Cloudera Runtime 7.2.18

Morphlines Reference Guide

Date published: 2015-05-05
Date modified: 2024-04-03



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Implementing your own Custom Command..... 5

Morphline commands overview..... 5

kite-morphlines-core-stdio..... 10

kite-morphlines-core-stdlib..... 14

kite-morphlines-avro.....42

kite-morphlines-json..... 46

kite-morphlines-hadoop-core..... 48

kite-morphlines-hadoop-parquet-avro.....49

kite-morphlines-hadoop-rcfile..... 50

kite-morphlines-hadoop-sequencefile.....51

kite-morphlines-maxmind..... 52

kite-morphlines-metrics-servlets..... 54

kite-morphlines-protobuf..... 61

kite-morphlines-tika-core..... 63

kite-morphlines-tika-decompress..... 64

kite-morphlines-saxon.....65

kite-morphlines-solr-core..... 70

kite-morphlines-solr-cell.....75

kite-morphlines-useragent.....80

Implementing your own Custom Command

Before we dive into the currently available commands, it is worth noting again that perhaps the most important property of the Morphlines framework is how easy it is to add new transformations and I/O commands and integrate existing functionality and third party systems. If none of the existing commands match your use case, you can easily write your own command and plug it in.

Simply implement the Java interface Command or subclass AbstractCommand, have it handle a Record and add the resulting Java class to the classpath, along with a CommandBuilder implementation that defines the name(s) of the command and serves as a factory. Here are two example implementations: toString and readLine. No registration or other administrative action is required. Indeed, none of the standard commands are special or intrinsically known per se. All commands are implemented like this, even including standard commands such as pipe, if, and tryRules. This means your custom commands can even replace any standard commands, if desired. When writing your own custom Command implementation you can take advantage of lifecycle methods: you compile regexes, read config files and perform other expensive setup stuff once in the constructor of the command, then reuse the resulting optimized representation any number of times in the execution phase on method process(). Putting it all together, you can download and run this working example Maven project that demonstrates how to unit test Morphline config files and custom Morphline commands. With that said, the following tables provide a short description of each available command and a link to the complete documentation.

Morphline commands overview

Morphlines provides a set of frequently-used high-level transformation and I/O commands that can be combined in application specific ways. This is a short description of each available command and a link to the complete documentation.

kite-morphlines-core-stdio

readBlob

Converts a byte stream to a byte array in main memory.

readClob

Converts a byte stream to a string.

readCSV

Extracts zero or more records from the input stream of bytes representing a Comma Separated Values (CSV) file.

readLine

Emits one record per line in the input stream.

readMultiLine

Log parser that collapses multiple input lines into a single record, based on regular expression pattern matching.

kite-morphlines-core-stdlib

addCurrentTime

Adds the result of System.currentTimeMillis() to a given output field.

addLocalHost

Adds the name or IP of the local host to a given output field.

addValues

Adds a list of values (or the contents of another field) to a given field.

addValuesIfAbsent

Adds a list of values (or the contents of another field) to a given field if not already contained.

callParentPipe

Implements recursion for extracting data from container data formats.

contains

Returns whether or not a given value is contained in a given field.

convertTimestamp

Converts the timestamps in a given field from one of a set of input date formats to an output date format.

decodeBase64

Converts a Base64 encoded String to a byte[].

dropRecord

Silently consumes records without ever emitting any record. Think /dev/null.

equals

Succeeds if all field values of the given named fields are equal to the the given values and fails otherwise.

extractURIComponents

Extracts subcomponents such as host, port, path, query, etc from a URI.

extractURIComponent

Extracts a particular subcomponent from a URI.

extractURIQueryParameters

Extracts the query parameters with a given name from a URI.

findReplace

Examines each string value in a given field and replaces each substring of the string value that matches the given string literal or grok pattern with the given replacement.

generateUUID

Sets a universally unique identifier on all records that are intercepted.

grok

Uses regular expression pattern matching to extract structured fields from unstructured log or text data.

head

Ignores all input records beyond the N-th record, akin to the Unix head command.

if

Implements if-then-else conditional control flow.

java

Scripting support for Java. Dynamically compiles and executes the given Java code block.

logTrace, logDebug, logInfo, logWarn, logError

Logs a message at the given log level to SLF4J.

not

Inverts the boolean return value of a nested command.

pipe

Pipes a record through a chain of commands.

removeFields

Removes all record fields for which the field name matches a blacklist but not a whitelist.

removeValues

Removes all record field values for which the field name and value matches a blacklist but not a whitelist.

replaceValues

Replaces all record field values for which the field name and value matches a blacklist but not a whitelist.

sample

Forwards each input record with a given probability to its child command.

separateAttachments

Emits one separate output record for each attachment in the input record's list of attachments.

setValues

Assigns a given list of values (or the contents of another field) to a given field.

split

Divides a string into substrings, by recognizing a separator (a.k.a. "delimiter") which can be expressed as a single character, literal string, regular expression, or grok pattern.

splitKeyValue

Splits key-value pairs where the key and value are separated by the given separator, and adds the pair's value to the record field named after the pair's key.

startReportingMetricsToCSV

Starts periodically appending the metrics of all commands to a set of CSV files.

startReportingMetricsToJMX

Starts publishing the metrics of all commands to JMX.

startReportingMetricsToSLF4J

Starts periodically logging the metrics of all morphline commands to SLF4J.

toByteArray

Converts a String to the byte array representation of a given charset.

toString

Converts a Java object to it's string representation; optionally also removes leading and trailing whitespace.

translate

Replace a string with the replacement value defined in a given dictionary aka lookup hash table.

tryRules

Simple rule engine for handling a list of heterogeneous input data formats.

kite-morphlines-avro

readAvroContainer

Parses an Apache Avro binary container and emits a morphline record for each contained Avro datum.

readAvro

Parses containerless Avro and emits a morphline record for each contained Avro datum.

extractAvroTree

Recursively walks an Avro tree and extracts all data into a single morphline record.

extractAvroPaths

Extracts specific values from an Avro object, akin to a simple form of XPath.

toAvro

Converts a morphline record to an Avro record.

writeAvroToByteArray

Serializes Avro records into a byte array.

kite-morphlines-json

readJson

Parses JSON and emits a morphline record for each contained JSON object, using the Jackson library.

extractJsonPaths

Extracts specific values from a JSON object, akin to a simple form of XPath.

kite-morphlines-hadoop-core

downloadHdfsFile

Downloads, on startup, zero or more files or directory trees from HDFS to the local file system.

openHdfsFile

Opens an HDFS file for read and returns a corresponding Java InputStream.

kite-morphlines-hadoop-parquet-avro

readAvroParquetFile

Parses a Hadoop Parquet file and emits a morphline record for each contained Avro datum.

kite-morphlines-hadoop-rcfile

readRCFile

Parses an Apache Hadoop RCFile and emits morphline records row-wise or column-wise.

kite-morphlines-hadoop-sequencefile

readSequenceFile

Parses an Apache Hadoop SequenceFile and emits a morphline record for each contained key-value pair.

kite-morphlines-maxmind

geoIP

Returns Geolocation information for a given IP address, using an efficient in-memory Maxmind database lookup.

kite-morphlines-metrics-servlets

registerJVMMetrics

Registers metrics that are related to the Java Virtual Machine with the MorphlineContext.

startReportingMetricsToHTTP

Exposes liveness status, health check status, metrics state and thread dumps via a set of HTTP URLs served by Jetty, using the AdminServlet.

kite-morphlines-protobuf

readProtobuf

Parses an InputStream that contains protobuf data and emits a morphline record containing the protobuf object as an attachment.

