complex and include conditional rules and operators).

The following validation rules are predefined to help you validate data during the cleansing process. These rules can be used in conjunction with transformation rules, conditional rules, and conditional operators.

dataLength

This rule validates the length of the specified field value against the length you specify. You can validate against a range "greater than or equal to" or "less than" the given length. Used alone for the Data Cleanser, this rule rejects records containing field values that fail the validation. You can also use this rule with conditional rules and operators to define more complex rules. When used in a constrained analysis for the Data Profiler, the frequency report is generated for all records for which the data length rule is true.

The syntax for `dataLength` is:

```
<dataLength fieldName="name" len="length" more="true/false"/>
```

The parameters for `dataLength` are:

- *fieldName* – The qualified field name or assigned variable for the field to validate.
- *len* – The length to use to validate the field value.
- *more* – An indicator of whether to consider lengths that are greater than or equal to the length or that are less than the length. Specify "true" if the field value should be greater than or equal to the specified length. Specify "false" if the field value should be less than the specified length.

EXAMPLE 4 Sample dataLength Validation Rule

This sample defines a rule to check for records containing first names that are 30 or more characters and containing last names that are shorter than two characters.

```
<rule>
  <dataLength fieldName="Person.FirstName" len="30" more="true"/>
  <datalength fieldName="Person.LastName" len="2" more="false"/>
</rule>
```

dateRange

This rule validates the value of the specified date field against the range you specify. Used alone in the Data Cleanser, this rule rejects records that contain data that fails validation. You can also use this rule with conditional rules and operators to create more complex rules. When this rule is used with the Data Profiler, the frequency report includes all records that match the date range specified. Note that dates with a 2–digit year instead of a 4–digit year will have "00" appended to the front. For example, if the invalid year "54" is found in the data, it is considered to be "0054" and is converted to that value in the Data Cleanser output file. The Data Profiler does not check for valid day and month values, and they appear in frequency reports for your analysis. Days and months are validated in the Data Cleanser.

Enter the date arguments for this rule in the format specified for the master index application (defined in `object.xml`). The syntax for `dateRange` is:

```
<dateRange fieldName="name" min="minimum_date" max="maximum_date"/>
```

The parameters for `dateRange` are:

- *fieldName* – The qualified field name or assigned variable for the field to validate.
- *min* – The earliest date to include in the valid date range.
- *max* – The latest date to include in the valid date range.

EXAMPLE 5 Sample dateRange Rule

This sample defines a rule that validates the date field against a range beginning 01/01/1900 and ending 06/01/2008. When used in the Data Cleanser, if a date of birth does not fall within that range, the record will be rejected and written to the bad data file. When used with the Data Profiler, any dates of birth that fall within the range are included in the frequency report.

EXAMPLE 5 Sample dateRange Rule *(Continued)*

```
<rule>
  <dateRange fieldName="Person.DOB" min="01/01/1900" max="06/01/2008"/>
</rule>
```

matchFromFile

This rule validates the value of the specified field against a list of values that you supply in a delimited file known as a valid value list. If a field value or part of a field value does not match an entry in the file, the record is rejected and written to the bad data file. You can also define a list of bad data in the file, known as an exclusion list, and then use this rule with conditional rules to reject records containing the bad data. If you include an empty entry in this list to indicate a null value, include the empty field in the middle of the list and not at the beginning or end. The syntax for `matchFromFile` is:

```
<matchFromFile fieldName="name" filePath="path" delimiter="delimeter" exact="true/false"/>
```

The parameters for `matchFromFile` are:

- *fieldName* – The qualified field name or assigned variable for the field to validate.
- *path* – The path and filename of the file containing the good or bad field values.
- *delimiter* – The delimiter used to separate the values in the value list file.
- *exact* – An indicator of whether the full field value must exactly match one of the entries in the valid value list in order to be considered valid. Specify "true" if the full field value must match exactly. Specify "false" if any part of the field value can match one of the entries in the valid value list.

EXAMPLE 6 Sample matchFromFile Rule

This sample defines a rule that validates State fields against a list of valid state abbreviations contained in the file `State.txt`. The field value must exactly match an entry in the list or the record will be rejected.

```
<rule>
  <matchFromFile fieldName="Person.Address.State"
   filePath="C:\\Profiling\\ValidValues\\StateCodes.txt"
   delimiter=";" exact="true"/>
</rule>
```

In this case, the valid value list would look similar to this:

```
AL;AK;AS;AZ;AR;CA;CO;CT;DE;DC;...
```

patternMatch

This rule validates the pattern of the specified field value against a regular expression. See the Javadoc for `java.util.regex` for more information about using regular expressions. Used

alone in the Data Cleanser, this rule can be defined to either reject records containing field values do not match the pattern or to reject records containing fields values that do match the pattern. You can also use this rule with conditional rules and operators to define more complex rules. When this rule is used with the Data Profiler, the frequency report includes a frequency for any patterns that either match or do not match the specified pattern, depending on the value of the *found* parameter.

