Reference documentation
Reference documentation

5.0
Copyright © 2008-2015

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.
The contents of the document include:

I. Introduction to DataCleaner
   1. Background and concepts
      - What is data quality (DQ)?
      - What is data profiling?
      - What is data wrangling?
      - What is a datastore?
      - Composite datastore
      - What is data monitoring?
      - What is master data management (MDM)?

2. Getting started with DataCleaner desktop
   - Installing the desktop application
   - Connecting to your datastore
   - Adding components to the job
   - Wiring components together
   - Transformer output
   - Filter requirement
   - Output data streams
   - Executing jobs
   - Saving and opening jobs
   - Template jobs
   - Writing cleansed data to files

3. Getting started with DataCleaner monitor
   - Installing the monitoring web application
   - Connecting to your datastore
   - Building a job
   - Scheduling jobs
   - Adding metric charts on the dashboard

II. Analysis component reference

4. Transform
   - JavaScript transformer
   - Invoke child Analysis job
   - Equals
   - Max rows
   - Not null
   - Union

5. Improve
   - Duplicate detection
      - 'Model training' mode
      - 'Detection' mode
      - 'Untrained detection' mode

   Merge duplicates
6. Analyze ................................................................................................. 39
   Merge duplicates .................................................................................. 39
   Conclusion ............................................................................................. 41
   Synonym lookup .................................................................................. 41
   DE movers and deceased check

   Commercial editions only ...................................................................... 42
   Address and Mail Suppression data sources ........................................ 42
   Output ................................................................................................... 43
   UK movers, deceased and mailing preferences check

   Commercial editions only ...................................................................... 44
   Address and Mail Suppression data sources ........................................ 44
   Output ................................................................................................... 45
   US movers, deceased and do-not-mail check

   Commercial editions only ...................................................................... 46
   Address and Mail Suppression data sources ........................................ 46
   Output ................................................................................................... 46
   Table lookup .......................................................................................... 47
   National identifiers

   Commercial editions only ...................................................................... 49

   6. Analyze ................................................................................................. 49
      Boolean analyzer .................................................................................. 50
      Completeness analyzer ....................................................................... 51
      Character set distribution ................................................................... 52
      Date gap analyzer ................................................................................ 52
      Date/time analyzer .............................................................................. 52
      Number analyzer ................................................................................. 52
      Pattern finder ........................................................................................ 52
      Reference data matcher ....................................................................... 55
      Referential integrity ............................................................................. 55
      String analyzer ..................................................................................... 57
      Unique key check .................................................................................. 57
      Value distribution .................................................................................. 57
      Value matcher ....................................................................................... 58
      Weekday distribution ............................................................................. 58

   7. Write ....................................................................................................... 59
      Create CSV file ..................................................................................... 59
Reference documentation

Create Excel spreadsheet ............................................................... 59
Create staging table ....................................................................... 59
Insert into table ............................................................................ 60
Update table ................................................................................ 61
III. Reference data .............................................................................. 62
  8. Dictionaries ................................................................................. 64
  9. Synonyms (aka. Synonym catalogs) ............................................. 65
     Text file synonym catalog ............................................................. 65
     Datastore synonym catalog .......................................................... 65
  10. String patterns ........................................................................... 66
IV. Configuration reference ............................................................. 67
  11. Configuration file ......................................................................... 69
     XML schema ............................................................................... 69
     Datastores ................................................................................ 69
        Database (JDBC) connections ..................................................... 69
        Comma-Separated Values (CSV) files ....................................... 70
        Fixed width value files ............................................................. 70
        Excel spreadsheets ................................................................. 71
        XML file datastores ............................................................... 71
        ElasticSearch index ............................................................... 72
        MongoDB databases .............................................................. 72
        CouchDB databases ............................................................... 73
        Composite datastore ............................................................. 74
     Reference data ............................................................................ 74
        Dictionaries ............................................................................ 74
        Synonym catalogs .................................................................. 75
        String patterns ....................................................................... 76
        Task runner .............................................................................. 77
        Storage provider ..................................................................... 77
  12. Analysis job files ......................................................................... 79
     XML schema ............................................................................... 79
     Source section ............................................................................ 79
  13. Logging ........................................................................................... 81
     Logging configuration file .......................................................... 81
     Default logging configuration ..................................................... 81
     Modifying logging levels ............................................................ 82
     Alternative logging outputs ........................................................ 82
  14. Database drivers .......................................................................... 84
     Installing Database drivers in DataCleaner desktop ..................... 84
     Installing Database drivers in DataCleaner monitor ..................... 86
V. DataCleaner monitor repository ................................................... 87
  15. Repository configuration ........................................................... 89
     Configure repository location ....................................................... 89
          Directory-based repository ...................................................... 89
          Database-backed repository .................................................... 89
     Providing signed Java WebStart client files ................................. 90
     Producing the signed JARs .......................................................... 90
     Configuring DataCleaner monitor to use the signed JARs .......... 91
     Cluster configuration (distributed execution) ............................. 91

Commercial editions only
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. DataCleaner monitor web services</td>
<td>94</td>
</tr>
<tr>
<td>17. Job triggering</td>
<td>96</td>
</tr>
<tr>
<td>Trigger service</td>
<td>96</td>
</tr>
<tr>
<td>Polling for execution status</td>
<td>97</td>
</tr>
<tr>
<td>18. Repository navigation</td>
<td>98</td>
</tr>
<tr>
<td>Job files</td>
<td>98</td>
</tr>
<tr>
<td>Result files</td>
<td>98</td>
</tr>
<tr>
<td>Uploading content to the repository</td>
<td>99</td>
</tr>
<tr>
<td>Modifying result metadata</td>
<td>99</td>
</tr>
<tr>
<td>Renaming jobs</td>
<td>100</td>
</tr>
<tr>
<td>Copying jobs</td>
<td>100</td>
</tr>
<tr>
<td>Deleting jobs</td>
<td>101</td>
</tr>
<tr>
<td>19. Metric web services</td>
<td>102</td>
</tr>
<tr>
<td>Metrics background</td>
<td>102</td>
</tr>
<tr>
<td>Getting a list of available metrics</td>
<td>102</td>
</tr>
<tr>
<td>Getting the values of particular metrics</td>
<td>102</td>
</tr>
<tr>
<td>20. Atomic transformations (data cleaning as a service)</td>
<td>104</td>
</tr>
<tr>
<td>What are atomic transformation services?</td>
<td>104</td>
</tr>
<tr>
<td>Invoking atomic transformations</td>
<td>104</td>
</tr>
<tr>
<td>7. Invoking DataCleaner jobs</td>
<td>107</td>
</tr>
<tr>
<td>21. Command-line interface</td>
<td>109</td>
</tr>
<tr>
<td>Executables</td>
<td>109</td>
</tr>
<tr>
<td>Usage scenarios</td>
<td>109</td>
</tr>
<tr>
<td>Executing an analysis job</td>
<td>110</td>
</tr>
<tr>
<td>Listing datastore contents and available components</td>
<td>110</td>
</tr>
<tr>
<td>Parameterizable jobs</td>
<td>112</td>
</tr>
<tr>
<td>Dynamically overriding configuration elements</td>
<td>113</td>
</tr>
<tr>
<td>22. Apache Hadoop and Spark interface</td>
<td>115</td>
</tr>
<tr>
<td>Hadoop deployment overview</td>
<td>115</td>
</tr>
<tr>
<td>Setting up Spark and DataCleaner environment</td>
<td>116</td>
</tr>
<tr>
<td>Upload configuration file to HDFS</td>
<td>116</td>
</tr>
<tr>
<td>Upload job file to HDFS</td>
<td>117</td>
</tr>
<tr>
<td>Upload executables to HDFS</td>
<td>118</td>
</tr>
<tr>
<td>Launching DataCleaner jobs using Spark</td>
<td>118</td>
</tr>
<tr>
<td>Limitations of the Hadoop interface</td>
<td>119</td>
</tr>
<tr>
<td>8. Third party integrations</td>
<td>120</td>
</tr>
<tr>
<td>23. Pentaho integration</td>
<td>122</td>
</tr>
<tr>
<td>Configure DataCleaner in Pentaho Data Integration</td>
<td>122</td>
</tr>
<tr>
<td>Launch DataCleaner to profile Pentaho Data Integration steps</td>
<td>123</td>
</tr>
<tr>
<td>Run Pentaho Data Integration jobs in DataCleaner monitor</td>
<td>124</td>
</tr>
<tr>
<td>Run DataCleaner jobs in Pentaho Data Integration</td>
<td>125</td>
</tr>
<tr>
<td>9. Developer's guide</td>
<td>127</td>
</tr>
<tr>
<td>24. Architecture</td>
<td>129</td>
</tr>
<tr>
<td>Data access</td>
<td>129</td>
</tr>
<tr>
<td>Processing framework</td>
<td>129</td>
</tr>
<tr>
<td>25. Executing jobs through code</td>
<td>131</td>
</tr>
</tbody>
</table>
Overview of steps and options ................................................................. 131
Step 1: Configuration ........................................................................... 132
Step 2: Job ......................................................................................... 134
Step 3: Execution ............................................................................... 137
Step 4: Result ..................................................................................... 139
26. Developer resources ........................................................................ 141
    Extension development tutorials .................................................... 141
    Building DataCleaner ..................................................................... 141
27. Extension packaging ....................................................................... 142
    Annotated components .................................................................... 142
    Single JAR file ............................................................................... 142
    Extension metadata XML ................................................................ 142
    Component icons ............................................................................ 143
28. Embedding DataCleaner ................................................................. 144
List of Tables

4.1. JavaScript variables ................................................................. 29
4.2. JavaScript data types ............................................................... 30
5.1. Model training properties ........................................................ 35
5.2. DE movers and deceased check output ...................................... 43
5.3. DE movers and deceased check output ...................................... 45
5.4. DE movers and deceased check output ...................................... 46
6.1. Completeness analyzer properties ............................................. 52
6.2. Pattern finder properties ........................................................ 53
6.3. Referential integrity properties ................................................ 56
6.4. Unique key check properties ................................................... 57
6.5. Value distribution properties .................................................. 57
17.1. Job triggering HTTP parameters .............................................. 96
Part I. Introduction to DataCleaner
Table of Contents

1. Background and concepts ................................................................. 3
   What is data quality (DQ)? ............................................................. 3
   What is data profiling? ................................................................. 3
   What is data wrangling? ............................................................... 4
   What is a datastore? ..................................................................... 4
   Composite datastore .................................................................... 5
   What is data monitoring? ............................................................. 5
   What is master data management (MDM)? ................................... 5

2. Getting started with DataCleaner desktop ....................................... 6
   Installing the desktop application .................................................. 6
   Connecting to your datastore ....................................................... 6
   Adding components to the job ....................................................... 8
   Wiring components together ........................................................ 9
      Transformer output ..................................................................... 9
      Filter requirement ..................................................................... 10
      Output data streams .................................................................. 11
   Executing jobs ............................................................................. 13
   Saving and opening jobs ............................................................ 13
   Template jobs ............................................................................. 14
   Writing cleansed data to files ...................................................... 16

3. Getting started with DataCleaner monitor ....................................... 17
   Installing the monitoring web application ...................................... 17
   Connecting to your datastore ....................................................... 17
   Building a job ............................................................................. 18
   Scheduling jobs .......................................................................... 19
   Adding metric charts on the dashboard ........................................ 22
Chapter 1. Background and concepts

Abstract

In this chapter we will try to define how we see the concepts and terms surrounding the environment(s) around DataCleaner.

Although these terms have no strict definitions, you can use this chapter as a guide, at least for the scope of how to use and what to expect from DataCleaner in relation to the described topics.

As a lot of the statements in this chapter are in deed subjective or based upon personal experience, we encourage everyone to provide their feedback and to contribute corrections/improvements to it.

What is data quality (DQ)?

Data Quality (DQ) is a concept and a business term covering the quality of the data used for a particular purpose. Often times the DQ term is applied to the quality of data used in business decisions but it may also refer to the quality of data used in research, campaigns, processes and more.

Working with Data Quality typically varies a lot from project to project, just as the issues in the quality of data vary a lot. Examples of data quality issues include:

1. Completeness of data
2. Correctness of data
3. Duplication of data
4. Uniformedness/standardization of data

A less technical definition of high-quality data is, that data are of high quality "if they are fit for their intended uses in operations, decision making and planning" (J. M. Juran).

Data quality analysis (DQA) is the (human) process of examining the quality of data for a particular process or organization. The DQA includes both technical and non-technical elements. For example, to do a good DQA you will probably need to talk to users, business people, partner organizations and maybe customers. This is needed to assess what the goal of the DQA should be.

From a technical viewpoint the main task in a DQA is the data profiling activity, which will help you discover and measure the current state of affairs in the data.

What is data profiling?

Data profiling is the activity of investigating a datastore to create a 'profile' of it. With a profile of your datastore you will be a lot better equipped to actually use and improve it.

The way you do profiling often depends on whether you already have some ideas about the quality of the data or if you're not experienced with the datastore at hand. Either way we recommend an explorative approach, because even though you think there are only a certain amount of issues you need to look for, it is our experience (and reasoning behind a lot of the features of DataCleaner) that it is just as important to check those items in the data that you think are correct! Typically it's cheap to include a bit more data into your analysis and the results just might surprise you and save you time!

DataCleaner comprises (amongst other aspects) a desktop application for doing data profiling on just about any kind of datastore.
What is data wrangling?

From Wikipedia, we get a good introductory explanation of the term 'Data wrangling':

Tip

Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption of the data with the help of semi-automated tools. This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses.

As can be seen, it's often important to do changes to data in order to analyze it. That's why DataCleaner bundles a lot of transformational and improvement features which allow the user to not only profile data, but also to reshape it into a form that is fitting.

What is a datastore?

A datastore is the place where data is stored. Usually enterprise data lives in relational databases, but there are numerous exceptions to that rule.

To comprehend different sources of data, such as databases, spreadsheets, XML files and even standard business applications, we employ the umbrella term datastore. DataCleaner is capable of retrieving data from a very wide range of datastores. And furthermore, DataCleaner can update the data of most of these datastores as well.

A datastore can be created in the UI or via the configuration file. You can create a datastore from any type of source such as: CSV, Excel, Oracle Database, MySQL, etc.
Composite datastore

A composite datastore contains multiple datastores. The main advantage of a composite datastore is that it allows you to analyze and process data from multiple sources in the same job.

What is data monitoring?

We’ve argued that data profiling is ideally an explorative activity. Data monitoring typically isn't! The measurements that you do when profiling often times needs to be continuously checked so that your improvements are enforced through time. This is what data monitoring is typically about.

Data monitoring solutions come in different shapes and sizes. You can set up your own bulk of scheduled jobs that run every night. You can build alerts around it that send you emails if a particular measure goes beyond its allowed thresholds, or in some cases you can attempt ruling out the issue entirely by applying First-Time-Right (FTR) principles that validate data at entry-time. eg. at data registration forms and more.

As of version 3, DataCleaner now also includes a monitoring web application, dubbed "DataCleaner monitor". The monitor is a server application that supports orchestrating and scheduling of jobs, as well as exposing metrics through web services and through interactive timelines and reports. It also supports the configuration and job-building process through wizards and management pages for all the components of the solution. As such, we like to say that the DataCleaner monitor provides a good foundation for the infrastructure needed in a Master Data Management hub.

What is master data management (MDM)?

Master data management (MDM) is a very broad term and is seen materialized in a variety of ways. For the scope of this document it serves more as a context of data quality than an activity that we actually target with DataCleaner per-se.

The overall goals of MDM is to manage the important data of an organization. By "master data" we refer to "a single version of the truth", ie. not the data of a particular system, but for example all the customer data or product data of a company. Usually this data is dispersed over multiple datastores, so an important part of MDM is the process of unifying the data into a single model.

Obviously another of the very important issues to handle in MDM is the quality of data. If you simply gather eg. "all customer data" from all systems in an organization, you will most likely see a lot of data quality issues. There will be a lot of duplicate entries, there will be variances in the way that customer data is filled, there will be different identifiers and even different levels of granularity for defining "what is a customer?". In the context of MDM, DataCleaner can serve as the engine to cleanse, transform and unify data from multiple datastores into the single view of the master data.
Chapter 2. Getting started with DataCleaner desktop

Installing the desktop application

These are the system requirements of DataCleaner:

1. A computer (with a graphical display, except if run in command-line mode).
2. A Java Runtime Environment [http://www.java.com] (JRE), version 7 or higher.
3. A DataCleaner software license file for professional editions. If you've requested a free trial or purchased DataCleaner online, this file will have been sent to your email address.

Start the installation procedure using the installer program. The installer program is an executable JAR file, which is executable on most systems if you simply double-click it.

**Tip**

If the installer does not launch when you double-click it, open a command prompt and enter:

**Troubleshooting**

Usually the installation procedure is trivial and self-explanatory. But in case something is not working as expected, please check the following points:

1. On Windows systems, if you do not have *Administrative privileges* on the machine, we encourage you to install DataCleaner in your user's directory instead of in 'Program Files'.

2. On some Windows systems you can get a warning *'There is no script engine for file extension '.js' '.* This happens when .js files (JavaScript) files are associated with an editor instead of Windows' built-in scripting engine. To resolve this issue, please refer to these help links:
   a. answers.microsoft.com [http://answers.microsoft.com/en-us/windows/forum/windows_7-system/fix-there-is-no-script-engine-for-file-extension/95d58867-3f31-45e3-aef-32ecea4d33c1], which address the issue and recommends...
   b. winhelponline.com [http://www.winhelponline.com/articles/230/1/Error-There-is-no-script-engine-for-file-extension-when-running-js-files.html], which has a fix for the issue

3. If you have issues with locating or selecting the software license file, you can skip the step in the installer by copying the license file manually to this folder: ‘~/.datacleaner’ (where ~ is your User's home folder). Note that on Windows machines it is prohibited by Windows explorer to create directories starting with dot (.), but it can be done using the command prompt:

**Connecting to your datastore**

Below is a screenshot of the initial screen that will be presented when launching DataCleaner (desktop community edition). A new datastore can be added in the "New job from scratch" or in "Manage datastores" screens available by clicking the buttons in the bottom of the screen.
File datastores can be added using a drop zone (or browse button) located in the top of the screen. Below, there are buttons that enable adding databases or cloud services.
If the file is added using the drop zone, its format will be inferred. If you need more control over how the file is going to be interpreted, use the alternative way to add a new datastore - "Manage datastores" button in the welcome screen.

The "Datastore management" screen - except from viewing and editing existing datastores - has an option to add a new one based on its type. Choose an icon in the bottom of the screen that suits your datastore type.

Once you've registered ('created') your own datastore, you can select it from the list and (in "New job from scratch" screen) or select it from the list and click "Build job" (in "Datastore Management" screen) to start working with it!

Tip

You can also configure your datastore by means of the configuration file (conf.xml), which has both some pros and some cons. For more information, read the configuration file chapter.

Adding components to the job

There are a few different kinds of components that you can add to your job:

1. **Analyzers**, which are the most important components. Actually, without at least one analyzer the job will not run (if you execute the job without adding one, DataCleaner will suggest adding a basic one that will save the output to a file). An analyzer is a component that inspects the data that it receives and generates a result or a report. The majority of the data profiling cruft is created as analyzers.