extractProtobufPaths

Extracts specific values from a protobuf object, akin to a simple form of XPath.

kite-morphlines-tika-core

detectMimeType

Uses Apache Tika to autodetect the MIME type of binary data.

kite-morphlines-tika-decompress

decompress

Decompresses gzip and bzip2 format.

unpack

Unpacks tar, zip, and jar format.

kite-morphlines-saxon

convertHTML

Converts any HTML to XHTML, using the TagSoup Java library.

xquery

Parses XML and runs the given W3C XQuery over it, using the Saxon Java library.

xslt

Parses XML and runs the given W3C XSL Transform over it, using the Saxon Java library.

kite-morphlines-solr-core

solrLocator

Specifies a set of configuration parameters that identify the location and schema of a Solr server or SolrCloud.

loadSolr

Inserts, updates or deletes records into a Solr server or MapReduce Reducer.

generateSolrSequenceKey

Assigns a unique key that is the concatenation of a field and a running count of the record number within the current session.

sanitizeUnknownSolrFields

Removes record fields that are unknown to Solr schema.xml, or moves them to fields with a given prefix.

tokenizeText

Uses the embedded Solr/Lucene Analyzer library to generate tokens from a text string, without sending data to a Solr server.

kite-morphlines-solr-cell

solrCell

Uses Apache Tika to parse data, then maps the Tika output back to a record using Apache SolrCell.

kite-morphlines-useragent

userAgent

Parses a user agent string and returns structured higher level data like user agent family, operating system, version, and device type.

kite-morphlines-core-stdio

readBlob

The readBlob command (source code) converts a byte stream to a byte array in main memory. It emits one record for the entire input stream of the first attachment, interpreting the stream as a Binary Large Object (BLOB), i.e. emits a corresponding Java byte array. The BLOB is put as a Java byte array into the _attachment_body output field by default.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
outputField	_attachment_body	Name of the output field where the BLOB will be stored.

Example usage:

```
readBlob {}
```

readClob

The readClob command (source code) converts bytes to a string. It emits one record for the entire input stream of the first attachment, interpreting the stream as a Character Large Object (CLOB). The CLOB is put as a string into the message output field by default.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.

Property Name	Default	Description
charset	null	he character encoding to use, for example, UTF-8. If none is specified the charset specified in the _attachment_charset input field is used instead.
outputField	message	Name of the output field where the CLOB will be stored.

Example usage:

```
readClob {
  charset : UTF-8
}
```

readCSV

The readCSV command (source code) extracts zero or more records from the input stream of the first attachment of the record, representing a Comma Separated Values (CSV) file.

For the format see this article.

Some CSV files contain a header line that contains embedded column names. This command does not support reading and using such embedded column names as output field names because this is considered unreliable for production systems. If the first line of the CSV file is a header line, you must set the ignoreFirstLine option to true. You must explicitly define the columns configuration parameter in order to name the output fields.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
separator	","	The character separating any two fields. Must be a string of length one.
columns	n/a	The name of the output fields for each input column. An empty string indicates omit this column in the output. If more columns are contained in the input than specified here, those columns are automatically named columnN.
ignoreFirstLine	false	Whether to ignore the first line. This flag can be used for CSV files that contain a header line.
trim	true	Whether leading and trailing whitespace shall be removed from the output fields.
addEmptyStrings	true	Whether or not to add zero length strings to the output fields.
charset	null	The character encoding to use, for example, UTF-8. If none is specified the charset specified in the _attachment_charset input field is used instead.
quoteChar	""	Must be a string of length zero or one. If this parameter is a String containing a single character then a quoted field can span multiple lines in the input stream. To disable quoting and multiline fields set this parameter to the empty string "".

Property Name	Default	Description
commentPrefix	""	Must be a string of length zero or one, for example "#". If this parameter is a String containing a single character then lines starting with that character are ignored as comments. To disable the comment line feature set this parameter to the empty string "".
maxCharactersPerRecord	1000000	Records longer than maxCharactersPerRecord characters are handled according to the policy specified in the onMaxCharactersPerRecord parameter described below.
onMaxCharactersPerRecord	throwException	Records longer than maxCharactersPerRecord characters are handled according to the policy specified in the onMaxCharactersPerRecord parameter. Must be one of ignoreRecord or throwException. A value of ignoreRecord indicates to ignore such records and continue with the following record (warnings about such events are emitted to the log file). This value is typically used in production. A value of throwException indicates to throw an exception and fail hard in such cases. This value is typically used for testing.

If the parameter quoteChar is a String containing a single character then a quoted field can span multiple lines in the input stream, for example as shown in the following example CSV input containing a single record with three columns:

```
column0,"Look, new hot tub under redwood tree!
All bubbly!",column2
```

The above example can be parsed by specifying a double-quote character for the parameter quoteChar, using backslash syntax per the JSON specification, as follows:

```
readCSV {
  ...
  quoteChar : "\""
```

If the parameter commentPrefix is a String containing a single character then lines starting with that character are ignored as comments. Example:

```
#This is a comment line. It is ignored.
```

Example usage for CSV (Comma Separated Values):

```
readCSV {
  separator : ","
  columns : [Age,"",Extras,Type]
  ignoreFirstLine : false
  quoteChar : ""
  commentPrefix : ""
  trim : true
  charset : UTF-8
}
```

Example usage for TSV (Tab Separated Values):

```
readCSV {
  separator : "\t"
  columns : [Age,"",Extras,Type]
  ignoreFirstLine : false
```

```
  quoteChar : ""
  commentPrefix : ""
  trim : true
  charset : UTF-8
}
```

Example usage for SSV (Space Separated Values):

```
readCSV {
  separator : " "
  columns : [Age,"",Extras,Type]
  ignoreFirstLine : false
  quoteChar : ""
  commentPrefix : ""
  trim : true
  charset : UTF-8
}
```

Example usage for Apache Hive (Values separated by non-printable CTRL-A character):

```
readCSV {
  separator : "\u0001" # non-printable CTRL-A character
  columns : [Age,"",Extras,Type]
  ignoreFirstLine : false
  quoteChar : ""
  commentPrefix : ""
  trim : false
  charset : UTF-8
}
```

readLine

The readLine command (source code) emits one record per line in the input stream of the first attachment. The line is put as a string into the message output field. Empty lines are ignored.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
ignoreFirstLine	false	Whether to ignore the first line. This flag can be used for CSV files that contain a header line.
commentPrefix	""	A character that indicates to ignore this line as a comment for example, "#". To disable the comment line feature set this parameter to the empty string "".
charset	null	The character encoding to use, for example, UTF-8. If none is specified the charset specified in the _attachment_charset input field is used instead.

Example usage:

```
readLine {
  ignoreFirstLine : true
  commentPrefix : "#"
  charset : UTF-8
}
```

readMultiLine

The readMultiLine command (source code) is a multiline log parser that collapses multiple input lines into a single record, based on regular expression pattern matching. It supports regex, what, and negate configuration parameters similar to logstash. The line is put as a string into the message output field.

For example, this can be used to parse log4j with stack traces. Also see https://gist.github.com/smougenot/3182192 and http://logstash.net/docs/1.1.13/filters/multiline.

The input stream or byte array is read from the first attachment of the input record.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
regex	n/a	This parameter should match what you believe to be an indicator that the line is part of a multi-line record.
what	previous	This parameter must be one of "previous" or "next" and indicates the relation of the regex to the multi-line record.
negate	false	This parameter can be true or false. If true, a line not matching the regex constitutes a match of the multiline filter and the previous or next action is applied. The reverse is also true.
charset	null	The character encoding to use, for example, UTF-8. If none is specified the charset specified in the _attachment_charset input field is used instead.