The syntax for `patternMatch` is:

```
<patternMatch fieldName="name" matchPattern="pattern" found="true/false"/>
```

The parameters for `patternMatch` are:

- *field_name* – The qualified field name or assigned variable for the field to validate.
- *matchPattern* – A regular expression that defines the pattern to match against.
- *found* – A boolean indicator of whether to validate for field values that do match the pattern or that do not match the pattern. When this parameter is set to **`true`** in a cleansing rule, the Data Cleanser rejects records that do not match the pattern. For the Data Profiler, the generated report only includes field values that match the pattern. When this parameter is set to **`false`** in a cleansing rule, the Data Cleanser rejects records that match the pattern. For the Data Profiler, the generated report only includes fields that do not match the pattern.

EXAMPLE 7 Sample patternMatch Rule

This sample validates the social security number field to ensure it is in the format "NNN-NN-NNNN". Records containing a social security number in any other format are rejected or are not included in the frequency report.

```
<rule>
  <patternMatch fieldName="Person.SSN" matchPattern="[0-9]{3}-[0-9]{2}-[0-9]{4}"
   found="true"/>
</rule>
```

range

This rule validates the value of the specified field against a numeric range. The range is specified in integers. When used alone with the Data Cleanser, this rule rejects records containing field values that fall outside of the given range. You can also use this rule with conditional rules and operators to define more complex rules. When this rule is used with the Data Profiler, the frequency report includes a frequency of field values that fall within the range.

The syntax for `range` is:

```
<range fieldName="name" min="minimum" max="maximum"/>
```

The parameters for `range` are:

- *fieldName* – The qualified field name or assigned variable for the field to validate.
- *min* – An integer indicating the minimum value in the numeric range.
- *max* – An integer indicating the maximum value in the numeric range.

EXAMPLE 8 Sample range Rule

This sample validates United States postal codes to be sure they fall within a valid range. Records containing postal codes less than 06000 or greater than 99950 will be rejected or will not be included in a frequency report. Note that the PostalCode field needs to be a numeric data type in order for this to be a valid rule.

```
<rule>
  <range fieldName="Person.Age" min="06000" max="99950"/>
</rule>
```

reject

This rule rejects the specified field as bad data and writes the record to the bad data file. This rule is designed to be used in conditional statements as one action to take if a field value fails its validation. The syntax for `reject` is:

```
<reject fieldName="name"/>
```

The parameter for `reject` is:

- *fieldName* – The qualified field name or assigned variable for the field to reject.

EXAMPLE 9 Sample reject Rule

This sample checks whether the SSN field is null. If the SSN field is null, the record is rejected and written to the bad data file. If the field is not null, the record is returned as “good” data.

```
<rule>
  <if>
    <condition>
      <isnull fieldName="Person.SSN"/>
    </condition>
    <then>
      <reject fieldName="Person.SSN"/>
    </then>
    <else>
      <return fieldName="Person.SSN"/>
    </else>
  </if>
</rule>
```

return

This rule returns the specified field as good data . This rule is designed to be used in conditional statements as the action to take if a field value passes its validation. The syntax for `return` is:

```
<return fieldName="name"/>
```

The parameter for `return` is:

- *fieldName* – The qualified field name or assigned variable for the field to return as good data.

EXAMPLE 10 Sample return Rule

For a sample of the `return` rule, see the sample for "reject" on page 31.

validateDBField

This rule validates the specified fields against the length defined for those fields in `object.xml`. You can specify whether to reject records that exceed the defined length or to truncate the field value to the defined length. The syntax for `validateDBField` is:

```
<validateDBField>
  <field fieldName="name" action="reject/truncate"/>
  <field fieldName="name" action="reject/truncate"/>
  ...
</validateDBField>
```

The parameters for `validateDBField` are a list of fields and the action to take for each field. The field elements take the following parameters:

- *fieldName* – The qualified field name or assigned variable for the field to validate.
- *action* – The action to take if the field value exceeds the allowed length. Specify "reject" to reject the record as bad data, or specify "truncate" to truncate the field value to the length defined in `object.xml`.