2. **Transformers** are components used to modify the data before analyzing it. Sometimes it's neccessary to extract parts of a value or combine two values to correctly get an idea about a particular measure. In
other scenarios, transformers can be used to perform reference data lookups or other similar tasks and place the results of an operation into the stream of data in the job.

The result of a transformer is a set of output columns. These columns work exactly like regular columns in your job, except that they have a preceding step in the flow before they become materialized.

3. Filters are components that split the flow of processing in a job. A filter will have a number of possible outcomes and depending on the outcome of a filter, a particular row might be processed by different sub-flows. Filters are often used simply to disregard certain rows from the analysis, eg. null values or values outside the range of interest.

Each of these components will be presented as a node in the job graph. Double-clicking a component (graph node) will bring its configuration dialog.

Transformers and filters are added to your job using the "Transform" and "Improve" menus. The menus are available in component library on the left or by right-clicking on an empty space in the canvas. Please refer to the reference chapter Transformations for more information on specific transformers and filters.

Analyzers are added to your job using the "Analyze" menu (in most cases), but also "Write" menu for analyzers that save output to a datastore. Please refer to the reference chapter Analyzers for more information on specific analyzers.

Wiring components together

Simply adding a transformer or filter actually doesn't change your job as such! This is because these components only have an impact if you wire them together.

Transformer output

To wire a transformer you simply need to draw an arrow between the components in the graph. You can start drawing it by right-clicking the first of the components and choosing "Link to..." from the context menu. An alternative way to enter the drawing mode is to select the component and connect the components with Shift button pressed.
Filter requirement

To wire a filter you need to set up a dependency on either of its outcomes. All components have a button for selecting filter outcomes in the top-right corners of their configuration dialogs. Click this button to select a filter outcome to depend on.

If you have multiple filters you can chain these simply by having dependent outcomes of the individual filters. This will require all filter requirements in the chain to be met, for a record to be passed to the component (AND logic).
Chained filters

Using "Link to...", it is also possible to wire several filters to a component in a kind of diamond shape. In that case, if any of the filter requirements are met, the record will be passed to the component (OR logic).

"Diamond" filters

Output data streams

The "Link to..." option wires components together in the "main flow". However, some components are able to produce additional output data streams. For example, the main feature of a Completeness Analyzer is to produce a summary of records completeness in the job result window. Additionally, it produces two output data streams - "Complete records" and "Incomplete records". Output data streams behave similarly to a source table, although such a table is created dynamically by a component. This enables further processing of such output.

Components producing output data streams have additional "Link to..." position in the right-click menu to wire the output with subsequent components.

Instead of wiring components with "Link to..." menu option, double-clicking a component brings up a configuration dialog that can be used to choose its input columns. In the top-right corner of the dialog, the scope of the component can be chosen. Switching between scopes gives us the possibility to choose input columns from the "main flow" (default scope) or from output data streams.
An example job using output data streams:

Tip

The canvas displays messages (in the bottom of the screen) which contain instructions with the next steps that need to be performed in order to build a valid job.
Executing jobs

When a job has been built you can execute it. To check whether your job is correctly configured and ready to execute, check the status bar in the bottom of the job building window.

To execute the job, simply click the "Execute" button in the top-right corner of the window. This will bring up the result window, which contains:

1. The *Progress information* tab which contains useful information and progress indications while the job is being executed.

2. Additional tabs for each component type that produces a result/report. For instance 'Value distribution' if such a component was added to the job.

Here's an example of an analysis result window:

![Analysis result window](image)

Saving and opening jobs

You can save your jobs in order to reuse them at a later time. Saving a job is simple: just click the "Save" button in the top panel of the window.

Analysis jobs are saved in files with the ".analysis.xml" extension. These files are XML files that are readable and editable using any XML editor.

Opening jobs can be done using the "Open" menu item. Opening a job will restore a job building window from where you can edit and run the job.
Template jobs

DataCleaner contains a feature where you can reuse a job for multiple datastores or just multiple columns in the same datastore. We call this feature 'template jobs'.

When opening a job you are presented with a file chooser. When you select a job file a panel will appear, containing some information about the job as well as available actions:

If you click the 'Open as template' button you will be presented with a dialog where you can map the job's original columns to a new set of columns:
Getting started with DataCleaner desktop

First you need to specify the datastore to use. On the left side you see the name of the original datastore, but the job is not restricted to use only this datastore. Select a datastore from the list and the fields below for the columns will become active.

Then you need to map individual columns. If you have two datastore that have the same column names, you can click the "Map automatically" button and they will be automatically assigned. Otherwise you need to map the columns from the new datastore's available columns.

Finally your job may contain 'Job-level variables'. These are configurable properties of the job that you might also want to fill out.

Once these 2-3 steps have been completed, click the "Open job" button, and DataCleaner will be ready for executing the job on a new set of columns!
Writing cleansed data to files

Although the primary focus of DataCleaner is analysis, often during such analysis you will find yourself actually improving data by means of applying transformers and filters on it. When this is the case, obviously you will want to export the improved/cleansed data so you can utilize it in other situations than the analysis.

Please refer to the reference chapter Writers for more information on writing cleansed data.
Chapter 3. Getting started with DataCleaner monitor

Installing the monitoring web application

In addition to (and in some cases, even as a replacement for) the desktop version of DataCleaner, we also provide a web application for monitoring, scheduling and sharing analysis jobs and results.

A Java servlet container and web server is required to run the monitoring web application. An example of this is Apache Tomcat 7.x [http://tomcat.apache.org/download-70.cgi], which is often used and tested by the DataCleaner development team.

To install the monitoring web application, download [https://datacleaner.org/downloads] the Web Archive (.war) distribution of DataCleaner. Install the .war file in your container. If you’re using Apache Tomcat, this is done by copying the .war file to the "webapps" folder within your tomcat directory. Afterwards, start the container and go to http://localhost:8080/DataCleaner-monitor to see the welcome/login screen:

In the community edition of DataCleaner, you will find suggestions for login credentials directly on the screen, to get you started quickly.

Connecting to your datastore

DataCleaner monitor displays and manages datastores through the 'Datastores' page. Click the "Register datastore" button to set up a new datastore:
Follow the on-screen instructions to go through the process of registering the datastore. When the wizard is finished you can find the datastore on the 'Datastores' page where you can work with it in different ways - e.g. do ad-hoc querying, download the data (if it is a file-based datastore), launch DataCleaner desktop to build custom jobs etc.

**Tip**

You can also configure your datastore by means of the configuration file (conf.xml), which has both some pros and some cons. For more information, read the configuration file chapter.

**Building a job**

There are multiple ways to add a new job to DataCleaner monitor. The easiest way is to use one of the job wizards. These are found on the 'Scheduling' page, using the "Build job" button.
Simply follow the on-screen instructions to build the job according to the wizard's intent.

Once the job has been built, you will have options to trigger its execution, schedule it, set up alerting and start building monitoring timelines for its result's metrics.

Alternatively, you can create the job using the desktop application. In that case, make sure that the name of any involved datastore in the desktop application matches the name of the datastore in the repository! If so, you can safely drop the .analysis.xml job file in the jobs folder on the repository. It will be immediately available in the web UI of the monitor application.

**Scheduling jobs**

The DataCleaner monitor web application is the preferred application for scheduling of DataCleaner jobs.

Below is a screenshot of the Scheduling page of the DataCleaner monitor. To access this page, your user needs to have the ROLE_SCHEDULE_EDITOR security role.
Getting started with DataCleaner monitor

Things to notice on this page:

1. Each job is categorized by their datastores. This is to provide an overview of which jobs are executing based on which datastores. Technically, a job can interact with more than one datastore, but it will be categorized on this page based on which datastore is acting as source for the job's execution flow.

2. Besides the datastore and job name, we have a Scheduling column. In the case shown, all jobs are 'Manually triggered', meaning that they are not scheduled by the DataCleaner monitor. Click the link to change the trigger:
Getting started with
DataCleaner monitor

Customize schedule

You can configure the schedule of a job in a number of different ways. The most common way is by using a periodic trigger. Hereby your job gets triggered at a periodic interval, determined by the scheduler. Alternatively you can set up a dependent trigger, where the job gets triggered once another job has finished executing. This is convenient in order to connect a larger flow of jobs in sequence. Either you can select to only allow manual triggering of the job. In that case your job will not be automatically invoked, but you can invoke it manually using this web application, or using a web service invocation.

- **Periodic trigger**

  Please provide a scheduling expression (cron syntax):
  
  ![Cron expression input field]

  Use cron syntax by appending "seconds minutes hours day-of-month month-of-year day-of-week". Example: Every day at 14:05 is "0 14 * * ?".
  Or use a keyword: @monthly, @daily, @hourly. (Press @ to show all).

- **Dependent trigger**

  Please select the preceding job that this job will run after:
  
  ![Dependent trigger selection field]

- **Manual trigger**

  The job will not be automatically invoked.

  ![Save schedule and Cancel buttons]

The three types of triggering are explained on the screen: Periodic trigger, Dependent trigger, Manual trigger.

3. In the next column **Alerts** are defined. Alerts define conditions that should be inspected after execution. An alert is based on metrics, similar to those that can be shown in the timelines on the Dashboard page. An alert also contains a severity, defining how severe the alert is, if it is raised. The severity scale is (increasing severity): Intelligence, Surveillance, Warning, Fatal.

Typical handling of raised alerts include sending an email or SMS to the data stewards. These alert notification mechanisms are included in the commercial editions of DataCleaner. You can also develop your own alert notification handlers, by implementing the org.datacleaner.monitor.alertnotification.AlertNotifier interface in Java.

4. In the last column, **Actions** can be found. The following options are there:
a. **Trigger**, which triggers a job for immediate execution.

b. **Open**, which launches the DataCleaner desktop application with the job loaded. Note that for security and bandwidth concerns, only the first 20 input records of the job is transferred to the desktop application. It is in other words not possible to "trigger" the job run on the desktop as if it is running on the server, but it provides a good playground for experiments and testing. Jobs can even be modified and saved back onto the monitoring server this way.

c. **History**, which displays a dialog with log information about each triggered job execution.

### Adding metric charts on the dashboard

On the 'Dashboard' page you can set up a visual representation of the metrics that your jobs are gathering. The DataCleaner monitor comes with some example timeline charts that demonstrate how metrics are plotted, that you can click at a point in these timeline charts and see further details about those metrics.

Now let's see how we can define our own timelines. Select the appropriate group on the left-hand side of the page and then click "New timeline chart". You'll have to select the job which to base the metric data upon and then select from the metrics made available by that job:

Note that the contents of the dialog are highly dependent on the selected job, so the screenshot above will not necessarily resemble the options available to you.

There are four different types of metrics in DataCleaner monitor, and they may or may not be exposed through your job:
1. **Non-parameterized metrics.** These are the most "plain" metrics, which will be represented only by a checkbox.

2. **String-parameterized metrics.** These are metrics that take a string parameter. For instance we see that the "Match count" metric of the "Pattern finder" component takes a string parameter (screenshot above). In this case the parameter represents the pattern to look for when counting matches. Usually suggestions will be shown in the UI to help you figure out what type of parameter is expected.

3. **Column-parameterized metrics.** Some metrics are collected per column. Since we know which columns a specific component is working on, a list of columns and corresponding checkboxes are shown to make the parameter selection easier:

   ![String analyzer (5 columns)](image)

   - Avg chars:
   - Avg white spaces:
   - Blank count:
   - Diacritic chars:
   - Digit chars:
   - Entirely lowercase count:
   - Entirely uppercase count:
   - Lowercase chars:

4. **Formula metrics.** Lastly, you can define your own metrics based on a formula of other metrics. This functionality is similar to building formulas in a spreadsheet - you simply reference other metrics and provide an equation for calculating the metric.

   Adding a formula metric is done by clicking the "Add metric formula" button and filling out the formula details:
Metric formulas support the basic mathematical operators: plus (+), minus (-), divide (/) and multiply (*).

When the metric selection is done, a new unsaved timeline is shown:
Getting started with DataCleaner monitor

You can go back to selecting the metrics or customizing the chart's look and feel by clicking the 'Wrench' icon, or you can start by saving it using the save icon.

Names and colors of the metrics in the timeline chart can be customized by clicking the legend of the chart.
Part II. Analysis component reference
# Table of Contents

4. Transform .................................................................................................................. 29  
   JavaScript transformer ............................................................................................ 29  
   Invoke child Analysis job ....................................................................................... 31  
   Equals ...................................................................................................................... 32  
   Max rows .................................................................................................................. 32  
   Not null ..................................................................................................................... 33  
   Union ......................................................................................................................... 33  
5. Improve ..................................................................................................................... 34  
   Duplicate detection

| Merge duplicates
| 'Model training' mode ............................................................................................... 34  
| 'Detection' mode ...................................................................................................... 38  
| 'Untrained detection' mode .................................................................................... 39  

| Merge duplicates
| Conclusion ................................................................................................................... 39  
| Synonym lookup ....................................................................................................... 41  

DE movers and deceased check

| Address and Mail Suppression data sources ............................................................... 42  
| Output .......................................................................................................................... 43  

| UK movers, deceased and mailing preferences check
| Address and Mail Suppression data sources ............................................................... 44  
| Output .......................................................................................................................... 45  

| US movers, deceased and do-not-mail check
| Address and Mail Suppression data sources ............................................................... 46  
| Output .......................................................................................................................... 46  


Table lookup ................................................................. 47
National identifiers

6. Analyze ........................................................................ 50
   Boolean analyzer ......................................................... 51
   Completeness analyzer ............................................... 51
   Character set distribution ............................................ 52
   Date gap analyzer ....................................................... 52
   Date/time analyzer ...................................................... 52
   Number analyzer ......................................................... 52
   Pattern finder ............................................................ 52
   Reference data matcher .............................................. 55
   Referential integrity ................................................... 55
   String analyzer .......................................................... 57
   Unique key check ....................................................... 57
   Value distribution ..................................................... 57
   Value matcher .......................................................... 58
   Weekday distribution ................................................ 58

7. Write .......................................................................... 59
   Create CSV file .......................................................... 59
   Create Excel spreadsheet .......................................... 59
   Create staging table .................................................. 59
   Insert into table ......................................................... 60
   Update table ............................................................ 61
Chapter 4. Transform

Abstract

With transformations you can pre- and postprocess your data as part of your DQ project.

Technically speaking there are two kinds of transformations: Transformers and Filters. Transformers are used to extract, generate or refine data (new columns and sometimes also new rows), whereas filters are used to limit or split the dataset into separate processing streams.

There's quite a lot of transformations available in DataCleaner, more than will be feasible to describe all in detail. This chapter provides a documentation for some of the essential ones.

JavaScript transformer

The JavaScript transformer allows the user to define his/her own script which can perform rather intricate things like conditioning, looping. It can also be used as a way to express small business rules.

For this documentation, a complete reference of JavaScript is out of scope. But we will show a few examples and more importantly talk about the available variables and their types.

The JavaScript transformer returns a single string. The entered script should provide this string as the last line of the script. This is why the template script is as follows (so you can just implement the eval() function):

```javascript
function eval() {
  return "hello \" + values[0];
}
eval();
```

Variables:

Table 4.1. JavaScript variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>An array of all values in the row (as mapped by the &quot;Columns&quot; property). Using &quot;values&quot; you can reference eg. the first and third values like this:</td>
</tr>
<tr>
<td></td>
<td>var first = values[0]; var third = values[2];</td>
</tr>
<tr>
<td></td>
<td>Note that JavaScript arrays are 0-based. Instead of indexes you can also reference by column name, like this:</td>
</tr>
</tbody>
</table>
var idValue = values["id"];  

*column_name*  
Any column name that is also a valid JavaScript and not a reserved variable name will also be added directly to the scope of the script as a variable. For example, if you have two columns, FIRST_NAME and LAST_NAME, you can concatenate them easily, like this:

```javascript
var fullname = FIRST_NAME + " " + LAST_NAME;
```

**out**  
A reference to the system console's "out" stream. If running DataCleaner with the console visible, you can print messages to the console, like this:

```javascript
out.println("Value: " + values[0]);
```

**log**  
A reference to the logging subsystem. Logging can be configured and log messages are stored in files, which makes it a bit more flexible than simply using "out". Here's how you write a few log messages with varying severities:

```javascript
log.debug("This is a DEBUG message, it will probably be disregarded");
log.info("This is a INFO message, it will probably be written to the logs");
log.warn("This is a WARN message, it will most likely be written to the logs");
log.error("This is a ERROR message, it will almost certainly be written to the logs");
```

Data types:

**Table 4.2. JavaScript data types**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>String values are represented as JavaScript strings, which means that they have (among others) methods like:</td>
</tr>
<tr>
<td></td>
<td>var str = values[0];</td>
</tr>
<tr>
<td></td>
<td>// get the length of a string var len = str.length();</td>
</tr>
<tr>
<td></td>
<td>// uppercase variant of a string var up = str.toUpperCase();</td>
</tr>
<tr>
<td></td>
<td>// lowercase variant of a string var lw = str.toLowerCase();</td>
</tr>
<tr>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **NUMBER** | Numbers are treated as regular JavaScript numbers, which means that they have (among others) methods and operators like:  

```javascript
var num = values[0];

// format with 2 decimals
var formattedNumber = num.toFixed(2);

// add, subtract, multiply or divide
var m = (4 + num * 2 - 1) / 2;
```

For more information, we recommend W3 schools JavaScript number reference [http://www.w3schools.com/jsref/jsref_obj_number.asp] and also check out the Math function [http://www.w3schools.com/jsref/jsref_obj_math.asp] reference. |
| **DATE** | Date values are treated as Java dates, which is a bit unusual, but leaves you with almost an identical interface as a regular JavaScript date. Here’s a summary of typical methods:  

```javascript
var d = values[0];

var year = d.getYear();
var month = d.getMonth();
var date = d.getDate();
var hour = d.getHour();
var minutes = d.getMinutes();
var seconds = d.getSeconds();

// milliseconds since 1970-01-01
var timestamp = d.getTime();
```

For a full reference, please look at the Java Date class reference [http://download.oracle.com/javase/6/docs/api/java/util/Date.html] . |
| **BOOLEAN** | Boolean (true/false) values are simply booleans, no sugar coating added :) |

**Invoke child Analysis job**

With this transformation it is possible to compose a job that embeds functionality from another saved job.
The job that is invoked will be stripped from it’s analysis section, which means that only transformations remain. That way you can split up complex transformation flows into smaller pieces - each represented as a job file.

To configure the transformation, select the columns from your current job’s source which needs to be piped into the child job. The amount of columns selected needs to be the same as the amount of columns defined in the child job. This will be checked automatically, making it impossible to run the transformation otherwise.

Equals

The *Equals* filter provides a way to make a simple filtering condition based on a white list / valid list of values. Simply enter a list of values that you accept for a given column, and then you can map your flow to the VALID outcome of the filter.

Here’s an example of an Equals filter configuration where valid Gender values are being checked.

Use the plus/minus buttons to grow or shrink the list of values you want to accept.

If placed as the first component in a flow, the Equals filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.

Max rows

The *Max rows* filter is used to limit the amount of records that are passed further on in the job's flow.

If placed as the first component in a flow, the Max rows filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.
Transform

Not null

The Not null filter is a simple filter that can be used to exclude null values from your flow. Additionally you can select whether or not you want to accept empty strings (""") or not.