Example usage:

```
# parse log4j with stack traces
readMultiLine {
  regex : "(^.+Exception: .+)|(^\\s+at .+)|(^\\s+\\.\\.\\. \\d+ more)|(^\\s*
Caused by:.+)"
  what : previous
  charset : UTF-8
}

# parse sessions; begin new record when we find a line that starts with "S
tarted session"
readMultiLine {
  regex : "Started session.*"
  what : next
  charset : UTF-8
}
```

kite-morphlines-core-stdlib

This maven module contains standard transformation commands, such as commands for flexible log file analysis, regular expression based pattern matching and extraction, operations on fields for assignment and comparison, operations on fields with list and set semantics, if-then-else conditionals, string and timestamp conversions, scripting support for dynamic java code, a small rules engine, logging, and metrics and counters.

addCurrentTime

The addCurrentTime command (source code) adds the result of System.currentTimeMillis() as a Long integer to a given output field. Typically, a convertTimestamp command is subsequently used to convert this timestamp to an application specific output format.

The command provides the following configuration options:

Property Name	Default	Description
field	timestamp	The name of the field to set.
preserveExisting	true	Whether to preserve the field value if one is already present.

Example usage:

```
addCurrentTime {}
```

addLocalHost

The addLocalHost command (source code) adds the name or IP of the local host to a given output field.

The command provides the following configuration options:

Property Name	Default	Description
field	host	The name of the field to set.
preserveExisting	true	Whether to preserve the field value if one is already present.
useIP	true	Whether to add the IP address or fully-qualified hostname.

Example usage:

```
addLocalHost {
  field : my_host
  useIP : false
}
```

addValues

The addValues command (source code) adds a list of values (or the contents of another field) to a given field. The command takes a set of outputField : values pairs and performs the following steps: For each output field, adds the given values to the field. The command can fetch the values of a record field using a field expression, which is a string of the form @{fieldname}.

Example usage:

```
addValues {
  # add values "text/log" and "text/log2" to the source_type output field
  source_type : [text/log, text/log2]
  # add integer 123 to the pid field
  pid : [123]
  # add all values contained in the first_name field to the name field
  name : "@{first_name}"
}
```

addValuesIfAbsent

The addValuesIfAbsent command (source code) adds a list of values (or the contents of another field) to a given field if not already contained. This command is the same as the addValues command, except that a given value is only added to the output field if it is not already contained in the output field.

Example usage:

```
addValuesIfAbsent {
  # add values "text/log" and "text/log2" to the source_type output field
  # unless already present
  source_type : [text/log, text/log2]

  # add integer 123 to the pid field, unless already present
  pid : [123]

  # add all values contained in the first_name field to the name field
  # unless already present
  name : "@{first_name}"
}
```

callParentPipe

The callParentPipe command (source code) implements recursion for extracting data from container data formats. The command routes records to the enclosing pipe object. Recall that a morphline is a pipe. Thus, unless a morphline contains nested pipes, the parent pipe of a given command is the morphline itself, meaning that the first command of the morphline is called with the given record. Thus, the callParentPipe command effectively implements recursion, which is useful for extracting data from container data formats in elegant and concise ways. For example, you could use this to extract data from tar.gz files. This command is typically used in combination with the commands detectMi meType, tryRules, decompress, unpack, and possibly solrCell.

Example usage:

```
callParentPipe {}
```

For a real world example, see the solrCell command.

contains

The contains command (source code) returns whether or not a given value is contained in a given field. The command succeeds if one of the field values of the given named field is equal to one of the the given values, and fails otherwise. Multiple fields can be named, in which case the results are ANDed.

Example usage:

```
# succeed if the _attachment_mimetype field contains a value "avro/binary"
# fail otherwise
contains { _attachment_mimetype : [avro/binary] }

# succeed if the tags field contains a value "version1" or "version2",
# fail otherwise
contains { tags : [version1, version2] }
```

convertTimestamp

The convertTimestamp command (source code) converts the timestamps in a given field from one of a set of input date formats (in an input timezone) to an output date format (in an output timezone), while respecting daylight savings time rules. The command provides reasonable defaults for common use cases.

The command provides the following configuration options:

Property Name	Default	Description
field	timestamp	The name of the field to convert.
inputFormats	A list of common input date formats	A list of SimpleDateFormat or "unixTimeInMillis" or "unixTimeInSeconds". "unixTimeInMillis" and "unixTimeInSeconds" indicate the difference, measured in milliseconds and seconds, respectively, between a timestamp and midnight, January 1, 1970 UTC. Multiple input date formats can be specified. If none of the input formats match the field value then the command fails.
inputTimezone	UTC	The time zone to assume for the input timestamp.
inputLocale	""	The Java Locale to assume for the input timestamp.
outputFormat	"yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"	The SimpleDateFormat to which to convert. Can also be "unixTimeInMillis" or "unixTimeInSeconds". "unixTimeInMillis" and "unixTimeInSeconds" indicate the difference, measured in milliseconds and seconds, respectively, between a timestamp and midnight, January 1, 1970 UTC.
outputTimezone	UTC	The time zone to assume for the output timestamp.
outputLocale	""	The Java Locale to assume for the output timestamp.

Example usage with plain SimpleDateFormat:

```
# convert the timestamp field to "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
# The input may match one of "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
# or "yyyy-MM-dd'T'HH:mm:ss" or "yyyy-MM-dd".
convertTimestamp {
  field : timestamp
  inputFormats : ["yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", "yyyy-MM-dd'T'HH:mm:ss",
 "yyyy-MM-dd"]
  inputTimezone : America/Los_Angeles
  outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
  outputTimezone : UTC
}
```

Example usage with Solr date rounding:

A SimpleDateFormat can also contain a literal string for Solr date rounding down to, say, the current hour, minute or second. For example: '/MINUTE' to round to the current minute. This kind of rounding results in fewer distinct values and improves the performance of Solr several ways:

- it uses less memory for many functions, e.g. sorting by time, restricting by date ranges etc.
- it improves speed of range queries based on time, e.g. "restrict documents to those from the last 7 days"
- In the case of faceting by the values in the field it will improve both memory requirements and speed.

For these reasons, it's advisable to store dates in the coarsest granularity that's appropriate for your application.

Example usage with Solr date rounding:

```
# convert the timestamp field to "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
# and indicate to Solr that it shall round the time down to the current mi
nute
# per http://lucene.apache.org/solr/4_4_0/solr-core/org/apache/solr/util/Da
teMathParser.html
```

```
convertTimestamp {
  ...
  outputFormat : "yyyy-MM-dd'T'HH:mm:ss.SSS'Z/MINUTE'"
  ...
}
```

decodeBase64

The decodeBase64 command (source code) converts a Base64 encoded String to a byte[] per Section 6.8. "Base64 Content-Transfer-Encoding" of RFC 2045. The command converts each value in the given field and replaces it with the decoded value.

The command provides the following configuration options:

Property Name	Default	Description
field	n/a	The name of the field to modify.

Example usage:

```
decodeBase64 {
  field : screenshot_base64
}
```

dropRecord

The dropRecord command (source code) silently consumes records without ever emitting any record. This is much like piping to /dev/null in Unix.

Example usage:

```
dropRecord {}
```

equals

The equals command (source code) succeeds if all field values of the given named fields are equal to the the given values and fails otherwise. Multiple fields can be named, in which case a logical AND is applied to the results.