EXAMPLE 11 Sample validateDBField Rule

The following sample checks the length of the social security number and last name in each record. If the social security number is too long, the record is rejected and written to the bad data file. If the last name is too long, it is truncated.

```
<rule>
  <validateDBField>
    <field fieldName="Person.SSN" action="reject"/>
    <field fieldName="Person.LastName" action="truncate"/>
  </validateDBFields>
</rule>
```

Data Transformation Rules

You can define rules that modify the values or the patterns of certain fields. Use these rules alone to modify a field across all records, or use them within conditional rules to modify the values or patterns only in fields that meet the criteria you define. You can define multiple rules, which are listed in a *ruleList* element. Each rule you define is contained within a *rule* element, and each *rule* element only defines one rule (though one rule can be complex and include conditional rules and operators).

The following transformation rules are predefined to help you cleanse data. These rules can be used in conjunction with conditional rules and operators, as well as with validation rules.

assign

This rule assigns a new value to the specified field. The syntax for `assign` is:

```
<assign fieldName="name" value"new_value"/>
```

The parameters for `assign` are:

- *fieldName* – The qualified field name or assigned variable for the field to assign the value to.
- *value* – The new value to replace the existing value in the field.

EXAMPLE 12 Sample assign Rule

The following rule checks whether the value of the gender field equals 2, FEM, or FML. If any of those values are found, they are changed to "F". This standardizes the values that indicate a gender of female to the correct processing code for female. If the three listed values are not found in the gender field, the record is returned as good data.

```
<rule>
  <if>
    <condition>
      <or>
        <equals value1="Person.Gender" value2="2" exact="true"/>
        <equals value1="Person.Gender" value2="FEM" exact="true"/>
        <equals value1="Person.Gender" value2="FML" exact="true"/>
      </or>
    </condition>
    <then>
      <assign fieldName="Person.Gender" value="F"/>
    </then>
    <else>
      <return fieldName="Person.Gender"/>
    </else>
```

EXAMPLE 12 Sample assign Rule *(Continued)*

```
  </if>
</rule>
```

patternReplace

This rule checks the value of the specified field against a pattern. If the patterns match, this rule replaces the existing pattern with a new pattern. Use regular expressions to define the patterns (see the Javadoc for `java.util.regex` for more information). The syntax for `patternReplace` is:

```
<patternReplace fieldName="name" matchPattern"old_pattern" replace="new_pattern"/>
```

The parameters for `patternReplace` are:

- *fieldName* – The qualified field name or assigned variable for the field to check.
- *matchPattern* – The pattern to check for in the specified field.
- *replace* – The pattern that will replace the existing pattern in the specified field.

EXAMPLE 13 Sample patternReplace Rule

The following sample searches the SSN field for 9–digit values without hyphens. If it finds such values, it inserts hyphens at the appropriate places.

```
<rule>
  <patternReplace fieldName="Person.SSN" matchPattern="[0-9]{9}"
   replace="[0-9]{3}-[0-9]{2}-[0-9]{4}"/>
</rule>
```

replace

This rule checks the value of the specified field against a given string. If the field value or part of the field value matches the string you specify, this rule replaces the existing string with a new string. The syntax for `replace` is:

```
<replace fieldName="name" matchPattern="old_string" replace="new_string"/>
```

The parameters for `replace` are:

- *fieldName* – The qualified field name or assigned variable for the field to check.
- *matchPattern* – The string to check for in the specified field.
- *replace* – The string that will replace the existing string in the specified field.

EXAMPLE 14 Sample replace Rule

The following sample looks for the values 1, MALE, or MAL in the gender field. If those values are found, it replaces them with “M” in order to standardize them to the processing code for

EXAMPLE 14 Sample replace Rule *(Continued)*

male. Note that the replacement for MALE is defined before the replacement for MAL. Since this rule looks for and replaces full field values or partial field values, performing the process on MAL first would replace the "MAL" in MALE with "M", resulting in "ME" for the new field value.

```
<rule>
  <replace fieldName="Person.Gender" matchPattern="1" replace="M"/>
  <replace fieldName="Person.Gender" matchPattern="MALE" replace="M"/>
  <replace fieldName="Person.Gender" matchPattern="MAL" replace="M"/>
</rule>
```

truncate

This rule checks the value of the specified field against its length as defined in `object.xml`. If the field value exceeds the length, the rule truncates the field to its actual length. The syntax for `truncate` is:

```
<truncate fieldName="name"/>
```

The parameter for `truncate` is:

- *fieldName* – The qualified field name or assigned variable for the field to check and to truncate if it is too long.