If placed as the first component in a flow, the Not null filter is optimizable in a way where it will modify your originating query. This means that it is also an appropriate filter to use if you just want to sample the data used in your job.

Union

The Union transformer allows you to combine multiple streams into one. Providing what is equivalent to a union of tables. Use it to fuse data streams coming from different source tables. You can define new fields whose values represent whatever is available from one of the input streams.

Below is an example of a job with the Union transformation and a composite datastore. The composite datastore contains several data sources, each with customer information. The Union transformer in this example aims to combine tables of these data sources so that a single stream of data can be consumed with records from all the sources.

The configuration of the Union transformation is done by lining up column from the sources that should be combined. In the example shown 'CUSTOMERNUMBER' and 'id' are combined into one new field. Similarly 'CUSTOMERNAME' and 'family_name' is combined and so on.
Chapter 5. Improve

Abstract

The functions in the ‘Improve’ tree are all what we would describe as first class ‘Data Quality functions’. They not only analyze a problem but also usually present a solution to it.

Duplicate detection

The ‘Duplicate detection’ function allows you to do fuzzy matching of duplicate records - records that represent the same person, organization, product or other entity.

The main characteristics of the Duplicate detection function is:

1. **High Quality** - Quality is the hallmark of matching, our duplicate detection feature delivers on this promise.

2. **Scalable** - For large datasets Duplicate detection leverages the Hadoop framework for practically unlimited scalability.

3. **Fast and interactive** - On a single machine you can work quickly and interactively to refine your duplicate detection model.
4. *International* - International data is supported and no regional knowledge has been encoded into the deduplication engine - you provide the business rules externally.

5. *Machine Learning based* - The Duplicate detection engine is configured by examples. During a series of training sessions you can refine the deduplication model simply by having a conversation with the tool about what is and what isn't a good example of a duplicate.

**Tip**

Duplicate detection does work fine with raw data. But if you have dirty data and the way data is registered has a lot of variance, we suggest you first do your best to standardize the data before finding duplicates.

Standardization can be made by trimming, tokenizing, removing unwanted characters, replace synonyms and things like that. Explore the transformations available in DataCleaner in order to get your data cleansed before trying to deduplicate it.

In the following sections we will walk through how to use the 'Duplicate detection' function. The function has three modes: Model training, Detection and Untrained detection.

**'Model training' mode**

In the Model training mode the user of Duplicate detection is looking to train the Machine Learning engine. When running your job in Model training mode you will be shown a number of potential duplicate record pairs, and determine if they are duplicate or not.

To start the Training mode, simply add the function and select the columns that you wish to use for matching. Additionally you might wish to configure:

**Table 5.1. Model training properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max records for training</td>
<td>The Training tool will keep a random sample of the dataset in memory to provide as training input, and an interactive user experience. This number determines how many records will be selected for this sample.</td>
</tr>
<tr>
<td>Key column</td>
<td>If your dataset has a unique key, we encourage you to select it using this property. Configuring the key column has the benefit that if you wish to export a training reference later, it can be re-applied very easily.</td>
</tr>
</tbody>
</table>

In contrast to most other analyzers in DataCleaner which shows a result screen after execution, the Training mode opens a new dialog when started. The training tool dialog allows users to train matching models. The top of the dialog contains a button bar. Below the button bar, the training tool shows some tab buttons. By default the potential duplicates will be shown. For each potential duplicate you can toggle the button on the right side to determine if the pair is a duplicate or not:
To help you, columns with equal values are shown in a grey font, while different values are shown in black.

Right-clicking on the classification button opens a small menu that allows you to mark all examples on the (remainder of) this page or all examples on all pages as Undecided, duplicates or uniques. This helps when almost all examples are duplicates or uniques. You can mark all examples as duplicates, review, and only toggle the examples that are no duplicates.

You do not need to classify all samples shown. Recommended usage is:

1. Classify at least 20-30 duplicate pairs or more (more is better)
2. Classify at least 20-30 unique records or more (more is better)

Once you've classified records you can press the 'Train model' button in the upper right corner. This will refine the matching model and present a new set of interesting potential duplicates. You can continue this way and quite quickly have classified the required amount of pairs.

The model is automatically saved every time after training. There is no need to save the model by hand. The saved model includes the matching rules, settings, and all pairs the user classified as duplicate or unique.

Some more hints for training:

1. Classifying uniques is just as important as classifying duplicates. Keep the numbers of duplicate examples and unique examples roughly equal.
2. Try to find and mark some examples of every duplicate category that you know of. You can use the "search pairs" tool to help you.
3. Sometimes the machine learning gets skewed and does not provide examples of a category of duplicate records or unique records. In those cases, close then re-open the training tool as described below, but do not press the train model button yet. The training tool shows a less specialised set of duplicate samples. You should now be able to find examples of the category you need added to the model.

All duplicate detection models may have irregularities. When you ask a computer to do a complex task like matching, it may come up with a model that has slight differences from your classifications. You can inspect the current model's differences from your classifications in the tab 'Discrepancies'.

Every time you classify a duplicate, it is added to the reference of the Training session. You can inspect your complete reference in the tab 'Duplicates reference'.

If you're looking for particular types of duplicate pair examples, you may want to go to the 'Search pairs' tab. In this tab you will find options to search for records with matching or non-matching values for particular fields. This may be a very useful shortcut for finding proper duplicate examples.

Finally, the tab 'Training parameters' presents buttons and sliders for you to influence the Machine Learning approach.

Moving the top slider to the left makes duplicate detection compare more records. This will take more time, but also increase the matching quality. Moving this slider to the right makes duplicate detection to make less comparisons, resulting in higher speed, but can lead to more missed matches (false negatives).

Moving the bottom slider to the left makes comparison of records more strict. Moving this slider to the right makes it more lax.

The user defined rules enable you to enforce fixed rules. The possible types of fixed rules are listed below. You can apply a rule to each column. Rules that force a pair to be unique take precedence over rules that force a pair to be duplicate. Empty values count as different.

1. forces pairs to be duplicate when equals - The pair is always a duplicate if any column marked with this value is equal.

2. forces pairs to be unique when equals - The pair is never a duplicate if any columns marked with this value are different.

3. forces pairs to be unique when different - The pair is never a duplicate if any columns marked with this value are different

4. forces pairs to be duplicate when equals and unique when different - The pair is never a duplicate if any columns marked with this value is different, but the pairs is always a duplicate if all columns marked with this value are equal.

5. forces pairs to be unique when equals and unique when different - The pair is never a duplicate unless the value in one of the records is empty.

We recommend applying fixed rules only after training the model and only when strictly necessary.

After updating the matching model, the user can continue in 2 ways. If the user is satisfied with the model (few false positives and false negatives) then he can save the model and start using it in duplicate detection. Otherwise, the user can classify more of the presented samples and refine the model again.

More training typically allows for a more advanced matching model, capable of handling more corner cases. The false negatives and false positives lists give a good impression of the current state of the matching model. The user should continue training until the differences in these lists are acceptable.
To validate the training results and obtain the best model, training can be repeated on a different sample. The already classified record pairs will automatically be added to the new sample.

1. Close the training tool.
2. Re-run the Training tool. A new sample will be generated. All marked pairs in the saved reference are automatically included in the new sample.
3. Press the 'Train model' button in the Training tool. This will train a model on the existing reference.
4. You can view the discrepancies (false positives, false negatives) of the trained model against the records in the new sample.
5. You can review the potential duplicates to determine if a category of duplicates is missing.
6. Add more pairs to the reference as needed.

'Detection' mode

When the matching model is complete you are ready to find all the duplicates in the dataset. Use the same Duplicate detection component, but change the execution mode in "Duplicate detection".

When you run the job, you will see a Duplicate detection result with complete groups of duplicates, like this:

Once you have a duplicate detection result that you wish to post-process, e.g. manual inspection, you can export the result by clicking the 'Write duplicates' button in the top of the result screen. You can save the
duplicate records, the duplicate pairs and also the unique records in a datastore table of your choice. Or you can create an excel file or a staging table.

**Tip**

It is now possible to feed the result directly to the merger, using the new data streams feature. You can read more about linking them together in the documentation of merge duplicates.

The Duplicate detection analyzer can run stand-alone to find duplicates in datasets up to a half to 1 million records (depending on the amount of columns). For larger datasets, the Duplicate detection component can be used in combination with an Hadoop server component. This server component is an Enterprise edition feature.

### 'Untrained detection' mode

Finally, there's also a 'Untrained detection' mode. This allows you to skip 'Model training' and just ask the application to do its best effort without any proper model. This mode is not recommended for production use and is considered 'experimental', but may provide a great quick impression of some of the duplicates you have in your dataset.

### Merge duplicates

Merging duplicates is the next step after detecting them in a dataset. It helps to restore a single version of truth by combining information from all the duplicate records representing the same physical entity.

In this section, we assume that the steps from previous section "Duplicate detection" has been completed, and you have a job with the duplicate detection analyzer prepared.

### Merge duplicates

In the Improve menu, in the Deduplication submenu, there are two components that are useful for merging duplicates into a single record - Merge duplicates (simple) and Merge duplicates (advanced). The simple version just picks one record from the group of duplicates while the advanced one enables the user to combine records, taking some values from one record, some from another. In this example we will use the simple version.

Add "Merge duplicates (Simple)" transformer to your job, and configure the new component. Change the scope to "Duplicate Detection: Duplicate and unique rows". Now click "Select all", then disable the four synthetic columns Record id, Group id, Group size and Group score.

Set the property Group id to the column Group id, and the property Group count to the column Group size.
The resulting configuration should look something like this.

Merge duplicates (Simple) transformer will output all the input columns as output + additional columns carrying metadata about the record. One of them is "Merge status" that can have three possible values: SURVIVOR, NON_SURVIVOR and UNIQUE.

1. SURVIVOR records are records that has duplicates, but has been chosen as the representative of their duplicate group, and should therefore "survive".

2. NON_SURVIVOR records are records that has duplicates, but where another record has been chosen as the representative of the duplicate group. These records should not be used.

3. UNIQUE records are records that never had any duplicates.

Let's add an Equals filter to the job in order to write only these SURVIVOR and UNIQUE records, to create a data set with no duplicates.
Save the output of the filter to a new staging table.

**Warning**

Unlike duplicate detection, "Merge duplicates" is a transformer, which means that new columns are created with the result in them. Make sure that you only choose the merged columns in your staging table.

The whole job graph should look similar to this:

**Conclusion**

Following above sections, we obtained a datastore with unique values and merged duplicate inside, ready to be exported to the format of user's preference.

**Synonym lookup**

The Synonym lookup transformation is a critical part of DataCleaner's ability to standardize and cleanse data. Using this component you can look up values in a synonym catalog and have it replaced with its master term, if it is found to be a synonym.

Below is a screenshot of the synonym lookup's configuration panel:
The configuration of the Synonym lookup is simple:

1. Select the column to apply the lookup function to.

2. Use the 'Retain original value' option to determine if unmatched values (non-synonyms) should be retained or if a null value should be returned if there is no match.

3. Select the synonym catalog to use for lookups.

If your synonym catalog contains all the allowed values for a particular column, it can be a good idea to uncheck the 'Retain original value' checkbox and then do a simple null-check on the resulting output column. If null values are found, it's because there are values in the column that the synonym catalog is not able to standardize.

**DE movers and deceased check**

This component provides Address Correction and Mail Suppression services for Germany. Use it to check that the name and address data you have about people is up to date and correct. The following Mail Suppression checks currently exist:

1. Change of Address (with new address) check

2. Moved Away (without new address) check

3. Deceased check
**Tip**

Customers need a set of credentials to access the service.

**Address and Mail Suppression data sources**

The service combines data of several sources, including Deutsche Post.

**Output**

This component outputs name and address fields, in addition to the following indicators:

**Table 5.2. DE movers and deceased check output**

<table>
<thead>
<tr>
<th>Output column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address validation status</td>
<td>The status code of the record. Following values are possible:</td>
</tr>
<tr>
<td></td>
<td>1. Valid</td>
</tr>
<tr>
<td></td>
<td>2. Corrected to valid</td>
</tr>
<tr>
<td></td>
<td>3. Ambiguous</td>
</tr>
<tr>
<td></td>
<td>4. Invalid</td>
</tr>
<tr>
<td></td>
<td>5. Not processed</td>
</tr>
<tr>
<td></td>
<td>6. Failure</td>
</tr>
<tr>
<td></td>
<td>7. Unknown</td>
</tr>
<tr>
<td>Is moved?</td>
<td>A boolean flag indicating if the person has moved to a new address.</td>
</tr>
<tr>
<td>Is moved to unknown address?</td>
<td>A boolean flag indicating if the person has moved away (with no known new address).</td>
</tr>
<tr>
<td>Is deceased?</td>
<td>A boolean flag indicating if the person is deceased.</td>
</tr>
</tbody>
</table>

**Link to...**

The component produces an outputstream that can be fed into another component.
UK movers, deceased and mailing preferences check

This component provides Address Correction and Mail Suppression services for the United Kingdom / Great Britain. Use it to check that the name and address data you have about people is up to date and correct. The following Mail Suppression checks currently exist:

1. Change of Address (with new address) check
2. Moved Away (without new address) check
3. Deceased check
4. Mailing Preference Service (MPS) check

Tip

Customers need a set of credentials to access the service.

Address and Mail Suppression data sources

The service combines data of several sources, including Royal Mail, MPS and the Bereavement registry.

The service allows you to get a Price Quotation including summary statistics about the expected results before accepting it:
Output

This component outputs name and address fields, in addition to the following indicators:

Table 5.3. DE movers and deceased check output

<table>
<thead>
<tr>
<th>Output column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address validation status</td>
<td>A numeric status code of the record. Following value ranges are possible:</td>
</tr>
<tr>
<td></td>
<td>1. 0 - 10: The address is valid.</td>
</tr>
<tr>
<td></td>
<td>2. 11 - 100: The address was corrected.</td>
</tr>
<tr>
<td></td>
<td>3. 101 - 500: The address is invalid.</td>
</tr>
<tr>
<td></td>
<td>4. 700: The address was not processed/skipped.</td>
</tr>
<tr>
<td>AddressCorrectionMessage</td>
<td>A humanly readable message about the address correction outcome.</td>
</tr>
<tr>
<td>IsNCOATracked</td>
<td>A boolean flag indicating if the person has moved to a new address.</td>
</tr>
<tr>
<td>IsNCOAFlagged</td>
<td>A boolean flag indicating if the person has moved away (with no known new address).</td>
</tr>
<tr>
<td>IsDeceased</td>
<td>A boolean flag indicating if the person is deceased.</td>
</tr>
<tr>
<td>IsDoNotMail</td>
<td>A boolean flag indicating if the person does not want to receive unsolicited mail.</td>
</tr>
</tbody>
</table>
US movers, deceased and do-not-mail check

This component provides CASS Certified(tm) Address Correction and Mail Suppression services for the United States of America. Use it to check that the name and address data you have about people is up to date and correct. The following Mail Suppression checks currently exist:

1. Change of Address check
2. Deceased check
3. Do-Not-Mail check

Tip

Customers need a set of credentials to access the service.

Address and Mail Suppression data sources

The service combines data of several sources, including the US Postal Service.

Output

This component outputs name and address fields, in addition to the following indicators:

Table 5.4. DE movers and deceased check output

<table>
<thead>
<tr>
<th>Output column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address validation status</td>
<td>A numeric status code of the record. Following value ranges are possible:</td>
</tr>
<tr>
<td></td>
<td>1. 0 - 10: The address is valid.</td>
</tr>
<tr>
<td></td>
<td>2. 11 - 100: The address was corrected.</td>
</tr>
<tr>
<td></td>
<td>3. 101 - 500: The address is invalid.</td>
</tr>
<tr>
<td></td>
<td>4. 700: The address was not processed / skipped.</td>
</tr>
<tr>
<td>AddressCorrectionMessage</td>
<td>A humanly readable message about the address correction outcome.</td>
</tr>
</tbody>
</table>
### Output column

<table>
<thead>
<tr>
<th><strong>Output column</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>EcoaFootnote</td>
<td>An indicator value telling what the outcome of the 'Change of Address' check was. The following tokens can occur:</td>
</tr>
<tr>
<td></td>
<td>1. <em>N</em> - No change</td>
</tr>
<tr>
<td></td>
<td>2. <em>M</em> - The party has a new address</td>
</tr>
<tr>
<td></td>
<td>3. <em>K</em> - The party has moved away without a new address</td>
</tr>
<tr>
<td></td>
<td>Furthermore the field may have a token representing the type of party that was identified:</td>
</tr>
<tr>
<td></td>
<td>1. <em>I</em> - Individual</td>
</tr>
<tr>
<td></td>
<td>2. <em>F</em> - Family</td>
</tr>
<tr>
<td></td>
<td>3. <em>B</em> - Business</td>
</tr>
<tr>
<td>IsDeceased</td>
<td>A boolean flag indicating if the person is deceased.</td>
</tr>
<tr>
<td>IsDoNotMail</td>
<td>A boolean flag indicating if the person does not want to receive unsolicited mail.</td>
</tr>
</tbody>
</table>

**Link to...**

The component produces an output stream that can be fed into another component.

---

**Table lookup**

The table lookup transformer allows you to look up values in a different table. Any amount of columns can be used for mapping (lookup conditioning) and for outputting (the outcome of the lookup).

The configuration screen for the table lookup transformer looks like this:
To make the mapping you need to select the target datastore, schema and table names. Once selected you will be able to select which columns to use for condition setting when looking up values.

The semantics of the Table lookup are close to the semantics of a LEFT JOIN. If no lookup value is found, nulls will be returned. However, if multiple records are found to match the conditions, only the first will be returned.

Note that the Table lookup will use a cache for looking up values, to avoid querying the target table for every incoming value.
National identifiers

With the 'National identifiers' transformations you can validate and extract useful information from common national identifiers, such as social security numbers, company registrations etc.

There are components available for a number of countries and domains, see the screenshot below:

![Screen shot of national identifiers]

Each transformation will produce a properly formatted variant of the identifier, and a boolean output column which determines if a particular identifier is valid or not. In addition the transformations will produce the following output fields, all depending on the domain and conventions applied within the country:

1. Gender of person
2. Birthdate
3. The gender of the person
4. The date of founding the company
Chapter 6. Analyze

Abstract

This chapter deals with one of the most important concepts in DataCleaner: Analysis of data quality.

An analyzer is a component that consumes a (set of) column(s) and generates an analysis result based on the values in the consumed columns.

Here is an example of a configuration panel pertaining to an analyzer:

In the panel there will always be one or more selections of columns. The configuration panel may also contain additional properties for configuration.
Boolean analyzer

Boolean analyzer is an analyzer targeted at boolean values. For a single boolean column it is quite simple: It will show the distribution of true/false (and optionally null) values in a column. For several columns it will also show the value combinations and the frequencies of the combinations. The combination matrix makes the Boolean analyzer a handy analyzer for use with combinations of matching transformers and other transformers that yield boolean values.

Boolean analyzer has no configuration parameters, except for the input columns.

Completeness analyzer

The completeness analyzer provides a really simple way to check that all required fields in your records have been filled. Think of it like a big "not null" check across multiple fields. In combination with the monitoring application, this analyzer makes it easy to track which records needs additional information.