Example usage:

```
# succeed if the _attachment_mimetype field contains the value "avro/binary"
# and nothing else, fail otherwise
equals { _attachment_mimetype : [avro/binary] }
# succeed if the tags field contains nothing but the values "version1"
# and "highPriority", in that order, fail otherwise
equals { tags : [version1, highPriority] }
```

extractURIComponents

The extractURIComponents command (source code) extracts the following subcomponents from the URIs contained in the given input field and adds them to output fields with the given prefix: scheme, authority, host, port, path, query, fragment, schemeSpecificPart, userInfo.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field that contains zero or more URIs.
outputFieldPrefix	""	A prefix to prepend to output field names.

Property Name	Default	Description
failOnInvalidURI	false	If an URI is syntactically invalid (i.e. throws a URISyntaxException on parsing), fail the command (true) or ignore this URI (false).

Example usage:

```
extractURIComponents {
  inputField : my_uri
  outputFieldPrefix : uri_component_
}
```

For example, given the input field myUri with the value http://userinfo@www.bar.com:8080/errors.log?foo=x&bar=y&foo=z#fragment the expected output is as follows:

Name	Value
myUri	http://userinfo@www.bar.com:8080/errors.log?foo=x&bar=y&foo=z#fragment
uri_component_authority	userinfo@www.bar.com:8080
uri_component_fragment	fragment
uri_component_host	www.bar.com
uri_component_path	/errors.log
uri_component_port	8080
uri_component_query	foo=x&bar=y&foo=z
uri_component_scheme	http
uri_component_schemeSpecificPart	//userinfo@www.bar.com:8080/errors.log?foo=x&bar=y&foo=z
uri_component_userInfo	userinfo

extractURIComponent

The extractURIComponent command (source code) extracts a subcomponent from the URIs contained in the given input field and adds it to the given output field. This is the same as the extractURIComponents command, except that only one component is extracted.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field that contains zero or more URIs.
outputField	n/a	The field to add output values to.
failOnInvalidURI	false	If an URI is syntactically invalid (i.e. throws a URISyntaxException on parsing), fail the command (true) or ignore this URI (false).
component	n/a	The type of information to extract. Can be one of scheme, authority, host, port, path, query, fragment, schemeSpecificPart, userInfo

Example usage:

```
extractURIComponent {
  inputField : my_uri
  outputField : my_scheme
  component : scheme
```

```
}
```

extractURIQueryParameters

The extractURIQueryParameters command (source code) extracts the query parameters with a given name from the URIs contained in the given input field and appends them to the given output field.

The command provides the following configuration options:

Property Name	Default	Description
parameter	n/a	The name of the query parameter to find.
inputField	n/a	The name of the input field that contains zero or more URI values.
outputField	n/a	The field to add output values to.
failOnInvalidURI	false	If an URI is syntactically invalid (i.e. throws a URISyntaxException on parsing), fail the command (true) or ignore this URI (false).
maxParameters	1000000000	The maximum number of values to append to the output field per input field.
charset	UTF-8	The character encoding to use, for example, UTF-8.

Example usage:

```
extractURIQueryParameters {
  parameter : foo
  inputField : myUri
  outputField : my_query_params
}
```

For example, given the input field myUri with the value http://userinfo@www.bar.com/errors.log?foo=x&bar=y&foo=z#fragment the expected output record is:

```
my_query_params:x
my_query_params:z
```

findReplace

The findReplace command (source code) examines each string value in a given field and replaces each substring of the string value that matches the given string literal or grok pattern with the given replacement.

This command also supports grok dictionaries and regexes in the same way as the grok command.

The command provides the following configuration options:

Property Name	Default	Description
field	n/a	The name of the field to modify.
pattern	n/a	The search string to match.
isRegex	false	Whether or not to interpret the pattern as a grok pattern (true) or string literal (false).
dictionaryFiles	[]	A list of zero or more local files or directory trees from which to load dictionaries. Only applicable if isRegex is true. See grok command.

Property Name	Default	Description
dictionaryString	null	An optional inline string from which to load a dictionary. Only applicable if isRegex is true. See grok command.
replacement	n/a	The replacement pattern (isRegex is true) or string literal (isRegex is false).
replaceFirst	false	For each field value, whether or not to skip any matches beyond the first match.

Example usage with grok pattern:

```
findReplace {
  field : message
  dictionaryFiles : [kite-morphlines-core/src/test/resources/grok-dictionar
ies]
  pattern : """%{WORD:myGroup}"""
  #pattern : """(\b\w+\b)"""
  isRegex : true
  replacement : "${myGroup}!"
  #replacement : "$1!"
  #replacement : ""
  replaceFirst : false
}
```

Input: "hello world"

Expected output: "hello! world!"

generateUUID

The generateUUID command (source code) sets a universally unique identifier on all records that are intercepted. An example UUID is b5755073-77a9-43c1-8fad-b7a586fc1b97, which represents a 128bit value.

The command provides the following configuration options:

Property Name	Default	Description
field	id	The name of the field to set.
preserveExisting	true	Whether to preserve the field value if one is already present.
prefix	""	The prefix string constant to prepend to each generated UUID.
type	secure	This parameter must be one of "secure" or "nonSecure" and indicates the algorithm used for UUID generation. Unfortunately, the cryptographically "secure" algorithm can be comparatively slow - if it uses /dev/random on Linux, it can block waiting for sufficient entropy to build up. In contrast, the "nonSecure" algorithm never blocks and is much faster. The "nonSecure" algorithm uses a secure random seed but is otherwise deterministic, though it is one of the strongest uniform pseudo random number generators known so far.

Example usage:

```
generateUUID {
  field : my_id
}
```

grok

The grok command (source code) uses regular expression pattern matching to extract structured fields from unstructured log data.

This is well suited for syslog logs, apache, and other webserver logs, mysql logs, and in general, any log format that is generally written for humans and not computer consumption.

A grok command can load zero or more dictionaries. A dictionary is a file, file on the classpath, or string that contains zero or more REGEXNAME to REGEX mappings, one per line, separated by space. Here is an example dictionary:

```
INT (?:[+-]?(?:[0-9]+))
HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z-]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z-]{
0,62}))*(\.?|\b)
```

In this example, the regex named "INT" is associated with the following regex pattern:

```
[+-]?(?:[0-9]+)
```

and matches strings like "123", whereas the regex named "HOSTNAME" is associated with the following regex pattern:

```
\b(?:[0-9A-Za-z][0-9A-Za-z-]{0,62})(?:\.(?:[0-9A-Za-z][0-9A-Za-z-]{0,62}))*(
\.?|\b)
```

and matches strings like "www.cloudera.com".

Morphlines ships with several standard grok dictionaries. Dictionaries may be loaded from a file or directory of files on the local filesystem (see "dictionaryFiles"), files found on the classpath (see "dictionaryResources"), or literal inline strings in the morphlines configuration file (see "dictionaryString").

A grok command can contain zero or more grok expressions. Each grok expression refers to a record input field name and can contain zero or more grok patterns. The following is an example grok expression that refers to the input field named "quot;message" and contains two grok patterns:

```
expressions : {
  message : """\s+%{INT:pid} %{HOSTNAME:my_name_servers}"""
}
```

The syntax for a grok pattern is

```
%{REGEX_NAME:GROUP_NAME}
```

for example

```
%{INT:pid}
```

or

```
%{HOSTNAME:my_name_servers}
```

The REGEXNAME is the name of a regex within a loaded dictionary.

The GROUPNAME is the name of an output field.

If all expressions of the grok command match the input record, then the command succeeds and the content of the named capturing group is added to this output field of the output record. Otherwise, the record remains unchanged and the grok command fails, causing backtracking of the command chain.

Note: The morphline configuration file is implemented using the HOCON format (Human Optimized Config Object Notation). HOCON is basically JSON slightly adjusted for the configuration file use case. HOCON syntax is defined

at HOCON github page and as such, multi-line strings are similar to Python or Scala, using triple quotes. If the three-character sequence """ appears, then all Unicode characters until a closing """ sequence are used unmodified to create a string value.