EXAMPLE 15 Sample truncate Rule

The following sample checks the lengths defined for the LastName, MaidenName, and MotherMN fields in `object.xml`. If any of these field values are found to be longer than the defined length, the value is truncated to the exact length defined in `object.xml`.

```
<rule>
  <truncate fieldName="Person.LastName"/>
  <truncate fieldName="Person.MaidenName"/>
  <truncate fieldName="Person.MotherMN"/>
</rule>
```

Conditional Data Rules

You can define conditional rules to use in conjunction with the validation and transformation rules described in "Data Validation Rules" on page 27 and "Data Transformation Rules" on page 33. Conditional rules return either true or false. They only define a condition and not an action, so they must be used with other types of rules.

Conditional rules use if, then, and else statements in the following format:

```
<rule>
  <if>
    <condition>
      ...
    </condition>
    <then>
      ...
    </then>
    <else>
      ...
    </else>
  </if>
</rule>
```

The following conditional rules are predefined:

- "dataLength" on page 36
- "equals" on page 37
- "isnull" on page 38
- "matches" on page 38

In addition, you can use the conditional operators described in "Conditional Operators" on page 39.

dataLength

This rule checks the length of the value of the specified field against the length defined in `object.xml`. You can check for lengths greater than or equal to the defined length or less than the defined length. This rule returns "true" if the length matches the specified length range; otherwise it returns "false". The syntax for `dataLength` is:

```
<dataLength fieldName="name" len="length" more="true/false"/>
```

The parameters for `dataLength` are:

- *fieldName* – The qualified field name or assigned variable for the field to check.
- *len* – The length to use to validate the field value.
- *more* – An indicator of whether to check for lengths greater than or equal to the given length or for lengths less than the given length. Specify "true" to check for lengths greater than or equal to the given length, or specify "false" to check for lengths less than the given length.

EXAMPLE 16 Sample dataLength Conditional Rule

The following sample checks postal code fields to verify they do not exceed the United States length of five characters. If a postal code containing more than five characters is found, the record is rejected an written to the bad data file. If a postal code contains five or fewer characters, the field is returned as good.

```
<rule>
  <if>
    <condition>
```

EXAMPLE 16 Sample dataLength Conditional Rule *(Continued)*

```
      <dataLength fieldName="Person.Address.PostalCode" len="6" more="true"/>
    </condition>
    <then>
      <reject fieldName="Person.Address.PostalCode"/>
    </then>
    <else>
      <return fieldName="Person.Address.PostalCode"/>
    </else>
  </if>
</rule>
```

equals

This rule checks whether a specific value is equal to or found within the value of the specified field. This rule returns true if the conditions are matched; otherwise it returns false. The syntax for `equals` is:

```
<equals fieldName="name" value2="value" exact="true/false"/>
```

The parameters for `equals` are:

- *fieldName* – The qualified field name or assigned variable for the field to check.
- *value2* – The value to compare with the value of the specified field.
- *exact* – An indicator of whether to match exactly against the specified value or to find the specified value within the field value. Specify "true" to match exactly against the specified value, or specify "false" to check for the value within the field.

EXAMPLE 17 Sample equals Rule

The following rule checks whether the value of the gender field equals 2, FEM, or FEMALE. If any of those values are found, they are changed to "F". This standardizes the values that indicate a gender of female to the correct processing code for female. If the three listed values are not found in the gender field, the record is returned as good data.

```
<rule>
  <if>
    <condition>
      <or>
        <equals value1="Person.Gender" value2="FEMALE" exact="true"/>
        <equals value1="Person.Gender" value2="2" exact="true"/>
        <equals value1="Person.Gender" value2="FEM" exact="true"/>
        <equals value1="Person.Gender" value2="FML" exact="true"/>
      </or>
    </condition>
    <then>
      <assign fieldName="Person.Gender" value="F"/>
    </then>
    <else>
      <return fieldName="Person.Gender"/>
```

EXAMPLE 17 Sample equals Rule *(Continued)*

```
    </else>
  </if>
</rule>
```

isnull

This rule checks the value of the specified field and returns true if the field value is null. The syntax for `isnull` is:

```
<isnull fieldName="name"/>
```

The parameter for `isnull` is:

- *fieldName* – The qualified field name or assigned variable for the field to check for null values.