Here is a screenshot of the configuration panel of the Completeness analyzer:

![Completeness analyzer screenshot]

The configuration properties of the Completeness analyzer are:
Table 6.1. Completeness analyzer properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Select the columns you want to evaluate with your completeness analyzer. For each selected column you get to choose whether the analyzer should simply do a null-check, or if it should also check for blank values.</td>
</tr>
<tr>
<td>Evaluation mode</td>
<td>This determines the mode that the completeness check runs in. Here you can configure whether the analyzer should consider records as &quot;incomplete&quot; if any of the selected values are null/blank, or if all the values need to be null/blank before the record is counted as incomplete.</td>
</tr>
</tbody>
</table>

Character set distribution

The Character set distribution analyzer inspects and maps text characters according to character set affinity, such as Latin, Hebrew, Cyrillic, Chinese and more.

Such analysis is convenient for getting insight into the international aspects of your data. Are you able to read and understand all your data? Will it work in your non-internationalized systems?

Date gap analyzer

The Date gap analyzer is used to identify gaps in recorded time series. This analyzer is useful for example if you have employee time registration systems which record FROM and TO dates. It will allow you to identify if there are unexpected gaps in the data.

Date/time analyzer

The Date/time analyzer provides general purpose profiling metrics for temporal column types such as DATE, TIME and TIMESTAMP columns.

Number analyzer

The number analyzer provides general purpose profiling metrics for numerical column types.

Pattern finder

The pattern finder is one of the more advanced, but also very popular analyzers of DataCleaner.

Here is a screenshot of the configuration panel of the Pattern finder:
From the screenshot we can see that the Pattern finder has these configuration properties:

**Table 6.2. Pattern finder properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group column</td>
<td>Allows you to define a pattern group column. With a pattern group column you can separate the identified patterns into separate buckets/groups. Imagine for example that you want to check if the phone numbers of your customers are consistent. If you have an international customer based, you should then group by a country column to make sure that phone patterns identified are not matched with phone patterns from different countries.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Discriminate text case</td>
<td>Defines whether or not to discriminate (ie. consider as different pattern parts) based on text case. If true &quot;DataCleaner&quot; and &quot;datacleaner&quot; will be considered instances of different patterns, if false they will be matched within same pattern.</td>
</tr>
<tr>
<td>Discriminate negative numbers</td>
<td>When parsing numbers, this property defines if negative numbers should be discriminated from positive numbers.</td>
</tr>
<tr>
<td>Discriminate decimals</td>
<td>When parsing numbers, this property defines if decimal numbers should be discriminated from integers.</td>
</tr>
<tr>
<td>Enable mixed tokens</td>
<td>Defines whether or not to categorize tokens that contain both letters and digits as &quot;mixed&quot;, or alternatively as two separate tokens. Mixed tokens are represented using questionmark ('?') symbols.</td>
</tr>
<tr>
<td></td>
<td>This is one of the more important configuration properties. For example if mixed tokens are enabled (default), all these values will be matched against the same pattern: foo123, 123foo, foobar123, foo123bar. If mixed tokens are NOT enabled only foo123 and foobar123 will be matched (because 123foo and foo123bar represent different combinations of letter and digit tokens).</td>
</tr>
<tr>
<td>Ignore repeated spaces</td>
<td>Defines whether or not to discriminate based on amount of whitespaces.</td>
</tr>
<tr>
<td>Upper case patterns expand in size</td>
<td>Defines whether or not upper case tokens automatically &quot;expand&quot; in size. Expandability refers to whether or not the found patterns will include matches if a candidate has the same type of token, but with a different size. The default configuration for upper case characters is false (ie. ABC is not matched with ABCD).</td>
</tr>
<tr>
<td>Lower case patterns expand in size</td>
<td>Defines whether or not lower case tokens automatically &quot;expand&quot; in size. As with upper case expandability, this property refers to whether or not the found patterns will include matches if a candidate has the same type of token, but with a different size. The default configuration for lower case characters is true (ie. 'abc' is not matched with 'abc').</td>
</tr>
<tr>
<td></td>
<td>The defaults in the two &quot;expandability&quot; configuration properties mean that eg. name pattern recognition is meaningful: 'James' and 'John' both pertain to the same pattern ('Aaaaa'), while 'McDonald' pertain to a different pattern ('AaAaaaAa').</td>
</tr>
<tr>
<td>Predefined token name</td>
<td>Predefined tokens make it possible to define a token to look for and classify using either just a fixed list of</td>
</tr>
</tbody>
</table>
Analyze

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Description values or regular expressions. Typically this is used if the values contain some additional parts which you want to manually define a matching category for. The 'Predefined token name' property defines the name of such a category.</td>
<td></td>
</tr>
<tr>
<td>Predefined token regexes</td>
<td>Defines a number of string values and/or regular expressions which are used to match values against the (pre)defined token category.</td>
</tr>
<tr>
<td>Decimal separator</td>
<td>The decimal separator character, used when parsing numbers</td>
</tr>
<tr>
<td>Thousand separator</td>
<td>The thousand separator character, used when parsing numbers</td>
</tr>
<tr>
<td>Minus sign</td>
<td>The minus sign character, used when parsing numbers</td>
</tr>
</tbody>
</table>

**Reference data matcher**

The 'Reference data matcher' analyzer provides an easy means to match several columns against several dictionaries and/or several string patterns. The result is a matrix of match information for all columns and all matched resources.

**Referential integrity**

With the 'Referential integrity' analyzer you can check that key relationships between records are intact. The analyzer will work with relationships within a single table, between tables and even between tables of different datastores.

Here is a screenshot of the configuration panel of the Referential integrity analyzer:
Apply the analyzer on the table with the foreign key in the relationship, and configure it to do a check on the table that holds all the valid keys.

**Table 6.3. Referential integrity properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache lookups</td>
<td>Whether or not the analyzer should speed up referential integrity checking by caching previous lookup results. Whether or not this will gain performance ultimately depends on the amount of repetition in the keys to be checked. If all foreign key values are more or less unique, it should definitely be turned off. But if there is a fair amount of duplication in the foreign keys (e.g. orderlines referring to the same products or customers), then it makes the lookups faster.</td>
</tr>
<tr>
<td>Ignore null values</td>
<td>Defines whether or not &quot;null&quot; values should be ignored or if they should be considered as an integrity issue. When ignored, all records with null foreign key values will simply be discarded by the analyzer.</td>
</tr>
</tbody>
</table>
String analyzer

The string analyzer provides general purpose profiling metrics for string column types. Of special concern to the string analyzer is the amount of words, characters, special signs, diacritics and other metrics that are vital to understanding what kind of string values occur in the data.

Unique key check

The 'Unique key check' analyzer provides an easy way to verify that keys/IDs are unique - as it is usually expected.

The properties of 'Unique key check' are:

Table 6.4. Unique key check properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Pick the column that this analyzer should perform the uniqueness check on.</td>
</tr>
<tr>
<td>Buffer size</td>
<td>The buffer represents the internal resource for sorting and comparison of keys. Having a large buffer makes the analyzer run faster and take up fewer resources on disk, but at the expense of using memory. If your job is not already memory intensive, we recommend increasing the buffer size up to 1M.</td>
</tr>
</tbody>
</table>

Value distribution

The value distribution (often also referred to as 'Frequency analysis') allows you to identify all the values of a particular column. Furthermore you can investigate which rows pertain to specific values.

Here are the configuration properties for the value distribution analyzer:

Table 6.5. Value distribution properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group column</td>
<td>Allows you to define a column for grouping the result. With a group column you can separate the identified value distributions into separate buckets/groups. Imagine for example that you want to check if the postal codes and city names correspond or if you just want to segment your value distribution on eg. country or gender or ...</td>
</tr>
<tr>
<td>Record unique values</td>
<td>By default all unique values will be included in the result of the value distribution. This can potentially cause memory issues if your analyzed columns contains a LOT of unique values (eg. if it's a unique key). If the actual unique values are not of interest, then uncheck this checkbox to only count (but not save for inspection) the unique values.</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top n most frequent values</td>
<td>An optional number used if the analysis should only display eg. the &quot;top 5 most frequent values&quot;. The result of the analysis will only contain top/bottom n most frequent values, if this property is supplied.</td>
</tr>
<tr>
<td>Bottom n most frequent values</td>
<td>An optional number used if the analysis should only display eg. the &quot;bottom 5 most frequent values&quot;. The result of the analysis will only contain top/bottom n most frequent values, if this property is supplied.</td>
</tr>
</tbody>
</table>

### Value matcher

The value matcher works very similar to the Value distribution, except for the fact that it takes a list of expected values and everything else is put into a group of 'unexpected values'. This division of values means a couple of things:

1. You get a built-in validation mechanism. You expect maybe only 'M' and 'F' values for your 'gender' column, and everything else is in a sense invalid, since it is unexpected.

2. The division makes it easier to monitor specific values in the data quality monitoring web application.

3. This analyzer scales much better for large datasets, since the groupings are deterministic and thus can be prepared for in the batch run.

### Weekday distribution

The weekday distribution provides a frequency analysis for date columns, where you can easily identify which weekdays a date field represents.
Chapter 7. Write

Abstract

Although the primary focus of DataCleaner is analysis, often during such analysis you will find yourself actually improving data. When this is the case, obviously you will want to export the improved/cleansed data so you can utilize it in other situations than the analysis. In this chapter we will look at the various writing options that DataCleaner provide.

In the following sections each output format option will be described:

Create CSV file

Writes a data set to a Comma Separated Values file. CSV files are a popular choice for interoperability with other systems and loading of data into databases.

Create Excel spreadsheet

Writes a data set to an Excel spreadsheet. An advantage of this approach is that a single file can contain multiple sheets, and that it is easily navigable in Microsoft Excel. A disadvantage is that for very large data sets it is less performant.

Create staging table

Writes a data set to an embedded relational database, which DataCleaner manages. This option is primarily used for staging data for further analysis. The advantage of using the feature is that it retains column type information, it can handle a lot of data and multiple data sets can be written to the same datastore. A disadvantage is that the data is not easily readable by third party applications (unless exported again).
Insert into table

Using this writer you can insert your data into a table of an existing datastore. If you already have a table layout ready or if you want to append to e.g. a database table, then this writing option is the right one for you.

Optionally, you can make the 'Insert into table' component truncate your table before insertion. This will delete all existing records in the table, useful for initial load situations.

Currently target tables can be from any of the following datastore types:

1. CSV file. In this case data will be appended to the file.
2. Excel spreadsheet. In this case data will be appended to the file.
3. Relational database. In this case data will be inserted to the table using an INSERT statement.
4. *MongoDB database*. In this case data will be inserted into the MongoDB collection.

5. *CouchDB database*. In this case data will be inserted into the CouchDB database.

6. *Salesforce.com*. In this case data will be uploaded/inserted into Salesforce.com using the SOQL web services.

7. *ElasticSearch index*. In this case data will be indexed into ElasticSearch.

**Update table**

The 'Update table' writer works like the 'Insert into table' writer except that it issues UPDATE statements instead of INSERT statements. This obviously means that it has an additional property, used to specify the condition (the WHERE part) of the update.
Part III. Reference data
Table of Contents

8. Dictionaries ............................................................................................................. 64
9. Synonyms (aka. Synonym catalogs) ........................................................................ 65
    Text file synonym catalog ...................................................................................... 65
    Datastore synonym catalog .................................................................................... 65
10. String patterns ...................................................................................................... 66
Chapter 8. Dictionaries

Dictionaries are reference data lists used for verifying or categorizing values against certain black- or whitelists. Dictionaries are generally enumerable and finite, whereas eg. string patterns are dynamic and evaluated each time.

Examples of meaningful dictionaries are:

1. A dictionary of product types like "jewelry", "menswear", "sportswear" etc.
2. A dictionary of gender symbols like "M", "F" and maybe "UNKNOWN".
3. A dictionary of age group names (eg. infant, child, young, mature, senior)
4. Two dictionaries for male and female given names (in order to determine gender of persons)
Chapter 9. Synonyms (aka. Synonym catalogs)

Abstract

Synonym catalogs are used to replace and standardize values to their master terms, in order to avoid multiple terms for the same real world thing.

There are many real life examples of synonyms that make for messy data, for example:

1. Company and brand names, like "Coca-Cola", "Coca cola" and "Coke".
2. Titles, like "Doctor", "Dr." and "Doc"

In the following sections we will describe how to set up synonym catalogs that can be used in a variety of ways to standardize your database.

Text file synonym catalog

A text file synonym catalog is the easiest and often also the fastest way to perform synonym replacement. Simply create a text file with content in a format, where the master term is succeeded with a comma-separated list of synonyms, like this:

M,Male,Man,Guy,Boy
F,Female,Woman,Girl

In the above example, most typical gender tokens will be replaced with either "M" or "F".

Datastore synonym catalog

If your synonyms are located in a database or another type of datastore, then you can also create synonym catalogs based on this.

Datastore synonym catalogs allow you to specify a single master term column and multiple synonym columns. The synonym catalog will look then find synonym matches by searching/querying the datastore.
Chapter 10. String patterns

String patterns define a "template" for string values which they may or may not conform to.

DataCleaner currently supports two type of popular string formats:

1. Regular expressions, which is a general purpose string pattern matching language popular in computer science. Regular expressions does take a bit of time to learn, but are very powerful once harnessed.

   Explaining the syntax of regular expressions is definitely outside the scope of the DataCleaner documentation. We recommend the Java Regular Expressions lesson [http://docs.oracle.com/javase/tutorial/essential/regex/] if you are looking for a resource on this.

2. Simple string patterns, which use the same syntax as the Pattern finder analyzer. Patterns such as "aaaa@aaaa.aaa" could for example be used to match typical email addresses.
Part IV. Configuration reference
# Table of Contents

11. Configuration file ........................................................................................................... 69  
   XML schema ....................................................................................................................... 69  
   Datastores .......................................................................................................................... 69  
   Database (JDBC) connections ......................................................................................... 69  
   Comma-Separated Values (CSV) files .............................................................................. 70  
   Fixed width value files ..................................................................................................... 70  
   Excel spreadsheets .......................................................................................................... 71  
   XML file datastores .......................................................................................................... 71  
   ElasticSearch index ......................................................................................................... 72  
   MongoDB databases ........................................................................................................ 72  
   CouchDB databases ......................................................................................................... 73  
   Composite datastore ........................................................................................................ 74  
   Reference data ................................................................................................................ 74  
   Dictionaries ...................................................................................................................... 74  
   Synonym catalogs ........................................................................................................... 75  
   String patterns ................................................................................................................ 76  
   Task runner ...................................................................................................................... 77  
   Storage provider .............................................................................................................. 77  
12. Analysis job files ............................................................................................................ 79  
   XML schema ..................................................................................................................... 79  
   Source section .................................................................................................................. 79  
13. Logging .......................................................................................................................... 81  
   Logging configuration file ............................................................................................... 81  
   Default logging configuration ......................................................................................... 81  
   Modifying logging levels ................................................................................................. 82  
   Alternative logging outputs ............................................................................................. 82  
14. Database drivers ............................................................................................................ 84  
   Installing Database drivers in DataCleaner desktop ....................................................... 84  
   Installing Database drivers in DataCleaner monitor ....................................................... 86
Chapter 11. Configuration file

Abstract
In this chapter we go through the elements of a configuration file, conf.xml, making it possible (although optional) to change the static configuration and configure the environment of DataCleaner. The configuration file and related files are stored by convention in a folder called .datacleaner within your user's home directory.

In the DataCleaner monitoring web application, the conf.xml file is the only point of configuration. The file is located in the root of each tenant's repository folder. For more information, refer to the repository chapter.

Most of the elements in the configuration file is also editable within the Desktop application. It is however important to note that changes made in the GUI are not saved directly to the configuration file, but to the userpreferences.dat file. You can consider the relationship between the two files this way: The configuration file defines a static, unmodifiable prototype of the applications environment. All customizations made to this prototype in the Desktop application is saved in the userpreferences.dat file.

XML schema
The configuration file (conf.xml) is an XML file pertaining to the XML namespace "http://eobjects.org/analyzerbeans/configuration/1.0".

For XML-savvy readers, who prefer to use XML schema aware editors to edit their XML files, you can find the XML schema for this namespace here: https://github.com/datacleaner/DataCleaner/blob/master/engine/xml-config/src/main/resources/configuration.xsd.

Datastores
Datastores can be configured in the configuration file under the element <datastore-catalog>. The following sections will go into further details with particular types of datastores.

Database (JDBC) connections
Here are a few examples of common database types.

Tip
The DataCleaner User Interface makes it a lot easier to figure out the url (connection string) and driver class part of the connection properties. It's a good place to start if you don't know these properties already.

MySQL

<jdbc-datastore name="MySQL datastore">
  <driver>com.mysql.jdbc.Driver</driver>
  <username>username</username>
  <password>password</password>
  <multiple-connections>true</multiple-connections>
</jdbc-datastore>
Oracle

<jdbc-datastore name="Oracle datastore">
  <url>jdbc:oracle:thin:@hostname:1521:sid</url>
  <driver>oracle.jdbc.OracleDriver</driver>
  <username>username</username>
  <password>password</password>
  <multiple-connections>true</multiple-connections>
</jdbc-datastore>

Microsoft SQL Server

A typical connection to Microsoft SQL server will look like this:

<jdbc-datastore name="MS SQL Server datastore">
  <url>jdbc:jtds:sqlserver://hostname/database;useUnicode=true;characterEncoding=UTF-8</url>
  <driver>net.sourceforge.jtds.jdbc.Driver</driver>
  <username>username</username>
  <password>password</password>
  <multiple-connections>true</multiple-connections>
</jdbc-datastore>

However, if you want to use an instance name based connection, then the SQL Server Browser service MUST BE RUNNING and then you can include the instance parameter: Here’s an example for connecting to a SQLEXPRESS instance:

<url>jdbc:jtds:sqlserver://hostname/database;instance=SQLEXPRESS;useUnicode=true;characterEncoding=UTF-8</url>

Comma-Separated Values (CSV) files

This is an example of a CSV file datastore

<csv-datastore name="my_csv_file">
  <filename>/path/to/file.csv</filename>
  <quote-char>"</quote-char>
  <separator-char>;</separator-char>
  <encoding>UTF-8</encoding>
  <fail-on-inconsistencies>true</fail-on-inconsistencies>
  <header-line-number>1</header-line-number>
</csv-datastore>

Fixed width value files

Files with fixed width values can be registered in two ways - either with a single fixed-width size for all columns, or with individual value-widths.

Here's an example with a fixed width specification for all columns:

<fixed-width-datastore name="My file">
Configuration file

<filename>foobar.txt</filename>
<width-specification>
  <fixed-value-width>20</fixed-value-width>
</width-specification>
<encoding>UTF-8</encoding>
<fail-on-inconsistencies>true</fail-on-inconsistencies>
</fixed-width-datastore>

Here's an example with individual (3 columns) width specifications:

<fixed-width-datastore name="My file">
  <filename>foobar.txt</filename>
  <width-specification>
    <value-width>4</value-width>
    <value-width>17</value-width>
    <value-width>19</value-width>
  </width-specification>
  <encoding>UTF-8</encoding>
  <fail-on-inconsistencies>true</fail-on-inconsistencies>
</fixed-width-datastore>

Excel spreadsheets

This is an example of an Excel spreadsheet datastore

<excel-datastore name="my_excel_spreadsheet">
  <filename>/path/to/file.xls</filename>
</excel-datastore>

XML file datastores

Defining XML datastores can be done in both a simple (automatically mapped) way, or an advanced (and more performant and memory effective way).