In addition, the grok command supports the following parameters:

Property Name	Default	Description
dictionaryFiles	[]	A list of zero or more local files or directory trees from which to load dictionaries.
dictionaryResources	[]	A list of zero or more classpath resources (i.e. dictionary files on the classpath) from which to load dictionaries. Unlike "dictionaryFiles" it is not possible to specify directories.
dictionaryString	null	An optional inline string from which to load a dictionary.
extract	true	Can be "false", "true", or "inplace". Add the content of named capturing groups to the input record ("inplace"), to a copy of the input record ("true"), or to no record ("false").
numRequiredMatches	atLeastOnce	Indicates the minimum and maximum number of field values that must match a given grok expression for each input field name. Can be "atLeastOnce" (default), "once", or "all".
findSubstrings	false	Indicates whether the grok expression must match the entire input field value or merely a substring within.
addEmptyStrings	false	Indicates whether zero length strings stemming from empty (but matching) capturing groups shall be added to the output record.

Example usage:

```
# Index syslog formatted files
#
# Example input line:
#
# <164>Feb  4 10:46:14 syslog sshd[607]: listening on 0.0.0.0 port 22.
#
# Expected output record fields:
#
# syslog_pri:164
# syslog_timestamp:Feb  4 10:46:14
# syslog_hostname:syslog
# syslog_program:sshd
# syslog_pid:607
# syslog_message:listening on 0.0.0.0 port 22.
#
grok {
  dictionaryFiles : [kite-morphlines-core/src/test/resources/grok-dictionar
ies]
  expressions : {
    message : """<%{POSINT:syslog_pri}>%{SYSLOGTIMESTAMP:syslog_timestamp}
 %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_p
id}\])?: %{GREEDYDATA:syslog_message}"""

    #message2 : "(?<queue_field>.*)"
    #message4 : "%{NUMBER:queue_field}"
  }
}
```

More example usage:

```
# Split a line on one or more whitespace into substrings,
# and add the substrings to the "columns" output field.
#
# Example input line with tabs:
#
# "hello\t\tworld\tfoo"
#
# Expected output record fields:
#
# columns:hello
# columns:world
# columns:foo
#
grok {
  expressions : {
    message : """(?<columns>.+?)(\s+|\z)"""
  }
  findSubstrings : true
}
```

Even more example usage:

```
# Index a custom variant of syslog files where subfacility is optional.
#
# Dictionaries in this example are loaded from three places:
# * The my-jar-dictionaries/my-commands file found on the classpath.
# * The local file kite-morphlines-core/src/test/resources/grok-dictionaries
.
# * The inline definition shown in dictionaryString.
#
# Example input line:
#
# <179>Jun 10 04:42:51 www.foo.com Jun 10 2013 04:42:51 : %myproduct-3-mysu
bfacility-123456: Health probe failed
#
# Expected output record fields:
#
# my_message_code:%myproduct-3-mysubfacility-123456
# my_product:myproduct
# my_level:3
# my_subfacility:mysubfacility
# my_message_id:123456
# syslog_message:%myproduct-3-mysubfacility-123456: Health probe failed
#
grok {
  dictionaryResources : [my-jar-dictionaries/my-commands]
  dictionaryFiles : [kite-morphlines-core/src/test/resources/grok-dictionari
es]
  dictionaryString : """
    MY_CUSTOM_TIMESTAMP %{MONTH} %{MONTHDAY} %{YEAR} %{TIME}
  """
  expressions : {
    message : """<%{POSINT}>%{SYSLOGTIMESTAMP} %{SYSLOGHOST} %{MY_CUSTOM_
TIMESTAMP} : (?<syslog_message>(?<my_message_code>%%{\w+:my_product}-%{\w+:m
y_level}(-%{\w+:my_subfacility})?-%{\w+:my_message_id}): %{GREEDYDATA})"""
  }
}
```

Note: An easy way to test grok out is to use an online grok debugger.

head

The head command (source code) ignores all input records beyond the N-th record, thus emitting at most N records, akin to the Unix head command. This can be helpful to quickly test a morphline with the first few records from a larger dataset.

The command provides the following configuration options:

Property Name	Default	Description
limit	-1	The maximum number of records to emit. -1 indicates never ignore any records.

Example usage:

```
# emit only the first 10 records
head {
  limit : 10
}
```

if

The if command (source code) implements if-then-else conditional control flow. It consists of a chain of zero or more conditions commands, as well as an optional chain of zero or or more commands that are processed if all conditions succeed ("then commands"), as well as an optional chain of zero or more commands that are processed if one of the conditions fails ("else commands").

If one of the commands in the then chain or else chain fails, then the entire if command fails and any remaining commands in the then or else branch are skipped.

The command provides the following configuration options:

Property Name	Default	Description
conditions	[]	A list of zero or more commands.
then	[]	A list of zero or more commands.
else	[]	A list of zero or more commands.

Example usage:

```
if {
  conditions : [
    { contains { _attachment_mimetype : [avro/binary] } }
  ]
  then : [
    { logInfo { format : "processing then..." } }
  ]
  else : [
    { logInfo { format : "processing else..." } }
  ]
}
```

More example usage - Ignore all records that don't have an id field:

```
if {
  conditions : [
    { equals { id : [] } }
  ]
  then : [
    { logTrace { format : "Ignoring record because it has no id: {}", args :
 ["@{}"] } }
    { dropRecord {} }
```

```
  ]
}
```

More example usage - Ignore all records that contain at least one value in the malformed field:

```
if {
  conditions : [
    { not { equals { malformed : [] } } }
  ]
  then : [
    { logTrace { format : "Ignoring record containing at least one malfor
med value: {}", args : ["@{}"] } }
    { dropRecord {} }
  ]
}
```

java

The java command (source code) provides scripting support for Java. The command compiles and executes the given Java code block, wrapped into a Java method with a Boolean return type and several parameters, along with a Java class definition that contains the given import statements.

The following enclosing method declaration is used to pass parameters to the Java code block:

public static boolean evaluate(Record record, com.typesafe.config.Config config, Command parent, Command child, MorphlineContext context, org.slf4j.Logger logger) {

```
  // your custom java code block goes here...
}
```

Compilation is done in main memory, meaning without writing to the filesystem.

The result is an object that can be executed (and reused) any number of times. This is a high performance implementation, using an optimized variant of JSR 223 Java Scripting". Calling eval() just means calling Method.i nvoke(), and, as such, has the same minimal runtime cost. As a result of the low cost, this command can be called on the order of 100 million times per second per CPU core on industry standard hardware.

The command provides the following configuration options:

Property Name	Default	Description
imports	A default list sufficient for typical usage.	A string containing zero or more Java import declarations.
code	[]	A Java code block as defined in the Java language specification. Must return a Boolean value.

Example usage:

```
java {
  imports : "import java.util.*;"
  code: """
    // Update some custom metrics - see http://metrics.codahale.com/getting-
started/
    context.getMetricRegistry().counter("myMetrics.myCounter").inc(1);
    context.getMetricRegistry().meter("myMetrics.myMeter").mark(1);
    context.getMetricRegistry().histogram("myMetrics.myHistogram").update(1
00);
    com.codahale.metrics.Timer.Context timerContext = context.getMetricRegi
stry().timer("myMetrics.myTimer").time();

    // manipulate the contents of a record field
```

```
    List tags = record.get("tags");
    if (!tags.contains("hello")) {
      return false;
    }
    tags.add("world");

    logger.debug("tags: {} for record: {}", tags, record); // log to SLF4J
    timerContext.stop(); // measure how much time the code block took
    return child.process(record); // pass record to next command in chain
        """
}
```

The main disadvantage of the scripting "java" command is that you can't reuse things like compiled regexes across command invocations so you end up having to compile the same regex over and over again, for each record again. The main advantage is that you can implement your custom logic exactly the way you want, without recourse to perhaps overly generic features of certain existing commands.

logTrace, logDebug, logInfo, logWarn, logError

These commands log a message at the given log level to SLF4J. The command can fetch the values of a record field using a field expression, which is a string of the form @{fieldname}. The special field expression @{} can be used to log the entire record.