EXAMPLE 18 Sample isnull Rule

The following sample checks the first name field for null values. If any first names are null, the record is rejected and written to the bad data file. Otherwise, the field is returned as good data.

```
<rule>
  <if>
    <condition>
      <isnull fieldName="Person.FirstName"/>
    </condition>
    <then>
      <reject fieldName="Person.FirstName"/>
    </then>
    <else>
      <return fieldName="Person.LastName"/>
    </else>
  </if>
</rule>
```

matches

This rule checks the value of the specified field for a certain pattern. If the pattern is found, the rule returns true. Use regular expressions to define the pattern. For more information, see the Javadoc for `java.util.regex`. The syntax for `matches` is:

```
<matches fieldName="name" pattern="pattern"/>
```

The parameters for `matches` are:

- *fieldName* – The qualified field name or assigned variable for the field to check against the given pattern.
- *pattern* – The pattern to find in the given field.

EXAMPLE 19 Sample matches Rule

The following sample checks for phone numbers of the format (ddd)ddd-dddd (where "d" is a digit). Phone fields that are in that format are returned as good data. Fields that are not in that format are rejected and the records are written to the bad data file.

```
<rule>
  <if>
    <condition>
      <matches fieldName="Person.Phone.Phone" pattern="([0-9]{3})[0-9]{3}-[0-9]{4}"/>
    </condition>
    <then>
      <return fieldName="Person.Phone.Phone"/>
    </then>
    <else>
      <reject fieldName="Person.Phone.Phone"/>
    </else>
  </if>
</rule>
```

Conditional Operators

You can use the following conditional operators with the conditional rules.Example 20 and Example 21 provide examples of conditional operator usage for multiple conditions.

- if
- else
- then
- or
- and
- not

EXAMPLE 20 Sample for Nested Conditional Operators

The following sample checks for phone numbers of the format (ddd)ddd-dddd (where "d" is a digit). Phone fields that are in that format are returned as good data. Fields that are not in that format are rechecked to see if the field value is a string of 10 digits. Phone fields that return true for that validation are reformatted to include the parentheses and hyphen. Phone fields that return false for the final validation are rejected and the records are written to the bad data file.

```
<rule>
  <if>
    <condition>
      <matches fieldName="Person.Phone.Phone" pattern="([0-9]{3})[0-9]{3}-[0-9]{4}"/>
    </condition>
    <then>
      <return fieldName="Person.Phone.Phone"/>
    </then>
    <else>
      <if>
        <condition>
          <matches fieldName="Person.Phone.Phone" pattern="[0-9]{10}"/>
```

EXAMPLE 20 Sample for Nested Conditional Operators *(Continued)*

```
        </condition>
        <then>
          <patternReplace fieldname"Person.Phone.Phone" matchPattern="[0-9]{10}"
           replace="([0-9]{3})[0-9]{3}-[0-9]{4}"/>
        </then>
        <else>
          <reject fieldName="Person.Phone.Phone"/>
        </else>
      </if>
    </else>
  </if>
</rule>
```

EXAMPLE 21 Sample for Conditional Operations Using "and"

The following sample checks postal code fields to verify they do not exceed the United States length of five characters. If a postal code containing more than five characters is found, the record is rejected an written to the bad data file. If a postal code contains five or fewer characters, the field is then checked to see whether it contains nine characters (indicating the extension might be appended to the postal code). If it is nine characters, the value is truncated to five characters, leaving the postal code.

```
<rule>
  <if>
    <condition>
      <and>
        <dataLength fieldName="Person.Address.PostalCode" len="5" more="true"/>
        <datalength fieldName="Person.Address.PostalCode" len="6" more="false"/>
      </and>
    </condition>
    <then>
      <return fieldName="Person.Address.PostalCode"/>
    </then>
    <else>
      <if>
        <condition>
          <and>
            <dataLength fieldName="Person.Address.PostalCode" len="9" more="true"/>
            <datalength fieldName="Person.Address.PostalCode" len="10" more="false"/>
          </and>
        </condition>
        <then>
          <truncate fieldName="Person.Address.PostalCode"/>
        </then>
        <else>
            <reject fieldName="Person.Address.PostalCode"/>
        </else>
      </if>
    </else>
  </if>
</rule>
```

Data Profiler Report Samples

The following topics provide sample frequency configurations along with excerpts of the reports they produce.