The simple way is just to define a xml-datastore with a filename, like this:

<xml-datastore name="my_xml_datastore">
  <filename>/path/to/file.xml</filename>
</xml-datastore>

This kind of XML datastore works find when the file size is small and the hierarchy is not too complex. The downside to it is that it tries to automatically detect a table structure that is fitting to represent the XML contents (which is a tree structure, not really a table).

To get around this problem you can also define your own table structure in which you specify the XPaths that make up your rows and the values within your rows. Here's an example:

<xml-datastore name="my_xml_datastore">
  <filename>/path/to/file.xml</filename>
  <table-def>
    <rowXpath>/greetings/greeting</rowXpath>
  </table-def>
</xml-datastore>
The datastore defines a single table, where each record is defined as the element which matches the XPath "/greetings/greeting". The table has two columns, which are represented by the "how" and "what" elements that are child elements to the row's path.

For more details on the XPath expressions that define the table model of XML datastores, please refer to MetaModel's tutorial on the topic [http://metamodel.eobjects.org/example_xml_mapping.html] (MetaModel is the data access library used to read data in DataCleaner).

**ElasticSearch index**

This is an example of an ElasticSearch index datastore

```
<elasticsearch-datastore name="my_elastic_search_index">
  <hostname=localhost</hostname>
  <port>9300</port>
  <cluster-name>my_es_cluster</cluster-name>
  <index-name>my_index</index-name>
</elasticsearch-datastore>
```

**MongoDB databases**

This is an example of a fully specified MongoDB datastore, with an example table structure based on two collections.

```
<mongodb-datastore name="my_mongodb_datastore">
  <hostname=localhost</hostname>
  <port>27017</port>
  <database-name>my_database</database-name>
  <username>user</username>
  <password>pass</password>
  <table-def>
    <collection>company_collection</collection>
    <property>
      <name>company_name</name>
      <type>VARCHAR</type>
    </property>
    <property>
      <name>customer</name>
      <type>BOOLEAN</type>
    </property>
    <property>
      <name>num_employees</name>
      <type>INTEGER</type>
    </property>
    <property>
      <name>address_details</name>
      <type>MAP</type>
    </property>
```
If the hostname and port elements are left out, localhost:27017 will be assumed.

If the username and password elements are left out, an anonymous connection will be made.

If there are no table-def elements, the database will be inspected and table definitions will be auto-detected based on the first 1000 documents of each collection.

**CouchDB databases**

This is an example of a fully specified CouchDB datastore, with an example table structure based on two CouchDB databases.

```xml
<couchdb-datastore name="my_couchdb_datastore">
    <hostname>localhost</hostname>
    <port>5984</port>
    <username>user</username>
    <password>pass</password>
    <ssl>true</ssl>
    <table-def>
        <database>company_collection</database>
        <field>
            <name>company_name</name>
            <type>VARCHAR</type>
        </field>
        <field>
            <name>customer</name>
            <type>BOOLEAN</type>
        </field>
        <field>
            <name>num_employees</name>
            <type>INTEGER</type>
        </field>
        <field>
            <name>address_details</name>
```
<type>MAP</type>
</field>
</table-def>
<table-def>
<database>person_collection</database>
<field>
<name>person_name</name>
<type>VARCHAR</type>
</field>
<field>
<name>birthdate</name>
<type>DATE</type>
</field>
<field>
<name>emails</name>
<type>LIST</type>
</field>
</table-def>
</couchdb-datastore>

If the hostname and port elements are left out, localhost:5984 will be assumed.

If the username and password elements are left out, an anonymous connection will be made.

If the "ssl" element is false or left out, a regular HTTP connection will be used.

If there are no table-def elements, the database will be inspected and table definitions will be auto-detected based on the first 1000 documents of each database.

**Composite datastore**

This is an example of a composite datastore. It contains data from 2 other datastores: Datastore 1 and Datastore 2.

```xml
<composite-datastore name="my composite">
<datastore-name>Datastore 1</datastore-name>
<datastore-name>Datastore 2</datastore-name>
</composite-datastore>
```

**Reference data**

Reference data items (dictionaries, synonym catalogs and string patterns) are defined in the configuration file in the element <reference-data-catalog>. Below some examples:

**Dictionaries**

Dictionaries are stored within the <dictionaries> element within the reference data section. Three types of dictionaries can be added.

*Datastore dictionaries*
<reference-data-catalog>
   <dictionaries>
      ... 
      <datastore-dictionary name="Lastnames" description="My datastore based dictionary">
         <datastore-name>orderdb</datastore-name>
         <column-path>EMPLOYEES.LASTNAME</column-path>
      </datastore-dictionary>
      ...
   </dictionaries>
</reference-data-catalog>

**Text file dictionaries**

<reference-data-catalog>
   <dictionaries>
      ...
      <text-file-dictionary name="Firstnames" description="My file based dictionary">
         <filename>/path/to/first.txt</filename>
         <encoding>UTF-8</encoding>
      </text-file-dictionary>
      ...
   </dictionaries>
</reference-data-catalog>

**Value list dictionaries**

<reference-data-catalog>
   <dictionaries>
      ...
      <value-list-dictionary name="Greetings" description="My simple value list">
         <value>hello</value>
         <value>hi</value>
         <value>greetings</value>
         <value>godday</value>
      </value-list-dictionary>
      ...
   </dictionaries>
</reference-data-catalog>

**Synonym catalogs**

Synonym catalogs are stored within the <synonym-catalogs> element within the reference data section. Two types of dictionaries can be added.

**Text file synonym catalogs**

<reference-data-catalog>
   <synonym-catalogs>
      ...
   </synonym-catalogs>
</reference-data-catalog>
Configuration file

Datastore synonym catalogs

String patterns

Dictionaries are stored within the <string-patterns> element within the reference data section. Two types of string patterns can be added.

Regular expression (regex) string patterns

Simple string patterns
Task runner

The task runner defines how DataCleaner's engine will execute the tasks of an analysis job. Typically you shouldn't edit this element. However, here are the two options:

```xml
<multithreaded-taskrunner max-threads="30" />
```

Defines a multi threaded task runner with a thread pool of 30 available threads. Beware that although 30 might seem like a high number that too small a pool of threads might cause issues because some tasks schedule additional tasks and thus there's a risk of dead lock when thread count is very low.

```xml
<singlethreaded-taskrunner />
```

Defines a single threaded task runner. On legacy hardware or operating systems this setting will be better, but it will not take advantage of the multi threading capabilities of modern architecture.

Storage provider

The storage provider is used for storing temporary data used while executing an analysis job. There are two types of storage: Large collections of (single) values and "annotated rows", ie. rows that have been sampled or marked with a specific category which will be of interest to the user to inspect.

To explain the storage provider configuration let's look at the default element:

```xml
<storage-provider>
  <combined>
    <collections-storage>
      <berkeley-db/>
    </collections-storage>
    <row-annotation-storage>
      <in-memory max-rows-threshold="1000" max-sets-threshold="200"/>
    </row-annotation-storage>
  </combined>
</storage-provider>
```

The element defines a combined storage strategy.

Collections are stored using berkeley-db, an embedded database by Oracle. This is the recommended strategy for collections.

Row annotations are stored in memory. There's a threshold of 1000 rows in maximum 200 sets. This means that if more than 1000 records are annotated with the same category then additional records will not be saved (and thus is not viewable by the user). Furthermore it means that only up until 200 sample sets will be saved. Further annotations will not be sampled, but metrics still be counted. Most user scenarios will not require more than max. 1000 annotated records for inspection, but if this is really necessary a different strategy can be pursued:
Using MongoDB for annotated rows

If you have a local MongoDB [http://www.mongodb.org/] instance, you can use this as a store for annotated rows. This is how the configuration looks like:

```
<row-annotation-storage>
    <custom-storage-provider class-name="org.datacleaner.storage.MongoDbStorageProvider"/>
</row-annotation-storage>
```

The MongoDB storage provider solution has shown very good performance metrics, but does add more complexity to the installation, which is why it is still considered experimental and only for savvy users.
Chapter 12. Analysis job files

Abstract

Job files contain the information about the execution of a DataCleaner job. Typically these files have the file extension `.analysis.xml`. In this file we will explain the file format, which is XML based, and explain how it relates to what DataCleaner does.

A job will always reference items in a configuration, such as datastores, reference data and more. Therefore a job alone is not enough to execute. But multiple jobs can use the same configuration. For more information on the configuration, see the configuration file chapter.

XML schema

Analysis job files are written in an XML format pertaining to the XML namespace "http://eobjects.org/analyzerbeans/job/1.0".


Read on in this chapter for notes on individual parts of the job file format.

Source section

The source section of the job file format is probably the most interesting one to manually edit or review. Here's an example source section:

```xml
<source>
  <data-context ref="orderdb" />
  <columns>
    <column path="PUBLIC.EMPLOYEES.EMPLOYEENUMBER" id="col_0" type="INTEGER" />
    <column path="PUBLIC.EMPLOYEES.LASTNAME" id="col_1" type="VARCHAR" />
    <column path="PUBLIC.EMPLOYEES.FIRSTNAME" id="col_2" type="VARCHAR" />
    <column path="PUBLIC.EMPLOYEES.EXTENSION" id="col_3" type="VARCHAR" />
    <column path="PUBLIC.EMPLOYEES.EMAIL" id="col_4" type="VARCHAR" />
    <column path="PUBLIC.EMPLOYEES.OFFICECODE" id="col_5" type="VARCHAR" />
    <column path="PUBLIC.EMPLOYEES.REPORTSTO" id="col_6" type="INTEGER" />
    <column path="PUBLIC.EMPLOYEES.JOBTITLE" id="col_7" type="VARCHAR" />
  </columns>
  <variables>
    <variable id="employee type" value="Sales Rep" />
  </variables>
</source>
```

From this source section we can derive these interesting facts:

1. The job is using the datastore called 'orderdb'. How this datastore is configured, the job is not aware of, since it is defined in the configuration. Potentially the job could be used with multiple similar datastores, as long as their name was 'orderdb'.

79
2. The columns defined make up the base of the source query that the job will fire. Each column is assigned an artificial ID, and a hint about its data type is provided. This information is there to be able to detach or replace a column with a new definition. That means that if you've spent a long time building the perfect job, but want to apply it to a different column, you can potentially "just" change the column definition here and retain the original column ID.

3. In this source section we also see some variables. This is an optional sub-section and not that common. The variables are property values that can be replaced at runtime with new values. See the chapter Parameterizable jobs for more information and examples.
Chapter 13. Logging

Abstract

Logging in DataCleaner is configurable either by supplying an XML file or a properties file. In this chapter we explore logging configuration and how you can fine-tune logging to your needs.

Logging configuration file

Logging in DataCleaner is based on Log4j, an open source logging framework by the Apache foundation. With Log4j you can configure logging at a very detailed level, while at the same time keeping a centralized configuration.

There are three approaches to configuring logging in DataCleaner:

1. The default logging configuration. This requires no changes to the standard distribution of DataCleaner. Log files will be generated in the log/datacleaner.log file.

2. Specifying your own XML log configuration. This requires you to put a file named log4j.xml in the root directory of DataCleaner.

3. Specifying your own property file log configuration. This requires you to put a file named log4j.properties in the root directory of DataCleaner.

The recommended way of doing custom configuration of DataCleaner logging is using the XML format. In the following sections we will explain this approach using examples. For more detailed documentation on Log4j configuration, please refer to the Log4j website [http://logging.apache.org/log4j/].

Default logging configuration

Here’s a listing of the default logging configuration, in XML format:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
  <appender name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%-5p %d{HH:mm:ss} %c{1} - %m%n"/>
    </layout>
    <filter class="org.apache.log4j.varia.LevelRangeFilter">
      <param name="levelMin" value="WARN"/>
    </filter>
  </appender>
  <appender name="fileAppender" class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="${user.home}/.datacleaner/log/datacleaner.log"/>
    <param name="DatePattern" value=".'yyyy-MM-dd'.log'"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%-5p %d{HH:mm:ss.SSS} %c{1} - %m%n"/>
    </layout>
  </appender>
</log4j:configuration>
```
This logging configuration specifies the INFO level as the default level of logging. It appends (outputs) log messages to the console (if available) and to a file with the path: `${user.home}.datacleaner/log/datacleaner.log`.

We recommend using this default configuration as a template for custom log configurations. Next we will explore how to modify the configuration and create new logging outputs.

**Modifying logging levels**

These are the logging levels available in DataCleaner and Log4j, order by priority (highest priority first):

1. error
2. warn
3. info
4. debug
5. trace

Typically the bottom-two logging levels (debug and trace) are not used unless unexpected situations has to be investigated by developers.

Modifying the logging levels can be done either globally or in a hierarchical manner:

1. If you change the `<priority>` element's value attribute, you change the global threshold for logging messages.
2. If you change the `<logger>` element's level, you change the logging priority logging messages that pertain to a particular hierarchy of loggers.

**Alternative logging outputs**

Log messages are printed to an output, typically a file or the console. In the configuration file this is configured in the `<appender>` elements. Here's a few examples of alternative appenders you can use. For more examples and documentation, please refer to the Log4j website [http://logging.apache.org/log4j/].
Logging in a PostgreSQL database:

```xml
<appender name="jdbcAppender" class="org.apache.log4j.jdbc.JDBCAppender">
  <param name="URL" value="jdbc:postgresql:db"/>
  <param name="Driver" value="org.postgresql.Driver"/>
  <param name="User" value="user"/>
  <param name="Password" value="password"/>
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="INSERT INTO log4j (log_date,log_level,log_location,log_message) VALUES ('%d{yyyy-MM-dd}','%p','%C;%L','%m')"/>
  </layout>
</appender>
```
Chapter 14. Database drivers

Abstract

DataCleaner ships with a set of standard database drivers, enabling you to connect to common databases such as PostgreSQL, MySQL, Microsoft SQL Server or Oracle.

It's also not uncommon to wish to install additional drivers for other database brands. DataCleaner uses the JDBC standard for managing database drivers and connection. In this chapter we cover the process of installing additional database drivers.

Installing Database drivers in DataCleaner desktop

Installing database drivers in DataCleaner desktop is done in the application itself while it is running.

First, go to the Options menu:

![Options menu](image)

Then select the 'Database drivers' tab. In this tab you will see a listing of your currently installed database drivers (that we know of):
If you click the 'Add database driver' button and then select the 'Local JAR file(s)' option, you will see this dialog:
In this dialog, select driver class name and the files containing the database driver. If you don't know what this is please refer to your database vendor's documentation on JDBC database drivers.

In the example above we see the IBM DB2 driver which involves installing two files since the software license also has to be included.

**Installing Database drivers in DataCleaner monitor**

If you want to install database drivers in the DataCleaner monitor web application, you should first consider if you want the driver installed only as part of the single web application or if it should be globally registered within your web server/container.

1. **Installing database driver in Container**: Refer to your container's documentation. Usually the process is a simple matter of copying the driver JAR files to a folder in the web container's workspace. For instance, if you're using Apache Tomcat, copy the driver JAR file(s) to the 'lib' folder of Tomcat.

2. **Installing database driver in the web application**: First, locate where the DataCleaner monitor .war file is deployed and 'exploded' into a directory. Within this directory you will find the folder 'WEB-INF/lib'. Copy the database driver file(s) into this directory.

We recommend installing drivers in the container. This makes your web application easier to upgrade and manage.
Part V. DataCleaner monitor repository
Table of Contents

15. Repository configuration ................................................................. 89
   Configure repository location .......................................................... 89
   Directory-based repository .............................................................. 89
   Database-backed repository ............................................................ 89
   Providing signed Java WebStart client files ......................................... 90
   Producing the signed JARs ................................................................. 90
   Configuring DataCleaner monitor to use the signed JARs ....................... 91
   Cluster configuration (distributed execution) ....................................... 91

16. Repository layout .............................................................................. 93
   Multi-tenant layout ............................................................................ 93
   Tenant home layout ............................................................................ 93
Chapter 15. Repository configuration

Abstract

In this chapter we will explain configuration of the repository of the DataCleaner monitor web application. By default the repository and other artifacts are bundled with the application, but for production deployments this configuration may not be sufficient. Learn about how to deploy a repository that is located independently of the web application code.

Configure repository location

By default DataCleaner monitor web application uses a file-based repository location which is relative to the deployed web archive. This makes it easy to deploy and test-drive, but it might not be the best production deployment choice.

Directory-based repository

To change the repository location, create or find the file $USER\HOME/.datacleaner/datacleaner-monitor.properties. Make sure that it contains a key 'repository.file.location' and set it's value to the location you wish. For instance:

repository.file.location=/var/datacleaner/repository

The repository directory approach is recommended for single-machine instances of DataCleaner monitor. If you have a cluster of DataCleaner servers please refer to the database-backed repository configuration below.

Database-backed repository

The database-backed repository approach makes it possible for multiple server instances to share the same repository. To enable it, follow these instructions.

In the configuration file $USER\HOME/.datacleaner/datacleaner-monitor.properties you should first configure the following keys with the database's JDBC connection details (url, driver, username and password respectively):

backend.datastore.jdbc.url=
backend.datastore.jdbc.driver=
backend.datastore.username=
backend.datastore.password=

Now go to the location where DataCleaner-monitor is installed and it's .war file is exploded (in Apache Tomcat this would be the 'webapps' folder of Tomcat). Locate the file: DataCleaner-monitor/WEB-INF/
Repository configuration

(classes/context/repository-context.xml). In this file you will see two bean definitions with id="repository". One of them is active (the file based) by default, and one is commented (the database variant).

Change it so that the file-based repository bean is commented and the database-backed repository is active. Also consider the two values "PUBLIC" and "BLOB". These should be changed if necessary to the schema name of your database and the data type name of BLOBs/bytes of your particular database (usually “BLOB” or “bytea”). You should end up with an element similar to this:

```xml
<bean id="repository" class="com.hi.datacleaner.repository.DatastoreRepository">
  <constructor-arg name="datastore" ref="backendDatastore"/>
  <constructor-arg name="schemaName" value="PUBLIC"/>
  <constructor-arg name="blobNativeType" value="BLOB"/>
</bean>
```

Providing signed Java WebStart client files

The DataCleaner monitor web application features an option to let the user launch the desktop application for editing and testing jobs deployed on the monitor server. To enable this special mode of interoperability, signed JAR files needs to be provided, since otherwise the desktop application will not be allowed to launch by most Java runtime configurations.

Producing the signed JARs

By default DataCleaner monitor will use a public set of DataCleaner JAR files to launch the Java Web Start client (when clicking on "Analyze" buttons on the 'Scheduling' and 'Datastores' pages).

If your server does not have internet access, you can provide your own set of JAR files for the DataCleaner Java Web Start client. The procedure involves signing these JAR files, which may depend on the availability of a certificate. If no certificate is available, you can also self-sign the JARs, but that will cause a warning to pop up before launching the client.

In-depth information about JAR file signing can be found on Oracle's website: http://docs.oracle.com/javase/tutorial/deployment/jar/signing.html

For the purposes of this installation, you will need to create a Java keystore alias, optionally based upon a certificate. If this is new to you, and you just want to have use simple self-signed JAR files, issue the following command:

```
keytool -keystore "%JAVA_HOME%/jre/lib/security/cacerts" -genkey -alias my_alias
```

You will be prompted for the keystore password, which is by default "changeit".

After running the command, it will eventually produce an alias in your keystore.

After installing DataCleaner, the installation folder will include either of these executables, depending on operating system:

```
sign-jar-files.cmd (on windows systems)
sign-jar-files.sh (on unix systems)
```
Open the appropriate executable. It will have a top section like this (example from .cmd file. The .sh file does not have the 'set' keyword):

```bash
set ALIAS=my_alias
set KEYPASS=my_password
set SIGFILE=
```

Set the KEYPASS variable to the alias password.
Set the ALIAS variable to the alias name.
Optionally set the SIGFILE variable to the name of your signature.
Run the sign-jar-files executable. A folder named signed_jars will be produced in the installation directory.

**Configuring DataCleaner monitor to use the signed JARs**

In the `${user.home}/.datacleaner/datacleaner-monitor.properties` file you can define the `jnlp.artifacts.location` property, which should point to the directory of your signed JAR files. This is the default definition:

```bash
datacleaner-monitor.properties
jnlp.artifacts.location=${user.home}/.datacleaner/jnlp-launch-artifacts
```

Restart the DataCleaner monitor after configuring this property.

**Cluster configuration (distributed execution)**

DataCleaner monitor allows execution of jobs through a cluster of machines - essentially to increase fault tolerance and performance by adding more machines instead of having to upgrade the hardware of a single machine.

When executing distributed jobs, DataCleaner will initially estimate how many records needs to be processed. Depending on this number, a number of "chunks" of records will be assigned to be executed on different slave execution nodes. After execution, the master node will collect the slave node results and combine them into a single result report.

The configuration of DataCleaner's cluster is handled through the file `WEB-INF/classes/context/cluster-context.xml` within the deployed web archive folder. By default it defines this `<bean>` element:

```xml
<bean id="clusterManagerFactory" class="org.datacleaner.monitor.cluster.HttpClusterManagerFactory">
  <property name="username" value="admin" />
  <property name="password" value="admin" />
  <property name="slaveServerUrls">
    <list>
      <value>http://localhost:8080/DataCleaner-monitor</value>
      <value>http://localhost:9090/DataCleaner-monitor</value>
    </list>
  </property>
</bean>
```
Repository configuration

The above definition states that the cluster has two slave execution nodes. As an example, these are using 'localhost' references, but you can also use other hostnames.

To enable clustered execution of a job, you need to open it's .schedule.xml file in the 'jobs' folder of the repository. In this XML file you will find a <distributed-execution> element which determines if local or distributed execution will be executed. For example, the file 'Customer completeness.schedule.xml' starts like this:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<schedule
xmlns="http://eobjects.org/datacleaner/schedule/1.0"
xmlns:ns2="http://eobjects.org/datacleaner/shared/1.0"
xmlns:ns3="http://eobjects.org/datacleaner/timeline/1.0"
xmlns:ns4="http://eobjects.org/datacleaner/execution-log/1.0">
  <cron-expression>@daily</cron-expression>
  <distributed-execution>false</distributed-execution>
  <alerts>
  ...
  </alerts>
</schedule>
```

Changing this value to 'true' would trigger DataCleaner monitor to use the cluster configuration when executing the job.

Tip

The enterprise edition of DataCleaner also include other mechanisms of communication between cluster nodes. One short-coming of the above approach is that it is not tolerant to network issues or crashing nodes. Consider DataCleaner enterprise edition for such deployments, since it supports elastic clusters without having the master to be aware of each single node.
Chapter 16. Repository layout

Abstract

In this chapter we look at the file and folder layout of a DataCleaner monitor repository. Beginning with the multi-tenant layout, and then proceeding with a typical tenant's repository layout.

Multi-tenant layout

The DataCleaner repository layout, used by the monitoring web application, is built to support multi-tenant deployments. Therefore, on the root level of the repository, folders are located which each represent a tenant's separate home folder. Users from one tenant are not able to access files or folders from other tenant's home folders.

By default the server is configured as a single-tenant instance. In the Enterprise Edition of DataCleaner this can be reconfigured in the following ways:

1. Tenant per user: This provides a completely separated workspace for each user of the monitor server.
2. Manual user groups: A configuration where the tenants are manually configured using a list of usernames.
3. Dynamic tenant management: For customers we can provide dynamic tenant management modules that advice with e.g. LDAP systems to determine the tenant of any particular user.

Tenant home layout

To function properly, each tenant home folder requires these files and folders:

1. conf.xml (file)
2. jobs (folder)
3. results (folder)
4. timelines (folder)

The conf.xml file represents the DataCleaner configuration for the particular tenant. The file format is the same as described in the Configuration file chapter. It is recommended to use the supplied example conf.xml file (for the 'DC' tenant) as a template for further customization. Specifically the custom elements for task-runner, descriptor-provider and storage-provider in this template conf.xml file is recommended for optimal performance.

The folders are all managed by the DataCleaner monitoring web application, so only in rare cases should you manually interact with them.

It is allowed to add more files and folders to the tenant home. These will not be managed by the monitor application, but can be referenced eg. as the filename paths of datastores defined in conf.xml.
Part VI. DataCleaner
monitor web services
# Table of Contents

17. Job triggering ........................................................................................................ 96  
   Trigger service ........................................................................................................ 96  
   Polling for execution status ..................................................................................... 97  

18. Repository navigation .............................................................................................. 98  
   Job files .................................................................................................................... 98  
   Result files ............................................................................................................... 98  
   Uploading content to the repository ........................................................................ 99  
   Modifying result metadata ....................................................................................... 99  
   Renaming jobs .......................................................................................................... 100  
   Copying jobs ............................................................................................................ 100  
   Deleting jobs ........................................................................................................... 101  

19. Metric web services .................................................................................................. 102  
   Metrics background .................................................................................................. 102  
   Getting a list of available metrics ............................................................................ 102  
   Getting the values of particular metrics .................................................................. 102  

20. Atomic transformations (data cleaning as a service) ............................................. 104  
   What are atomic transformation services? ............................................................... 104  
   Invoking atomic transformations .......................................................................... 104
Chapter 17. Job triggering

Abstract

The DataCleaner web monitor usually handles scheduling of jobs, but sometimes you need to trigger them manually. And sometimes third party systems need to trigger them manually.

In this chapter, a web service for triggering DataCleaner jobs in the DataCleaner monitor web application is explained.

Trigger service

The web service is reachable by this HTTP URL:

/DataCleaner-monitor/repository/{tenant}/jobs/{job}.trigger

The response from the trigger web service is a JSON document that looks like this:

```
{
    "status":"PENDING",
    "logOutput":null,
    "endDate":null,
    "beginDate":null,
    "resultId":"Customer completeness-1349876418933",
    "triggeredBy":"admin"
}
```

Optionally you can also add these parameters:

**Table 17.1. Job triggering HTTP parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>Set to ‘true’ if the server should wait until the job has executed before responding. This will ensure that the client is able to read a 'status' which is either 'SUCCESS' or 'FAILURE' (unless a timeout occurs).</td>
</tr>
<tr>
<td>timeoutMillis</td>
<td>To be used in combination with 'block'. Set this to the max number of milliseconds to wait before responding.</td>
</tr>
</tbody>
</table>

Using these parameters, a typical response will look like this:

```
{
    "status":"SUCCESS",
    "logOutput":
    "2012-10-10 16:04:02 - Job execution BEGIN"
```
Polling for execution status

Often times it is not practical to use the 'block' service parameter, if the job is a long-running job. Rather then you can retrieve the status of the job execution using this service URL:

/DataCleaner-monitor/repository/{tenant}/logs/{resultId}

The 'resultId' part in this URL is mean to match the 'resultId' coming from the responses of the trigger service.

The response of the service will be an XML representation of the execution status. For instance:

```xml
  <ns4:result-id>Customer duplicates-1347004507995</ns4:result-id>
  <ns4:job-begin-date>2012-09-07T09:55:10.607+02:00</ns4:job-begin-date>
  <ns4:job-end-date>2012-09-07T09:55:19.661+02:00</ns4:job-end-date>
  <ns4:execution-status>SUCCESS</ns4:execution-status>
  <ns4:trigger-type>MANUAL</ns4:trigger-type>
  <ns4:triggered-by>admin</ns4:triggered-by>
</ns4:execution-log>
```
Chapter 18. Repository navigation

Abstract

In this chapter we will learn how to use web service URLs to navigate the DataCleaner repository. The repository is available at the root level of the DataCleaner monitoring web application. Access is of course restricted to the tenant's home folder. Therefore, all these web services are located with URLs starting with the form:

/DataCleaner-monitor/repository/DC/...

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

Job files

Job files are available in the reserved folder name 'jobs'. To get a listing of all job files (in JSON format), go to:

/DataCleaner-monitor/repository/DC/jobs

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

The result will resemble this example result:

```
[
    {"repository_path":"/DC/jobs/Contributor name cleansing.analysis.xml","name":"Contributor name cleansing","filename":"Contributor name cleansing.analysis.xml"},
    {"repository_path":"/DC/jobs/Customer completeness.analysis.xml","name":"Customer completeness","filename":"Customer completeness.analysis.xml"},
    {"repository_path":"/DC/jobs/Customer duplicates.analysis.xml","name":"Customer duplicates","filename":"Customer duplicates.analysis.xml"},
    {"repository_path":"/DC/jobs/product_profiling.analysis.xml","name":"product_profiling","filename":"product_profiling.analysis.xml"}
]
```

Further navigating to one of these repository paths, you will be able to read the full XML description of the job, as described in the Analysis Job files chapter:

/DataCleaner-monitor/repository/DC/jobs/product_profiling.analysis.xml

Where 'DC' is the tenant identifier, 'product_profiling' is the name of the job and 'DataCleaner-monitor' is the web application archive name.

Result files

Result files are available in the reserved folder name 'results'. To get a listing of all job files (in JSON format), go to:

/DataCleaner-monitor/repository/DC/jobs/product_profiling.analysis.xml

Where 'DC' is the tenant identifier, 'product_profiling' is the name of the job and 'DataCleaner-monitor' is the web application archive name.
Repository navigation

/DataCleaner-monitor/repository/DC/results

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

The result will resemble this example result:

```
[
    {
        "repository_path": "/DC/results/Customer completeness-1345128427583.analysis.result.dat",
        "filename": "Customer completeness-1345128427583.analysis.result.dat"
    },
    {
        "repository_path": "/DC/results/Customer completeness-1345200106074.analysis.result.dat",
        "filename": "Customer completeness-1345200106074.analysis.result.dat"
    }
]
```

Further navigating to one of these repository paths, you will be able to inspect the analysis result, rendered in HTML format.

/DataCleaner-monitor/repository/DC/results/Customer completeness-1345200106074.

Where 'DC' is the tenant identifier, 'Customer completeness' is the name of the job, '1345200106074' is the timestamp of the result and 'DataCleaner-monitor' is the web application archive name.

**Uploading content to the repository**

Files can be uploaded to the repository using HTTP POST upload requests. The types of uploadable files are:

1. **DataCleaner jobs (.analysis.xml files)**, which are uploaded using this URL template:
   
   POST /DataCleaner-monitor/repository/{tenant}/jobs/{job}.analysis.xml

   Where the request parameter 'file' is a multi-part file containing the new job file content.

2. **DataCleaner results (.analysis.result.dat files)**, which are uploaded using this URL template:
   
   POST /DataCleaner-monitor/repository/{tenant}/results/{result}.analysis.result.dat

   Where the request parameter 'file' is a multi-part file containing the new result file content.

3. **Configuration (conf.xml file)**, which are uploaded using this URL template:
   
   POST /DataCleaner-monitor/repository/{tenant}/conf.xml

   Where the request parameter 'file' is a multi-part file containing the configuration file content.

**Modifying result metadata**

In some cases you can end up having results in the DataCleaner monitor repository which are not correctly configured. Specifically we sometimes see these situations:
1. The registered date of the result might not be correct, if the result was created at a later time than the data it represents.

2. The name of the associated job is wrong, if the result was uploaded from the desktop application but with a reference to a client-side job.

To fix this situation, there is a result modification web service available. The service requires the ADMIN security role.

The request URL is the same as the URL for inspecting the result, but where '.analysis.result.dat' is replaced with '.modify':

```
POST /DataCleaner-monitor/repository/{tenant}/results/{result}.modify
```

The POST request body needs to be a JSON document, specifying the new metadata for the result file. Here's an example:

```
{
    "job": "name of new job",
    "date": "2012-10-17 21:47:00",
    "overwrite": true
}
```

## Renaming jobs

Renaming a job in the DataCleaner monitor repository is possible through a dedicated webservice for job modification.

The request URL is the same as the URL for viewing the job XML, but where '.analysis.xml' is replaced with '.modify':

```
POST /DataCleaner-monitor/repository/{tenant}/jobs/{job}.modify
```

The POST request body needs to be a JSON document, specifying the new metadata for the result file. Here's an example:

```
{
    "name": "new name of job",
    "overwrite": true
}
```

## Copying jobs

Copying a job in the DataCleaner monitor repository is possible through a dedicated webservice for job copying.
The request URL is the same as the URL for viewing the job XML, but where '.analysis.xml' is replaced with '.copy':

```
POST /DataCleaner-monitor/repository/{tenant}/jobs/{job}.copy
```

The POST request body needs to be a JSON document, specifying the new metadata for the result file. Here's an example:

```
{
    "name": "name of copy"
}
```

### Deleting jobs

Deleting a job in the DataCleaner monitor repository is possible through a dedicated webservice for job deletion.

The request URL is the same as the URL for viewing the job XML, but where '.analysis.xml' is replaced with '.delete':

```
POST /DataCleaner-monitor/repository/{tenant}/jobs/{job}.delete
```

Note also that *ALL RELATED* results, timelines and schedules will be deleted, when a job is deleted.

```
POST /DataCleaner-monitor/repository/{tenant}/jobs/{job}.delete
```

The POST request takes no parameters or body information.
Chapter 19. Metric web services

Abstract

Accessing data quality metrics using web services is also possible using DataCleaner monitor. In this chapter we will explain how.

Metrics background

Data quality metrics are all contained within result files. Depending on the job that produced the result, the result may contain different metrics. For instance, if you have a Value distribution analyzer in the job, your result will expose a "Value count" metric (and many more).

To further complicate things, some metrics are parameterized. A metric can be parameterized by either a column name or a query string. For instance, some analyzers (like the String analyzer or Number analyzer) record metrics for multiple columns at the same time. To retrieve metrics from these analyzers you need to provide a column name as a parameter. Other metrics like the "Value count" metric of a Value distribution require a string parameter, specifying which value to get the count of.

Getting a list of available metrics

To get a list of metrics exposed by a particular result, use the following URL pattern:

GET /DataCleaner-monitor/repository/DC/results/[resultname].metrics

Here's an example result:

```
[  
  {"children":null,"analyzerDescriptorName":"Completeness analyzer","metricDescriptorName":"Invalid row count","analyzerName":null,"analyzerInputName":null,"parameterizedByColumnName":false,"parameterizedByQueryString":false,"metricDisplayName":null,"metricColor":null,"formula":null,"paramColumnName":null,"paramQueryString":null}  
]
```

From the metrics list we can see that the result carries 3 metrics of a Completeness analyzer. This analyzer only exposes a few basic metrics, which makes it a nice example, but most other analyzers do expose a lot more.

Getting the values of particular metrics

Using the available metrics we saw before, we can build our own list of metrics of interest. This same list can be sent to the same URL with a HTTP POST request:
POST /DataCleaner-monitor/repository/DC/results/[resultname].metrics

As the request body we need to provide the list of metrics that we're interested in. Null values can be left out. For instance:

```
[  
  {"analyzerDescriptorName":"Completeness analyzer","metricDescriptorName":"Valid row count"},
  {"analyzerDescriptorName":"Completeness analyzer","metricDescriptorName":"Invalid row count"}
]
```

This will return a response like this:

```
{
  "metricDate":1346154850730,
  "metricValues":{
    {"displayName":"Valid row count","value":123},
    {"displayName":"Invalid row count","value":19}
  }
}
```
Chapter 20. Atomic transformations (data cleaning as a service)

Abstract

In this chapter we will learn how to use the DataCleaner monitor as a web service for invoking DataCleaner jobs as on-demand / on-the-fly data cleansing operations. Like in the case of repository navigation, these web services are located with URLs starting with the form:

/DataCleaner-monitor/repository/DC/...

Where 'DC' is the tenant identifier and 'DataCleaner-monitor' is the web application archive name.

What are atomic transformation services?

DataCleaner jobs normally operate on batches of records, for example all records in a CSV file. However, any DataCleaner job can also be used to process single records on the fly. This feature is called "atomic transformations". Such transformations could be used as a part of a data processing pipeline in a SOA architecture or in a web application with flexible data processing needs.

Creating an atomic transformation is easy. Any DataCleaner job can be used for performing atomic transformations. Simply create a job that does some kind of transformation (anything from a simple 'String length' operation to a complete chain of 'Address cleansing' and regex parsing), and place the job in the DataCleaner monitor repository. A specialized web service endpoint will be automatically exposed to use the job as an atomic transformation. The endpoint URL will be:

/DataCleaner-monitor/repository/DC/jobs/[jobname].invoke

Invoking atomic transformations

The contract of the atomic transformation invocation service is dynamic, based on the contract defined by source data and transformed data in the job.

The web service is based on JSON data. You need to provide the JSON data corresponding to the source record format of the job.

For instance, let's say a job defines the following JavaScript transformation with two source columns (POSTALCODE and COUNTRY):
The web service will therefore expect that each JSON record contains two values. The service accepts one or multiple records in it's payload. Use a HTTP POST with a body like this (3 records and 2 columns):

```json
{
  "rows": [
    {
      "values": ["2200", "DK"],
      "values": ["2200", ""],
      "values": ["DK2200", "DK"]
    }
  ]
}
```

And the resulting response body will look like this:

```json
{
  "rows": [
    {
      "values": ["DK2200"],
      "values": ["2200"],
      "values": ["DK2200"]
    },
    "columns": [  
```
Atomic transformations
(data cleaning as a service)

"Country postalcode"
}
)

Note that the 'Content-Type' header of the request is assumed to be 'application/json'.
Part VII. Invoking DataCleaner jobs
Table of Contents

21. Command-line interface ................................................................. 109
   Executables .................................................................................. 109
   Usage scenarios ........................................................................... 109
   Executing an analysis job ............................................................. 110
   Listing datastore contents and available components ............. 110
   Parameterizable jobs ................................................................... 112
   Dynamically overriding configuration elements .................... 113

22. Apache Hadoop and Spark interface ........................................... 115
   Hadoop deployment overview ....................................................... 115
   Setting up Spark and DataCleaner environment ....................... 116
      Upload configuration file to HDFS ............................................ 116
      Upload job file to HDFS ........................................................ 117
      Upload executables to HDFS .................................................. 118
   Launching DataCleaner jobs using Spark ............................... 118
   Limitations of the Hadoop interface .......................................... 119
Chapter 21. Command-line interface

Abstract

DataCleaner offers a Command-Line Interface (CLI) for performing various tasks, including executing analysis jobs, via simple commands that can be invoked eg. as a scheduled tasks.