Example usage:

```
# log the entire record at DEBUG level to SLF4J
logDebug { format : "my record: {}", args : ["@{}"] }
```

More example usage:

```
# log the timestamp field and the entire record at INFO level to SLF4J
logInfo {
  format : "timestamp: {}, record: {}"
  args : ["@{timestamp}", "@{}"]
}
```

To automatically print diagnostic information such as the content of records as they pass through the morphline commands, consider enabling TRACE log level, for example by adding the following line to your log4j.properties file:

```
log4j.logger.org.kitesdk.morphline=TRACE
```

not

The not command (source code) inverts the boolean return value of a nested command. The command consists of one nested command, the Boolean return value of which is inverted.

Example usage:

```
if {
  conditions : [
    {
      not {
        grok {
          ... some grok expressions go here
        }
      }
    }
  ]
  then : [
    { logDebug { format : "found no grok match: {}", args : ["@{}"] } }
```

```
    { dropRecord {} }
  ]
  else : [
    { logDebug { format : "found grok match: {}", args : ["@{}"] } }
  ]
}
```

pipe

The pipe command (source code) pipes a record through a chain of commands. The pipe command has an identifier and contains a chain of zero or more commands, through which records get piped. A command transforms the record into zero or more records. The output records of a command are passed to the next command in the chain. A command has a Boolean return code, indicating success or failure. If any command in the pipe fails (meaning that it returns false), the whole pipe fails (meaning that it returns false), which causes backtracking of the command chain.

Because a pipe is itself a command, a pipe can contain arbitrarily nested pipes. A morphline is a pipe. "Morphline" is simply another name for the pipe at the root of the command tree.

The command provides the following configuration options:

Property Name	Default	Description
id	n/a	An identifier for this pipe.
importCommands	[]	A list of zero or more import specifications, each of which makes all morphline commands that match the specification visible to the morphline. A specification can import all commands in an entire Java package tree (specification ends with ".**"), all commands in a Java package (specification ends with ".*"), or the command of a specific fully qualified Java class (all other specifications). Other commands present on the Java classpath are not visible to this morphline.
commands	[]	A list of zero or more commands.

Example usage demonstrating a pipe with two commands, namely addValues and logDebug:

```
pipe {
  id : my_pipe

  # Import all commands in these java packages, subpackages and classes.
  # Other commands on the Java classpath are not visible to this morphline.
  importCommands : [
    "org.kitesdk.**",   # package and all subpackages
    "org.apache.solr.**", # package and all subpackages
    "com.mycompany.mypackage.*", # package only
    "org.kitesdk.morphline.stdlib.GrokBuilder" # fully qualified class
  ]

  commands : [
    { addValues { foo : bar }}
    { logDebug { format : "output record: {}", args : ["@{}"] } }
  ]
}
```

removeFields

The removeFields command (source code) removes all record fields for which the field name matches at least one of the given blacklist predicates, but matches none of the given whitelist predicates.

A predicate can be a regex pattern (e.g. "regex:foo.*") or POSIX glob pattern (e.g. "glob:foo*") or literal pattern (e.g. "literal:foo") or "*" which is equivalent to "glob:*".

The command provides the following configuration options:

Property Name	Default	Description
blacklist	"*"	The blacklist predicates to use. If the blacklist specification is absent it defaults to MATCH ALL.
whitelist	""	The whitelist predicates to use. If the whitelist specification is absent it defaults to MATCH NONE.

Example usage:

```
# Remove all fields where the field name matches at least one of foo.* or ba
r* or baz,
# but matches none of foobar or baro*
removeFields {
  blacklist : ["regex:foo.*", "glob:bar*", "literal:baz"]
  whitelist: ["literal:foobar", "glob:baro*"]
}
```

Input record:

```
foo:data
foobar:data
barx:data
barox:data
baz:data
hello:data
```

Expected output:

```
foobar:data
barox:data
hello:data
```

removeValues

The removeValues command (source code) removes all record field values for which all of the following conditions hold:

1) the field name matches at least one of the given nameBlacklist predicates but none of the given nameWhitelist predicates.

2) the field value matches at least one of the given valueBlacklist predicates but none of the given valueWhitelist predicates.

A predicate can be a regex pattern (e.g. "regex:foo.*") or POSIX glob pattern (e.g. "glob:foo*") or literal pattern (e.g. "literal:foo") or "*" which is equivalent to "glob:*".

This command behaves in the same way as the replaceValues command except that maching values are removed rather than replaced.

The command provides the following configuration options:

Property Name	Default	Description
nameBlacklist	"*"	The blacklist predicates to use for entry names (i.e. entry keys). If the blacklist specification is absent it defaults to MATCH ALL.

Property Name	Default	Description
nameWhitelist	""	The whitelist predicates to use for entry names (i.e. entry keys). If the whitelist specification is absent it defaults to MATCH NONE.
valueBlacklist	"*"	The blacklist predicates to use for entry values. If the blacklist specification is absent it defaults to MATCH ALL.
valueWhitelist	""	The whitelist predicates to use for entry values. If the whitelist specification is absent it defaults to MATCH NONE.

Example usage:

```
# Remove all field values where the field name and value matches at least on
e of foo.* or bar* or baz,
# but matches none of foobar or baro*
removeValues {
  nameBlacklist : ["regex:foo.*", "glob:bar*", "literal:baz", "literal:xxxx
"]
  nameWhitelist: ["literal:foobar", "glob:baro*"]
  valueBlacklist : ["regex:foo.*", "glob:bar*", "literal:baz", "literal:xxx
x"]
  valueWhitelist: ["literal:foobar", "glob:baro*"]
}
```

Input record:

```
foobar:data
foo:[foo,foobar,barx,barox,baz,baz,hello]
barx:foo
barox:foo
baz:[foo,foo]
hello:foo
```

Expected output:

```
foobar:data
foo:[foobar,barox,hello]
barox:foo
hello:foo
```

replaceValues

The replaceValues command (source code) replaces all record field values for which all of the following conditions hold:

1) the field name matches at least one of the given nameBlacklist predicates but none of the given nameWhitelist predicates.

2) the field value matches at least one of the given valueBlacklist predicates but none of the given valueWhitelist predicates.

A predicate can be a regex pattern (e.g. "regex:foo.*") or POSIX glob pattern (e.g. "glob:foo*") or literal pattern (e.g. "literal:foo") or "*" which is equivalent to "glob:*".

This command behaves in the same way as the removeValues command except that maching values are replaced rather than removed.

The command provides the following configuration options:

Property Name	Default	Description
nameBlacklist	"*"	The blacklist predicates to use for entry names (i.e. entry keys). If the blacklist specification is absent it defaults to MATCH ALL.
nameWhitelist	""	The whitelist predicates to use for entry names (i.e. entry keys). If the whitelist specification is absent it defaults to MATCH NONE.
valueBlacklist	"*"	The blacklist predicates to use for entry values. If the blacklist specification is absent it defaults to MATCH ALL.
valueWhitelist	""	The whitelist predicates to use for entry values. If the whitelist specification is absent it defaults to MATCH NONE.
replacement	n/a	The replacement string to use for matching entry values.