Simple Frequency Analysis Report Samples

Simple frequency analysis reports list the frequencies of various data values found in the specified fields without using any data verification, transformation, or conditional rules. You can specify a sort order for this type of report, and you can specify a frequency threshold. The sample frequency rule defined below analyzes first and last names, reporting the top 6 frequencies and only if the frequency is three or more.

```
<SimpleFrequencyAnalysis>
  <fields>
    <field fieldName="Person.FirstName"/>
    <field fieldName="Person.LastName"/>
  </fields>
  <sortOrder fieldName="Person.FirstName" increasing="false"/>
  <threshold value="3" more="true"/>
  <topNpatterns value="6" showall="false"/>
</SimpleFrequencyAnalysis>
```

This analysis generates a report similar to the following:

SF_PROFILE_SIMPLE_FRQ_1_1–0.csv

PERSON.LAWSTNAME	PERSON.FIRSTNAME	FREQUENCY
SMITH	ANN	38
JONES	SUSAN	31
SMITH	JOHN	31
THOMPSON	JAMES	28
JOHNSON	BETH	26
MILLER	FRANK	25

The sample frequency rule defined below analyzes social security numbers and analyzes whether there are duplicates (two or more occurrences).

```
<SimpleFrequencyAnalysis>
  <fields>
    <field fieldName="Person.SSN"/>
  </fields>
  <sortOrder fieldName="Person.SSN" increasing="false"/>
  <threshold value="2" more="true"/>
</SimpleFrequencyAnalysis>
```

This analysis generates a report similar to the following excerpt:

SF_PROFILE_SIMPLE_FRQ_2_1–0.csv

PERSON.SSN	FREQUENCY
999999999	457
000000000	125
123456789	41
222423535	4
992203847	2

Constrained Frequency Analysis Report Samples

Constrained frequency analysis reports list the frequencies of various data values found in the specified fields based on defined rules. For example, you can define rules that will only include certain patterns or that will exclude certain values. This topic includes two constrained analysis definitions along with corresponding sample reports.

```
<ConstrainedFrequencyAnalysis>
  <fields>
    <field fieldName="Person.SSN"/>
  </fields>
  <ruleList>
    <rule>
      <dataLength fieldName="Person.SSN" len="10" more="false"/>
    </rule>
  </ruleList>
</ConstrainedFrequencyAnalysis>
```

The above analysis generates a report for social security numbers with less than 10 characters (which means the hyphens are likely missing). Below is a sample output.

CF_PROFILE_CONSTRAINED_FRQ_1_1–100000.csv

PERSON.SSN	FREQUENCY
300555444	1

299557777	1
822331111	2
999999999	98
000000000	115

The following analysis generates a report for dates of birth that are prior to 01/01/1899 (which means they likely contain typographical errors). Below is a sample output.

```
<ConstrainedFrequencyAnalysis>
  <fields>
    <field fieldName="Person.DOB"/>
  </fields>
  <ruleList>
    <rule>
      <dataRange fieldName="Person.DOB" min="01/01/0001" max="01/01/1899"/>
    </rule>
  </ruleList>
</ConstrainedFrequencyAnalysis>
```

CF_PROFILE_CONSTRAINED_FRQ_2_1–100000.csv

PERSON.DOB	FREQUENCY
07/08/53	1
10/04/51	1
09/16/1682	1
12/28/1680	2
05/09/1898	2

Pattern Frequency Analysis Report Samples

Pattern frequency analysis reports list the frequencies of various data patterns found in the values of the specified fields. Patterns are expressed as regular expressions. This topic includes sample reports based on the pattern frequencies defined below for social security number and date of birth patterns.

```
<PatternFrqueencyAnalysis>
  <topNpatterns ="5" showall="true"/>
  <fields>
    <field fieldName="Person.SSN"/>
  </fields>
</PatternFrequencyAnalysis>
```

```
<PatternFrequencyAnalysis>
  <topNpatterns ="5" increasing="true"/>
  <fields>
    <field fieldName="Person.DOB"/>
  </fields>
</PatternFrequencyAnalysis>
```

The above rules generate two reports, one for social security number patterns and one for date of birth patterns. The reports only lists the top 5 patterns. Below are sample outputs for each. You can easily determine invalid values based on the patterns listed.

PF_PROFILE_PATTERN_FRQ_1_1–10000.csv

PERSON.SSN	FREQUENCY
NNN/NN/NNNN	11
	51
NNN-NN-NNN	64
NNNNNNNNN	92
NNN-NN-NNNN	7614

PF_PROFILE_PATTERN_FRQ_2_1–10000.csv

PERSON.DOB	FREQUENCY
NNNN/NN/NN	14
	22
NNNNNNNN	62
NN/NN/NN	84
NN/NN/NNNN	9766