Executables

Depending on your distribution of DataCleaner, you will have one of these CLI executables included:

1. `datacleaner-console.exe`, which is a Windows-only executable.
2. `datacleaner.cmd`, which is a script to start DataCleaner in Windows.
3. `datacleaner.sh`, which is a script to start DataCleaner in Unix-like systems, like Linux and Mac OS.
4. If you're running DataCleaner in Java Webstart mode, then there is no Command-Line Interface!

Usage scenarios

The usage scenarios of DataCleaner's CLI are:

1. Executing an analysis job
2. List registered datastores
3. List schemas in a datastore
4. List tables in a schema
5. List columns in a table
6. List available analyzers, transformers or filters

How these scenarios are attained is revealed by invoking your executable with the `-usage` argument:

```
> datacleaner-console.exe -usage
-conf (-configuration, --configuration-file) FILE
: XML file describing the configuration of DataCleaner
-ds (-datastore, --datastore-name) VAL
: Name of datastore when printing a list of schemas, tables or columns
-job (--job-file) FILE
: An analysis job XML file to execute
-list [ANALYZERS | TRANSFORMERS | FILTERS | DATASTORES | SCHEMAS | TABLES | COLUMNS]
: Used to print a list of various elements available in the configuration
-s (-schema, --schema-name) VAL
: Name of schema when printing a list of tables or columns
-t (-table, --table-name) VAL
: Name of table when printing a list of columns
```
Executing an analysis job

Here's how to execute an analysis job - we'll use the bundled example job "employees.analysis.xml":

```bash
> datacleaner-console.exe -job examples/employees.analysis.xml
SUCCESS!
```

```
... RESULT:
Value distribution for column: REPORTSTO
Top values:
- 1102: 6
- 1143: 6
- 1088: 5
Null count: 0
Unique values: 0
```

```
RESULT:
Match count Sample
Aaaaaaa      22 William
Aaaa Aaa     1 Foon Yue
```

```
RESULT:
Match count Sample
aaaaaaaaaaa   23 jfirrelli
```

```
RESULT:
Match count Sample
Aaaaa Aaa    17 Sales Rep
AA Aaaaaaaaa  2 VP Marketing
Aaaa Aaaaaaa (AAAA) 1 Sale Manager (EMEA)
Aaaaa Aaaaaaa (AA) 1 Sales Manager (NA)
Aaaa Aaaaaaa (AAAA, AAAA) 1 Sales Manager (JAPAN, APAC)
Aaaaaaaaa    1 President
```

As you can see from the listing, the results of the analysis will be printed directly to the command-line output. If you want to save the results to a file, simply use your operating systems built-in functionality to pipe command-line output to a file, typically using the '>' operator.

Listing datastore contents and available components

The Command-Line Interface allows for listing of datastore contents and available components. The intended usage for this is to aid in hand-editing an analysis file, if this is desired. By using the -list
arguments you can get the metadata of your datastore and the DataCleaner components that will allow you to manually compose an analysis file.

Listing the contents of a datastore is pretty self-explanatory, if you look at the output of the `-usage` command. Here's a few examples, using the example database `orderdb`:

```bash
> datacleaner-console.exe -list datastores
datastores:
----------
Country codes
orderdb
```

```bash
> datacleaner-console.exe -list tables -ds orderdb
tables:
-------
CUSTOMERS
CUSTOMER_W_TER
DEPARTMENT_MANAGERS
DIM_TIME
EMPLOYEES
OFFICES
ORDERDETAILS
ORDERFACT
ORDERS
PAYMENTS
PRODUCTS
QUADRANT_ACTUALS
TRIAL_BALANCE
```

```bash
> datacleaner-console.exe -list columns -ds orderdb -table employees
columns:
--------
EMPLOYEENUMBER
LASTNAME
FIRSTNAME
EXTENSION
EMAIL
OFFICECODE
REPORTSTO
JOBTITLE
```

Listing DataCleaner's components is done by setting the `-list` argument to one of the three component types: ANALYZER, TRANSFORMER or FILTER:

```bash
> datacleaner-console.exe -list analyzers
...
name: Matching analyzer
- Consumes multiple input columns (type: UNDEFINED)
- Property: name=Dictionary, type=Dictionary, required=false
- Property: name=String patterns, type=StringPattern, required=false
```
name: Pattern finder
- Consumes 2 named inputs
  - Input column: Column (type: STRING)
  - Input column: Group column (type: STRING)
- Property: name=Discriminate text case, type=Boolean, required=false
- Property: name=Discriminate negative numbers, type=Boolean, required=false
- Property: name=Discriminate decimals, type=Boolean, required=false
- Property: name=Enable mixed tokens, type=Boolean, required=false
- Property: name=Ignore repeated spaces, type=Boolean, required=false
- Property: name=Upper case patterns expand in size, type=boolean, required=false
- Property: name=Lower case patterns expand in size, type=boolean, required=false
- Property: name=Predefined token name, type=String, required=false
- Property: name=Predefined token regexes, type=String, required=false
- Property: name=Decimal separator, type=Character, required=false
- Property: name=Thousands separator, type=Character, required=false
- Property: name=Minus sign, type=Character, required=false
...

> datacleaner-console.exe -list transformers

...

name: Tokenizer
- Consumes a single input column (type: STRING)
- Property: name=Delimiters, type=char, required=true
- Property: name=Number of tokens, type=Integer, required=true
- Output type is: STRING

name: Whitespace trimmer
- Consumes multiple input columns (type: STRING)
- Property: name=Trim left, type=boolean, required=true
- Property: name=Trim right, type=boolean, required=true
- Property: name=Trim multiple to single space, type=boolean, required=true
- Output type is: STRING
...

**Parameterizable jobs**

If you want to make a part of a job parameterizable/variable, then it is possible to do so. Currently this is a feature only supported by means of editing the .analysis.xml files though, since the DataCleaner graphical user interface does not store job variables when saving jobs.

In the source section of your job, you can add variables which are key/value pairs that will be referenced throughout your job. Each variable can have a default value which will be used in case the variable value is not specified. Here’s a simple example:

```xml
...<source>
  <data-context ref="my_datastore" />
  <columns>
```
In the example we’ve defined two variables: *filename* and *separator*. These we can refer to for specific property values, further down in our job:

```xml
<analyzer>
  <descriptor ref="Write to CSV file"/>
  <properties>
    <property name="File" ref="filename" />
    <property name="Quote char" value="&quot;" />
    <property name="Separator char" ref="separator" />
  </properties>
  <input ref="col_1" />
  <input ref="col_2" />
</analyzer>
```

Now the property values of the *File* and *Separator char* properties in the *Write to CSV file* have been made parameterizable. To execute the job with new variable values, use `-var` parameters from the command line, like this:

```
DataCleaner-console.exe -job my_job.analysis.xml -var filename=/output/my_file.csv -var separator=;
```

### Dynamically overriding configuration elements

Since version 2.5 of DataCleaner it is possible to override elements in the configuration file dynamically from the command line. This is a feature which can be useful in scenarios where you want the to invoke the same job but with slightly different configuration details.

For example, you might want to reuse the same job to be executed on several similar CSV files, or similar database environments. Let us assume that you have a CSV datastore that is defined like this:

```xml
<csv-datastore name="My csv file">
  <filename>/path/to/file.csv</filename>
</csv-datastore>
```
To override the filename dynamically, you have to specify the property path (datastore catalog, then datastore name, then property name) with a `-D` parameter on the command line. Furthermore any spaces or dashes are removed and the succeeding character is uppercased. In the end it will look like "camelCase" strings, like this:

```
DataCleaner-console.exe ... -DdatastoreCatalog.myCsvFile.filename=anotherfile.csv
```

This mechanism can be used for any configuration property within the datastore catalog and reference data catalog.
Chapter 22. Apache Hadoop and Spark interface

Abstract

DataCleaner allows big data processing on the Apache Hadoop platform. In this chapter we will guide you through the process of setting up and running your first DataCleaner job on Hadoop.

Hadoop deployment overview

Apache Hadoop is a distributed system with a number of key components of which a few are important to understand:

1. **YARN**, which is often referred to as the 'operating system' of Hadoop. YARN is the managing entity which assigns resources to running a specific job or task.

2. **HDFS**, which is the Hadoop Distributed File System. This is the location where data is located, but also the place where executables are often shared so that a distributed process can be picked up on many nodes in the cluster.

3. **Namenode**, is a dedicated node in the cluster which deals with HDFS and distribution of data to other nodes, so-called datanodes.

In addition, the DataCleaner Hadoop support is built using Apache Spark, which is a data processing framework that works with Hadoop as well as other clustering technologies. A few important concepts of Apache Spark are useful to understand for DataCleaner's deployment on Hadoop:

1. **Cluster manager**, which is the component that negotiates with the cluster - for instance Hadoop/YARN. From the perspective of Apache Spark, YARN is a *cluster manager*.

2. **Driver program**, which is the program that directs the cluster manager and tells it what to do. In Apache Spark for Hadoop you have two choices: To run the Driver program as an external process ('yarn-client') or to run the Driver program as a process inside YARN itself ('yarn-cluster').

3. **Executor**, which is a node in a Spark cluster that executes a partition (chunk) of a job.

In the top-part of the below image you see Hadoop/YARN as well as Apache Spark, and how they are componentized.
DataCleaner's directory structure on HDFS. As you can see, the usual configuration and job files are used, but placed on HDFS. A special JAR file is placed on HDFS to act as executable for the Apache Spark executors.

**Setting up Spark and DataCleaner environment**

In order to work, Apache Spark requires either of environmental variables `HADOOP_CONF_DIR` or `YARN_CONF_DIR` to a directory containing your Hadoop/Yarn configuration files such as `core-site.xml`, `yarn-site.xml` etc.

**Upload configuration file to HDFS**

DataCleaner on Hadoop needs a regular DataCleaner configuration file (conf.xml). It's best to upload this to the hadoop distributed file system (HDFS). We recommend putting this file into the path `/datacleaner/conf.xml`. Simple example of a configuration file (conf.xml) with a CSV datastore based on a HDFS file or directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://eobjects.org/analyzerbeans/configuration/1.0">
  <datastore-catalog>
    <csv-datastore name="mydata">
      <filename>hdfs://bigdatavm:9000/path/to/data.txt</filename>
      <multiline-values>false</multiline-values>
    </csv-datastore>
  </datastore-catalog>
</configuration>
```

Notice the filename which is here specified with scheme, hostname and port:
<filename>hdfs://bigdatavm:9000/path/to/data.txt</filename>

This here refers to the Hadoop Namenode's hostname and port.

It can also be specified more implicitly, without the username and port:

<filename>hdfs:///path/to/data.txt</filename>

Or even without scheme:

<filename>/path/to/data.txt</filename>

**Upload job file to HDFS**

Upload the DataCleaner job you wish to run (a DataCleaner .analysis.xml job file) to HDFS. We recommend putting this file into a path such as /datacleaner/jobs/myjob.xml. The jobs can be built using the DataCleaner desktop UI, but do ensure that they map well to the configuration file also on HDFS.

Example job file based on the above datastore:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<job xmlns="http://eobjects.org/analyzerbeans/job/1.0">
  <source>
    <data-context ref="mydata" />
    <columns>
      <column id="col_country" path="country" />
      <column id="col_company" path="company" />
    </columns>
  </source>
  <analysis>
    <analyzer>
      <descriptor ref="Create CSV file"/>
      <properties>
        <property name="File" value="hdfs:///path/to/output.csv"/>
        <property name="Separator char" value="\#44;"/>
        <property name="Escape char" value="\"/>
        <property name="Include header" value="true"/>
        <property name="Encoding" value="UTF-8"/>
        <property name="Fields" value="[COUNTRY,CUSTOMERNUMBER]"/>
        <property name="Overwrite file if exists" value="true"/>
      </properties>
      <input ref="col_country" name="Columns"/>
      <input ref="col_company" name="Columns"/>
    </analyzer>
  </analysis>
</job>
```
This particular job is very simplistic - it just copies data from A to B. Notes about the job file contents:

1. The job is referring to `mydata` which was the name of the CSV datastore defined in the configuration file.
2. There is another HDFS file reference used in the "File" property. The filename format is the same as in the configuration file.

If your desktop application has access to the namenode then you can build this job in the desktop application, save it and run it on spark. There is nothing particular about this job that makes it runnable on spark, except that the file references involved are resolvable from the hadoop nodes.

### Upload executables to HDFS

In the installation of DataCleaner you will find the file 'DataCleaner-spark.jar'.

This jar file contains the core of what is needed to run DataCleaner with Apache Spark on Hadoop. It contains also the standard components of DataCleaner.

Upload this jar file to HDFS in the folder `/datacleaner/lib`.

Upload your DataCleaner license file to `/datacleaner/hi_datacleaner.lic`.

Upload any extension jar files that you need (for instance Groovy-DataCleaner.jar) to that same folder.

### Launching DataCleaner jobs using Spark

Go to the Spark installation path to run the job. Use the following command line template:

```
bin/spark-submit --class org.datacleaner.spark.Main --master yarn-cluster /path/to/DataCleaner-spark.jar /path/to/conf.xml /path/to/job_file.analysis.xml (/path/to/custom_properties.properties)
```

A convenient way to organize it is in a shell script like the below, where every individual argument can be edited line by line:

```
#!/bin/sh
SPARK_HOME=/path/to/apache-spark
SPARK_MASTER=yarn-cluster
DC_PRIMARY_JAR=/path/to/DataCleaner-spark.jar
DC_EXTENSION_JARS=/path/to/extension1.jar,/path/to/extension2.jar
DC_CONF_FILE=hdfs:///path/to/conf.xml
DC_JOB_FILE=hdfs:///path/to/job_file.analysis.xml
DC_PROPS=hdfs:///path/to/custom_properties.properties

DC_COMMAND="$SPARK_HOME/bin/spark-submit"
DC_COMMAND="$DC_COMMAND --class org.datacleaner.spark.Main"
DC_COMMAND="$DC_COMMAND --master $SPARK_MASTER"

echo "Using DataCleaner executable: $DC_PRIMARY_JAR"
if [ "$DC_EXTENSION_JARS" != "" ]; then
  echo "Adding extensions: $DC_EXTENSION_JARS"
  DC_COMMAND="$DC_COMMAND --jars $DC_EXTENSION_JARS"
fi
```
DC_COMMAND="$DC_COMMAND $DC_PRIMARY_JAR $DC_CONF_FILE $DC_JOB_FILE $DC_PROPS"

echo "Submitting DataCleaner job $DC_JOB_FILE to Spark $SPARK_MASTER"

The example makes it clear that there are a few more parameters to invoking the job. Let's go through them:

1. **SPARK_MASTER** represents the location of the Driver program, see the section on Hadoop deployment overview.

2. **DC_EXTENSION_JARS** allows you to add additional JAR files with extensions to DataCleaner.

3. **DC_PROPS** is maybe the most important one. It allows you to add a .properties file which can be used for a number of things:
   
   a. Special property `datacleaner.result.hdfs.path` which allows you to specify the filename (on HDFS) where the analysis result (.analysis.result.dat) file is stored. It defaults to `/datacleaner/results/[job name]-[timestamp].analysis.result.dat`

   b. Special property `datacleaner.result.hdfs.enabled` which can be either 'true' (default) or 'false'. Setting this property to false will disable result gathering completely from the DataCleaner job, which gives a significant increase in performance, but no analyzer results are gathered or written. This is thus only relevant for ETL-style jobs where the purpose of the job is to create/insert/update/delete from other datastores or files.

   c. Properties to override configuration defaults.

   d. Properties to set job variables/parameters.

**Limitations of the Hadoop interface**

While the Hadoop interface for DataCleaner allows distributed running of DataCleaner jobs on the Hadoop platform, there are a few limitations:

1. **Datastore support**

   Currently we support a limited set of source datastores from HDFS. CSV files are the primary source here. We do require that files on HDFS are UTF8 encoded and that only single-line values occur.

2. **Non-distributable components**

   A few components are by nature not distributable. If your job depends on these, DataCleaner will resort to executing the job on a single Spark executor, which may have a significant performance impact.

3. **Hadoop Distributions without Namenode**

   Some Hadoop Distributions (such as MapR) have replaced the concept of Namenode with something else. This is mostly fine, but it does mean that file paths with username+port of Namenodes are obviously not working.
Part VIII. Third party integrations
Table of Contents

23. Pentaho integration ........................................................................................................ 122
    Configure DataCleaner in Pentaho Data Integration .................................................. 122
    Launch DataCleaner to profile Pentaho Data Integration steps ................................. 123
    Run Pentaho Data Integration jobs in DataCleaner monitor .................................... 124
    Run DataCleaner jobs in Pentaho Data Integration .................................................... 125
Chapter 23. Pentaho integration

Abstract

DataCleaner offers a number of integrations with the Pentaho [http://www.pentaho.com] open source business intelligence suite. In this chapter we will present an overview of the options available.

Configure DataCleaner in Pentaho Data Integration

In order to use the plugin of DataCleaner in Pentaho, it is required to have a local installation of DataCleaner. The DataCleaner installation can be either a community or a professional edition. The configuration can be set from "Tools". Moreover, after setting the DataCleaner configuration, from the same menu one can launch DataCleaner independent of context in Pentaho.
Launch DataCleaner to profile Pentaho Data Integration steps

In Pentaho Data Integration you can launch DataCleaner by right-clicking any step of your transformations. This will start up DataCleaner with the transformations data pre-loaded, ready for profiling.
This functionality requires installation of the data profiling plugin for Pentaho Data Integration. The instructions and further documentation of this is maintained at Pentaho's wiki page: Kettle Data Profiling with DataCleaner [http://wiki.pentaho.com/display/EAI/Kettle+Data+Profiling+with+DataCleaner].

Run Pentaho Data Integration jobs in DataCleaner monitor

DataCleaner monitor can also be used to run and schedule other jobs than just DataCleaner jobs. An example of this is using DataCleaner monitor to schedule a Pentaho Data Integration transformation. The interoperability in this case is provided through the Carte service provided by Pentaho Data Integration.

Tip

For more information on configuring Carte, we recommend the Pentaho wiki page on the subject: http://wiki.pentaho.com/display/EAI/Carte+Configuration.

To set up the job in DataCleaner monitor, go to the 'Scheduling' page and click 'Build job'. Select the Pentaho Data Integration transformation option:
Follow the on-screen wizard to connect to the Pentaho Carte service and select the transformation to execute.

When the job has been registered, you can use DataCleaner monitor to schedule it, and to monitor execution metrics exposed by the job on the 'Dashboard' page (read more in the section about metric charting).

**Run DataCleaner jobs in Pentaho Data Integration**

*Pentaho Data Integration job entry*. If you want to have DataCleaner scheduled and integrated into an environment where you can eg. iterate over files in a folder etc., then you can use Pentaho Data Integration (PDI), which is an open source ETL tool that includes a scheduler.