Example usage:

```
# Replace with "myReplacement" all field values where the field name and val
ue
# matches at least one of foo.* or bar* or baz, but matches none of foobar
or baro*
replaceValues {
  nameBlacklist : ["regex:foo.*", "glob:bar*", "literal:baz", "literal:xxxx
"]
  nameWhitelist: ["literal:foobar", "glob:baro*"]
  valueBlacklist : ["regex:foo.*", "glob:bar*", "literal:baz", "literal:xxx
x"]
  valueWhitelist: ["literal:foobar", "glob:baro*"]
  replacement : "myReplacement"
}
```

Input record:

```
foobar:data
foo:[foo,foobar,barx,barox,baz,baz,hello]
barx:foo
barox:foo
baz:[foo,foo]
hello:foo
```

Expected output:

```
foobar:data
foo:[myReplacement,foobar,myReplacement,barox,myReplacement,myReplacement,he
llo]
barox:foo
baz:[myReplacement,myReplacement]
hello:foo
```

sample

The sample command (source code) forwards each input record with a given probability to its child command, and silently ignores all other input records. Sampling is based on a random number generator. This can be helpful to easily test a morphline with a random subset of records from a large dataset.

The command provides the following configuration options:

Property Name	Default	Description
probability	1.0	The probability that any given record will be forwarded; must be in the range 0.0 (forward no records) to 1.0 (forward all records).
seed	null	An optional long integer that ensures that the series of random numbers will be identical on each morphline run. This can be helpful for deterministic unit testing and debugging. If the seed parameter is absent the pseudo random number generator is initialized with a seed that's obtained from a cryptographically "secure" algorithm, leading to a different sample selection on each morphline run.

Example usage:

```
sample {
  probability : 0.0001
  seed : 12345
}
```

separateAttachments

The separateAttachments command (source code) emits one output record for each attachment in the input record's list of attachments. The result is many records, each of which has at most one attachment.

Example usage:

```
separateAttachments {}
```

setValues

The setValues command (source code) assigns a given list of values (or the contents of another field) to a given field. This command is the same as the addValues command, except that it first removes all values from the given output field, and then it adds new values.

Example usage:

```
setValues {
  # assign values "text/log" and "text/log2" to source_type output field
  source_type : [text/log, text/log2]

  # assign the integer 123 to the pid field
  pid : [123]

  # remove the url field
  url : []

  # assign all values contained in the first_name field to the name field
  name : "@{first_name}"
}
```

split

The split command (source code) divides strings into substrings, by recognizing a separator (a.k.a. "delimiter") which can be expressed as a single character, literal string, regular expression, or grok pattern. This class provides the functionality of Guava's Splitter class as a morphline command, plus it also supports grok dictionaries and regexes in the same way as the grok command, except it doesn't support the grok extraction features.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field.
outputField	null	The name of the field to add output values to, i.e. a single string. Example: tokens. One of outputField or outputFields must be present, but not both.
outputFields	null	The names of the fields to add output values to, i.e. a list of strings. Example: [firstName, lastName, "", age]. An empty string in a list indicates omit this column in the output. One of outputField or outputFields must be present, but not both.
separator	n/a	The delimiting string to search for.
isRegex	false	Whether or not to interpret the separator as a grok pattern (true) or string literal (false).
dictionaryFiles	[]	A list of zero or more local files or directory trees from which to load dictionaries. Only applicable if isRegex is true. See grok command.
dictionaryString	null	An optional inline string from which to load a dictionary. Only applicable if isRegex is true. See grok command.
trim	true	Whether or not to apply the String.trim() method on the output values to be added.
addEmptyStrings	false	Whether or not to add zero length strings to the output field.
limit	-1	The maximum number of items to add to the output field per input field value. -1 indicates unlimited.

Example usage with multiple output field names and literal string as separator:

```
split {
  inputField : message
  outputFields : [first_name, last_name, "", age]
  separator : ","
  isRegex : false
  #separator : """\s*,\s*"""
  #isRegex : true
  addEmptyStrings : false
  trim : true
}
```

Input record:

```
message:"Nadja,Redwood,female,8"
```

Expected output:

```
first_name:Nadja
last_name:Redwood
age:8
```

More example usage with one output field and literal string as separator:

```
split {
  inputField : message
  outputField : substrings
```

```
  separator : ","
  isRegex : false
  #separator : """\s*,\s*"""
  #isRegex : true
  addEmptyStrings : false
  trim : true
}
```

Input record:

```
message:"_a ,_b_ ,c__"
```

Expected output contains a "substrings" field with three values:

```
substrings:_a
substrings:_b_
substrings:c__
```

More example usage with grok pattern or normal regex:

```
split {
  inputField : message
  outputField : substrings
  # dictionaryFiles : [kite-morphlines-core/src/test/resources/grok-dicti
onaries]
  dictionaryString : """COMMA_SURROUNDED_BY_WHITESPACE \s*,\s*"""
  separator : """%{COMMA_SURROUNDED_BY_WHITESPACE}"""
  # separator : """\s*,\s*"""
  isRegex : true
  addEmptyStrings : true
  trim : false
}
```

splitKeyValue

The splitKeyValue command (source code) iterates over the items in a given record input field, interprets each item as a key-value pair where the key and value are separated by the given separator, and adds the pair's value to the record field named after the pair's key. Typically, the input field items have been placed there by an upstream split command with a single output field.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field.
outputFieldPrefix	""	A string to be prepended to each output field name.
separator	"="	The string separating the key from the value.
isRegex	false	Whether or not to interpret the separator as a grok pattern (true) or string literal (false).
dictionaryFiles	[]	A list of zero or more local files or directory trees from which to load dictionaries. Only applicable if isRegex is true. See grok command.
dictionaryString	null	An optional inline string from which to load a dictionary. Only applicable if isRegex is true. See grok command.

Property Name	Default	Description
trim	true	Whether or not to apply the String.trim() method on the output keys and values to be added.
addEmptyStrings	false	Whether or not to add zero length strings to the output field.

Example usage:

```
splitKeyValue {
  inputField : params
  separator : "="
  outputFieldPrefix : "/"
}
```

Input record:

```
params:foo=x
params: foo = y
params:foo
params:fragment=z
```

Expected output:

```
/foo:x
/foo:y
/fragment:z
```

Example usage that extracts data from iptables log file

```
# read each line in the file
{
  readLine {
    charset : UTF-8
  }
}

# extract timestamp and key value pair string
{
  grok {
    dictionaryFiles : [target/test-classes/grok-dictionaries/grok-patterns]
    expressions : {
      message : """%{SYSLOGTIMESTAMP:timestamp} %{GREEDYDATA:key_value_pair
s_string}"""
    }
  }
}

# split key value pair string on blanks into an array of key value pairs
{
  split {
    inputField : key_value_pairs_string
    outputField : key_value_array
    separator : " "
  }
}

# split each key value pair on '=' char and extract its value into record fi
elds named after the key
{
  splitKeyValue {
```

```
    inputField : key_value_array
    outputFieldPrefix : ""
    separator : "="
    addEmptyStrings : false
    trim : true
  }
}

# remove temporary work fields
{
  setValues {
    key_value_pairs_string : []
    key_value_array : []
  }
}
```

Input file:

```
Feb  6 12:04:42 IN=eth1 OUT=eth0 SRC=1.2.3.4 DST=6.7.8.9 ACK DF WINDOW=0
```

Expected output record:

```
timestamp:Feb  6 12:04:42
IN:eth1
OUT:eth0
SRC:1.2.3.4
DST:6.7.8.9
WINDOW:0
```

startReportingMetricsToCSV

The startReportingMetricsToCSV command (source code) starts periodically appending the metrics of all morphline commands to a set of CSV files. The CSV files are named after the metrics.