Construct a PDI "job" (ie. not a "transformation") and add the DataCleaner job entry. The entry can be found in the submenu 'Utility'. The configuration dialog will look like this:
The most tricky part is to fill out the executable and the job filename. Note that all configuration options can contain PDI variables, like it is the case with ${user.home} in the screenshot above. This is useful if you want to eg. timestamp your resulting files etc.
Part IX. Developer's guide
# Table of Contents

24. Architecture .................................................................................................................. 129  
   Data access ...................................................................................................................... 129  
   Processing framework ................................................................................................. 129  
25. Executing jobs through code ...................................................................................... 131  
   Overview of steps and options ..................................................................................... 131  
   Step 1: Configuration ................................................................................................... 132  
   Step 2: Job ....................................................................................................................... 134  
   Step 3: Execution ........................................................................................................... 137  
   Step 4: Result .................................................................................................................. 139  
26. Developer resources ..................................................................................................... 141  
   Extension development tutorials .................................................................................. 141  
   Building DataCleaner .................................................................................................... 141  
27. Extension packaging ..................................................................................................... 142  
   Annotated components ............................................................................................... 142  
   Single JAR file ............................................................................................................... 142  
   Extension metadata XML ............................................................................................ 142  
   Component icons ......................................................................................................... 143  
28. Embedding DataCleaner .............................................................................................. 144
Chapter 24. Architecture

Abstract

The architecture of DataCleaner can be described from different angles depending on the topic of interest. In the following sections we cover different aspects of the DataCleaner architecture.

Data access

In DataCleaner all sources of data are called 'datastores'. This concept covers both sources that are read/parsed locally and those that are 'connected to', eg. databases and applications. Some datastores can also be written to, for instance relational databases.

DataCleaner uses the Apache MetaModel framework [http://metamodel.apache.org] for data access. From DataCleaner’s perspective, Apache MetaModel provides a number of features:

1. A common way of interacting with different datastores.
2. A programmatic query syntax that abstracts away database-specific SQL dialects, and that is usable also for non-SQL oriented datastores (files etc.).
3. Out-of-the-box connectivity to a lot of sources, eg. CSV files, relational databases, Excel spreadsheets and a lot more.
4. A framework for modelling new sources using the same common model.

DataCleaners datastore model is also extensible in the way that you can yourself implement new datastores in order to hook up DataCleaner to legacy systems, application interfaces and more. For more information refer to the Developer resources chapter.

Processing framework

The way DataCleaner processes data is slightly different compared to most similar (ETL-like) tools. Firstly in the way multithreading is applied, secondly in the way DataCleaner may sometimes optimize the graph at execution time.

Multithreading: The multithreading strategy in DataCleaner enables the tool to have the minimum amount of blocking and buffering and the maximum amount of parallelism and potentially also distribution. Most ETL-like tools apply a threading strategy where each component in a job has its own thread-management as well as an input- and an output-buffer. In DataCleaner the thread management is made such that every record is processed in parallel - each unit of work is stepping through the complete job graph in one single pass. This has a number of interesting traits:

1. There is a high degree of automatic 'load balancing' among the components - less constraints and bottlenecks around the slowest components in the job.
2. The system lends itself to highly distributed processing because statefulness is the exception instead of the rule.
3. There is less waste in the form of buffers inbetween the components of a job.
4. One downside to this approach is that the order of the processed records cannot be guaranteed. This is only very rarely required in the domain of data profiling and analysis, and if it is required there are technical workarounds to apply.
Graph optimization: While a job graph (see wiring components together) may show a particular following order, the engine may at runtime do certain optimizations to it. Some components may provide optimization strategies that involves changing the source query so that the number of (or content of) processed records is changed. Obviously this is a side-effect of using a component that will only be applied in cases where it does not impact other components in a job. The principle is sometimes also referred to as 'Push down optimization'.

An example of this is a 'Null check' filter: If a Null check is applied on a source column and all other components require either a NULL or a NOT_NULL outcome (either explicitly or implicitly), then the 'Null check' filter may add a predicate to the source query to filter out all irrelevant records. For more information on this principle, please read the blog entry 'Push down query optimization in DataCleaner [http://kasper.eobjects.org/2011/12/push-down-query-optimization-in.html]' by Kasper Sørensen.
Chapter 25. Executing jobs through code

Abstract

In this chapter we will go through one of the most common use-cases for integrating DataCleaner into your own application: Executing a DataCleaner job through code.

Overview of steps and options

There's a couple of variants of this story - What kind of configuration options do you want? Would you like to build the job programatically, or have it somewhere on disk as a .analysis.xml file? Will you be doing any processing of the result, or will the job itself contain all the necessary logic.

The various steps and options are depicted in the diagram below. In the following sections we'll go through each of the 4 steps/columns in the diagram:
Step 1: Configuration

Load from file

Use JAXBConfigurationReader to parse configuration file

<interface>
AnalyzerBeansConfiguration

Build programmatically

Use AnalyzerBeansConfigurationImp (important: Datastores + TaskRunner)
The configuration for DataCleaner is represented in the class `DataCleanerConfiguration` (previously 'AnalyzerBeansConfiguration'). You need a DataCleanerConfiguration as a prerequisite for most of the coming operations.

The easiest and probably most convenient option for acquiring an DataCleanerConfiguration instance is to load it from a file, typically named conf.xml (See the Configuration file chapter for more details on this file format). To load the file, use the JaxbConfigurationReader class, like this:

```java
InputStream inputStream = new FileInputStream("conf.xml");
JaxbConfigurationReader configurationReader = new JaxbConfigurationReader();
DataCleanerConfiguration configuration = configurationReader.read(inputStream);
```

Alternatively, you can build the configuration programmatically, through code. This is typically more cumbersome, but in some cases also quite useful if the configuration is to be build dynamically or something like that.

Here's an example where we configure DataCleaner with 2 example datastores and a threadpool of 10 threads:

```java
Datastore datastore1 = new CsvDatastore("my CSV file", "some_data.csv");
boolean multipleConnections = true
Datastore datastore2 = new JdbcDatastore("my database",
    "username", "password", multipleConnections);

DataCleanerConfigurationImpl configuration = new DataCleanerConfigurationImpl();
configuration = configuration.replace(new MultiThreadedTaskRunner(10));
configuration = configuration.replace(new DatastoreCatalogImpl(datastore1, datastore2));
```

Either way we do it, we now have an `DataCleanerConfiguration` with the variable name 'configuration'. Then we can proceed to defining the job to run.
Step 2: Job

Like with the configuration, we can choose to either load the job we want to run from a file, or build it programmatically.

Let's start by simply loading a job from a file. We'll need to use the JaxbJobReader class:

```java
InputStream inputStream = new FileInputStream("my_job.analysis.xml");
JaxbJobReader jobReader = new JaxbJobReader(configuration);
AnalysisJob analysisJob = jobReader.read(inputStream);
```

Note that this is the ‘vanilla’ case. You can also use the JaxbJobReader to read metadata about a job, and even to read a job ‘as a template’, which makes it possible to instantiate the job with certain replacements. For an example of how this functionality is used in DataCleaner's desktop application, see the template jobs section.
The other way of producing a job is to build it programmatically. This is quite involved process that varies quite a lot depending on what kind of job you want to build. But the API has been designed to make it as easy as possible.

To give an overview of the API, consider this list of important classes:

1. **AnalysisJobBuilder**: Represents a mutable job that is being built. This builder object contains source columns of the job, and all the components that consume source columns (or sometimes transformed columns).

2. **TransformerComponentBuilder**, **FilterComponentBuilder**, and **AnalyzerComponentBuilder**: represents mutable components of the job that is being built. These can each have configuration properties, filter requirements, input and output columns.

**Tip**

Be aware of the unfortunate similarity between the 'AnalyzerComponentBuilder' class name and the 'AnalysisJobBuilder' class name. To rid the confusion, remember that the 'analysis' represents the full scope of the job, whereas an 'analyzer' is just a single active part ('component') of the job.

Let's see an example of building a job programmatically. And to ensure that we don't miss important insights, we'll make it a fairly non-trivial job with both filters, transformers and analyzers. The job will encompass:

1. Three source columns from the datastore 'my database': Name, Age and Company_name.
2. All records where 'Company_name' is null will be inserted into the datastore called 'my CSV file'. In the CSV file the columns are called 'fullname' and 'age_years'.
3. All records where 'Company_name' isn't null will 1) have their working address looked up in another table of the database, and 2) the name and the working address will be passed on to a 'Pattern finder' analyzer.

```java
Datastore myDatabase = configuration.getDatastoreCatalog().getDatastore("my database");
Datastore myCsvFile = configuration.getDatastoreCatalog().getDatastore("my CSV file");
AnalysisJobBuilder builder = new AnalysisJobBuilder(configuration);
builder.setDatastore(myDatabase);
builder.addSourceColumns("public.persons.Name","public.persons.Age","public.persons.Company_name")

InputColumn<?> nameColumn = builder.getSourceColumnByName("Name");
InputColumn<?> ageColumn = builder.getSourceColumnByName("Age");
InputColumn<?> companyColumn = builder.getSourceColumnByName("Company_name");

// add a filter to check for null 'company'
FilterComponentBuilder<NullCheckFilter> nullCheckBuilder = builder.addFilter(NullCheckFilter.class);
nullCheckBuilder.addInputColumn(companyColumn);

// add a InsertIntoTable analyzer to write the records without a company to the csv file
AnalyzerComponentBuilder<InsertIntoTableAnalyzer> insertBuilder = builder.addAnalyzer(InsertIntoTableAnalyzer.class);
insertBuilder.addInputColumns(nameColumn, ageColumn);
insertBuilder.setConfiguredProperty("Datastore", myCsvFile);
insertBuilder.setConfiguredProperty("Columns", new String[]{"fullname","age_years"});
insertBuilder.setRequirement(nullCheckBuilder.getOutcome(NullCheckFilter.Category.NULL));
```
Executing jobs through code

```java
// add a lookup for the company working address
TransformerComponentBuilder<TableLookupTransformer> lookupBuilder =
    builder.addTransformer(TableLookupTransformer.class);
lookupBuilder.addInputColumn(companyColumn);
lookupBuilder.setConfiguredProperty("Datastore", myDatabase);
lookupBuilder.setConfiguredProperty("Schema name", "public");
lookupBuilder.setConfiguredProperty("Table name", "companies");
lookupBuilder.setConfiguredProperty("Condition columns", new String[]{"name"});
lookupBuilder.setConfiguredProperty("Output columns", new String[]{"address"});
lookupBuilder.setRequirement(nullCheckBuilder.getOutcome(NullCheckFilter.Category.NOT_NULL));

// reference the 'working address' column and give it a proper name
MutableInputColumn<?> addressColumn = lookupBuilder.getOutputColumns().get(0);
addressColumn.setName("Working address");

// add the Pattern finder analyzer
PatternFinder patternFinder = jobBuilder.addAnalyzer(PatternFinder.class);
patternFinder.addInputColumns(nameColumn, addressColumn);

// validate and produce to AnalysisJob
AnalysisJob analysisJob = jobBuilder.toAnalysisJob();
```

Things to note from this example:

1. Notice how the filter requirements are set up using the .setRequirement(...) method on the succeeding components.

2. There aren't any explicit filter requirements set on the 'Pattern finder' analyzer. This isn't necessary since it depends on a transformed input column ('Working address') which itself has the requirement. DataCleaner will figure out the transitive requirements automatically.

3. One piece of 'magic' is how to set the properties of the components correctly. We can see that we call .setConfiguredProperty(String, Object), but not how to figure out what to pass as arguments. There are two proper ways to figure this out...

   a. You can use DataCleaner's command line to list all components of a specific type, e.g.:

   ```
   > DataCleaner-console.exe -list ANALYZERS
   ...
   name: Insert into table
    - Consumes 2 named inputs
     Input columns: Additional error log values (type: Object)
     Input columns: Values (type: Object)
    - Property: name=Column names, type=String, required=true
    - Property: name=Datastore, type=UpdateableDatastore, required=true
   ...
   ```

   b. Or you can simply open up the component class in an IDE to inspect it's @Configured properties. For instance, if we look at InsertIntoTableAnalyzer.java we'll see:

   ```
   ...
   ```
Executing jobs through code

@Inject
@Configured
@Description("Names of columns in the target table.")
@ColumnProperty
String[] columnNames;

@Inject
@Configured
@Description("Datastore to write to")
UpdateableDatastore datastore;

... 

From these fields we can infer that there will be two configured properties, 'Column names' and 'Datastore'.

Either way we do it, we now have an AnalysisJob with the variable name 'analysisJob'. Then we can proceed to actually executing the job.

**Step 3: Execution**

Executing the job is one of the easiest steps, but obviously there are options available beyond the 'vanilla' scenario.
The simple scenario of running the job is to use the plain `AnalysisRunnerImpl` class, like this:

```java
AnalysisRunner runner = new AnalysisRunnerImpl(configuration);
AnalysisResultFuture resultFuture = runner.run(analysisJob);
```

This will return a `AnalysisResultFuture`, which under most circumstances represents a still-running job. Your application can continue to do other work in the background, or it can decide to block by calling `.await()`.

Here's a typical example of handling the result future:

```java
// block until the job has finished
resultFuture.await();

if (resultFuture.isSuccessful()) {
    // do something with the successful result
    handleResult(resultFuture);
} else {
    List<Throwable> errors = resultFuture.getErrors();
    for (Throwable error : errors) {
        logger.error("An error occurred while executing job", error);
    }
    // usually the first error that occurred is the culprit, so we'll throw that
    throw errors.get(0);
}
```

You might ask what kind of errors will happen while executing a DataCleaner job? The answer is that it can be a lot of things, for instance:

1. The connection to the source database or resource may fail somehow.
2. One of the components in the job may throw an unexpected exception.
3. One of the components may throw an exception because it's configuration is incomplete or invalid (although this will in most cases be detected while building the `AnalysisJob` instance).
4. If you're writing data to another datastore, that may also fail for whatever datastore-dependent reasons.
5. If your job is doing something stupid like a Value Distribution of a billion unique IDs, then you'll run out of memory.

Let's now assume that your job has executed successfully. We'll now look at how you can post-process results and how to save/load them to/from a file.
Great, now we have an `AnalysisResultFuture`, and we've determined that it was successful. What can we do with it?

The results of each analyzer of the job are available through the 'AnalysisResult' interface, which `AnalysisResultFuture` implements. Note that the analyzer result types are very different from each other. For instance, the 'Insert into table' analyzer produces a `WriteDataResult`, while the 'Pattern finder' produces a `PatternFinderResult`. Let's see how you can extract information from them:

```java
// demonstrate the the result
// future implements the AnalysisResult interface, which is sufficient
// for all the followin operations
AnalysisResult analysisResult = resultFuture;
```
List<AnalyzerResult> results = analysisResult.getResults();
for (AnalyzerResult result : results) {
    if (result instanceof WriteDataResult) {
        WriteDataResult writeDataResult = (WriteDataResult)result;
        System.out.println("Inserted " + writeDataResult.getWrittenRowCount() + " records");
    }
    if (result instanceof PatternFinderResult) {
        PatternFinderResult patternFinderResult = (PatternFinderResult)result;
        int matches = patternFinderResult.getMatchCount("Aaaaa Aaaaaa")
        int total = patternFinderResult.getTotalCount();
        System.out.println("There were " + matches + " matches out of " + total + ");
    }
}

As you can see, how you handle the result depends a lot on what type of result is produced.

For generic handling of results, including all the possible result extensions that might occur, DataCleaner employs a renderer framework which selects a result renderer according to type and precedence. If you need such generic functionality, take a look at the classes RendererBean, RendererFactory, Renderer and RenderingFormat.

One common requirement is to persisting it. We recommend doing this by means of Java's serialization, since analysis results are polymorphic and it's structure may be dependent on extensions. You can also device a more "structured" persistence scheme, but beware that it will require quite some stability in terms of which analyzers you add to your jobs.

So let's see how we use Java serialization. But unfortunately AnalysisResultFuture isn't serializable! There is however a class which shares the interface 'AnalysisResult' with 'AnalysisResultFuture', that is serializable, Namely 'SimpleAnalysisResult'. Let's see how to use it and serialize our result to a .analysis.result.dat file, (which DataCleaner can read):

    // make the result serializeable
    AnalysisResult analysisResult = resultFuture;
    analysisResult = new SimpleAnalysisResult(analysisResult.getResultMap());
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("my_result.analysis.result.dat"));
    oos.writeObject(analysisResult);
    oos.close();

And now let's, for example sake, also load our file by deserializing it. For this we need to use the ChangeAwareObjectInputStream class, which ensures backwards compatible deserialization of objects:

    ObjectInputStream ois = new ChangeAwareObjectInputStream(new FileInputStream("my_result.analysis.result.dat"));
    AnalysisResult analysisResult = (AnalysisResult) ois.readObject();

Now the result is restored and you can further work with it.
Chapter 26. Developer resources

Extension development tutorials

There are many useful resources for those who engage in developing extensions (aka. plugins / add-ons) to DataCleaner. To help you on your way, here's a list of useful links. If you think this list is missing a link, please let us know:


Building DataCleaner

Get the source code for DataCleaner from GitHub:

   > git clone https://github.com/datacleaner/DataCleaner.git DataCleaner

Build the projects:

   > cd DataCleaner
   > mvn clean install

Run DataCleaner:

   > cd desktop/ui/target
   > java -jar DataCleaner-desktop-ui-[version].jar
Chapter 27. Extension packaging

Abstract

DataCleaner extensions are packages of added functionality, written in Java. To correctly package an extension, this chapter will walk through the details.

Annotated components

The main principle behind extension discovery in DataCleaner is annotated classes. Any component that should be discovered should have either of these annotations:

1. @java.inject.Named - for classes that implement the Transformer, Filter or Analyzer interface.
2. @org.datacleaner.api.RendererBean - for classes that implement the Renderer interface.

Please refer to the javadoc documentation of the interfaces for details on usage.

Single JAR file

The extension must consist of a single JAR file. If you have dependencies other than the libraries provided by the DataCleaner distribution, you need to package these inside your own JAR file. If you're using Maven for your build, the Maven Assembly Plugin can provide this functionality easily using this snippet in your POM:

```xml
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>2.2.1</version>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Extension metadata XML

To improve the experience, you can optionally include metadata about the extension in an XML file, bundled within the JAR file itself.

The name of the extension metadata file has to be datacleaner-extension.xml and be placed in the root directory of the JAR file. Here's an example of how the file looks like:

```xml
<extension xmlns="http://eobjects.org/datacleaner/extension/1.0">
  <name>My extension</name>
</extension>
```
The added value of this metadata is that DataCleaner can expose this information to the user and also use it to manage updates of the extension etc. The metadata file is however, completely optional.

**Component icons**

If you wish to add a custom icon for your components (eg. a transformer or analyzer), you need to place the icon as a PNG image with the same name as the fully classified class name of the component.

An example: If your component class name is "com.company.ext.MyAnalyzer", then the icon for this component should be located at "/com/company/ext/MyAnalyzer.png" in the extension JAR file.

Similarly, if you bundle your own ComponentCategory implementations (which define the menu groups in DataCleaner), you can define icons for these by adding a PNG file with a fully classified filename corresponding to the ComponentCategory class name.
Chapter 28. Embedding DataCleaner

It is possible to embed DataCleaner into other Java applications. This allows a simple way to add Data Quality Analysis (DQA) and Data Profiling functionality as an addition to the applications that you are building.

The simplest way to embed DataCleaner is simply by doing what DataCleaner's main executable does - instantiate the Bootstrap class with default arguments:

```java
BootstrapOptions bootstrapOptions = new DefaultBootstrapOptions(args);
Bootstrap bootstrap = new Bootstrap(bootstrapOptions);
bootstrap.run();
```

To customize further, add your own implementation of the BootstrapOptions class. The main scenario for embedding DataCleaner is to run the application in the so-called "single datastore mode". This can be achieved by implementing the BootstrapOptions and providing a non-null value for the getSingleDatastore() method.