The command provides the following configuration options:

Property Name	Default	Description
outputDir	n/a	The relative or absolute path of the output directory on the local file system. The directory and it's parent directories will be created automatically if they don't yet exist.
frequency	"10 seconds"	The amount of time between reports to the output file, given in HOCON duration format.
locale	JVM default Locale	Format numbers for the given Java Locale. Example: "en_US"
defaultDurationUnit	milliseconds	Report output durations in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days.
defaultRateUnit	seconds	Report output rates in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days. Example output: events/second
metricFilter	null	Only report metrics which match the given (optional) filter, as described in more detail below. If the filter is absent all metrics match.

metricFilter

A metricFilter uses pattern matching with include/exclude specifications to determine if a given metric shall be reported to the output destination.

A metric consists of a metric name and a metric class name. A metric matches the filter if the metric matches at least one include specification, but matches none of the exclude specifications. An include/exclude specification consists of zero or more expression pairs. Each expression pair consists of an expression for the metric name, as well as an expression for the metric's class name. Each expression can be a regex pattern (e.g. "regex:foo.*") or POSIX glob pattern (e.g. "glob:foo*") or literal string (e.g. "literal:foo") or "*" which is equivalent to "glob:*". Each expression pair defines one expression for the metric name and another expression for the metric class name.

If the include specification is absent it defaults to MATCH ALL. If the exclude specification is absent it defaults to MATCH NONE.

Example startReportingMetricsToCSV usage:

```
startReportingMetricsToCSV {
  outputDir : "mytest/metricsLogs"
  frequency : "10 seconds"
  locale : en_US
}
```

More example startReportingMetricsToCSV usage:

```
startReportingMetricsToCSV {
  outputDir : "mytest/metricsLogs"
  frequency : "10 seconds"
  locale : en_US
  defaultDurationUnit : milliseconds
  defaultRateUnit : seconds
  metricFilter : {
    includes : { # if absent defaults to match all
      "literal:foo" : "glob:foo*"
      "regex:.*" : "glob:*"
    }
    excludes : { # if absent defaults to match none
      "literal:foo.bar" : "*"
    }
  }
}
```

Example output log file:

```
t,count,mean_rate,m1_rate,m5_rate,m15_rate,rate_unit
1380054913,2,409.752100,0.000000,0.000000,0.000000,events/second
1380055913,2,258.131131,0.000000,0.000000,0.000000,events/second
```

startReportingMetricsToJMX

The startReportingMetricsToJMX command (source code) starts publishing the metrics of all morphline commands to JMX.

The command provides the following configuration options:

Property Name	Default	Description
domain	metrics	The name of the JMX domain (aka category) to publish to.
durationUnits	null	Report output durations of the given metrics in the given time units. This optional parameter is a JSON object where the key is the metric name and the value is a time unit. The time unit can be one of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days.

Property Name	Default	Description
defaultDurationUnit	milliseconds	Report all other output durations in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days.
rateUnits	null	Report output rates of the given metrics in the given time units. This optional parameter is a JSON object where the key is the metric name and the value is a time unit. The time unit can be one of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days.
defaultRateUnit	seconds	Report all other output rates in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days. Example output: events/second
metricFilter	null	Only report metrics which match the given (optional) filter, as described in more detail at metricFilter. If the filter is absent all metrics match.

Example startReportingMetricsToJMX usage:

```
startReportingMetricsToJMX {
  domain : myMetrics
}
```

More example startReportingMetricsToJMX usage:

```
startReportingMetricsToJMX {
  domain : myMetrics
  durationUnits : {
    myMetrics.myTimer : minutes
  }
  defaultDurationUnit : milliseconds
  rateUnits : {
    myMetrics.myTimer : milliseconds
    morphline.logDebug.numProcessCalls : milliseconds
  }
  defaultRateUnit : seconds
  metricFilter : {
    includes : { # if absent defaults to match all
      "literal:foo" : "glob:foo*"
      "regex:.*" : "glob:*"
    }
    excludes : { # if absent defaults to match none
      "literal:foo.bar" : "*"
    }
  }
}
```

startReportingMetricsToSLF4J

The startReportingMetricsToSLF4J command (source code) starts periodically logging the metrics of all morphline commands to SLF4J.

The command provides the following configuration options:

Property Name Default Description

logger metrics The name of the SLF4J logger to write to. marker null The optional name of the SLF4J marker object to associate with each logging request. frequency "10 seconds" The amount of time between reports to the output file, given in HOCON duration format. defaultDurationUnit milliseconds Report output durations in the given time unit.

One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days. defaultRateUnit seconds Report output rates in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days. Example output: events/second metricFilter null Only report metrics which match the given (optional) filter, as described in more detail at metricFilter. If the filter is absent all metrics match.

Example startReportingMetricsToSLF4J usage:

```
startReportingMetricsToSLF4J {
  logger : "org.kitesdk.morphline.domain1"
  frequency : "10 seconds"
}
```

More example startReportingMetricsToSLF4J usage:

```
startReportingMetricsToSLF4J {
  logger : "org.kitesdk.morphline.domain1"
  frequency : "10 seconds"
  defaultDurationUnit : milliseconds
  defaultRateUnit : seconds
  metricFilter : {
    includes : { # if absent defaults to match all
      "literal:foo" : "glob:foo*"
      "regex:.*" : "glob:*"
    }
    excludes : { # if absent defaults to match none
      "literal:foo.bar" : "*"
    }
  }
}
```

Example output log line:

```
457  [metrics-logger-reporter-thread-1] INFO  org.kitesdk.morphline.domain1
 - type=METER, name=morphline.logDebug.numProcessCalls, count=2, mean_rate=1
44.3001443001443, m1=0.0, m5=0.0, m15=0.0, rate_unit=events/second
```

toByteArray

The toByteArray command (source code) converts the Java objects in a given field via Object.toString() to their string representation, and then via String.getBytes(Charset) to their byte array representation. If the input Java objects are already byte arrays the command does nothing.

The command provides the following configuration options:

Property Name	Default	Description
field	n/a	The name of the field to convert.
charset	UTF-8	The character encoding to use.

Example usage:

```
toByteArray { field : _attachment_body }
```

toString

The toString command (source code) converts the Java objects in a given field using the Object.toString() method to their string representation, and optionally also applies the String.trim() method to remove leading and trailing whitespace.

The command provides the following configuration options:

Property Name	Default	Description
field	n/a	The name of the field to convert.
trim	false	Whether or not to apply the String.trim() method.

Example usage:

```
toString { field : source_type }
```

translate

The translate command (source code) examines each value in a given field and replaces it with the replacement value defined in a given dictionary aka lookup hash table.

The command provides the following configuration options:

Property Name	Default	Description
field	n/a	The name of the field to modify.
dictionary	n/a	The lookup hash table to use for finding matches and replacement values
fallback	null	The fallback value to use as replacement if no match is found. If no fallback is defined and no match is found then the command fails.

Example usage to translate Syslog severity level numeric codes to string labels:

```
translate {
  field : level
  dictionary : {
     0 : Emergency
     1 : Alert
     2 : Critical
     3 : Error
     4 : Warning
     5 : Notice
     6 : Informational
     7 : Debug
  }
  fallback : Unknown # if no fallback is defined and no match is found then
 the command fails
}
```

Input: level:0

Expected output: level:Emergency

Input: level:999

Expected output: level:Unknown

tryRules

The tryRules command (source code) is a simple rule engine for handling a list of heterogeneous input data formats. The command consists of zero or more rules. A rule consists of zero or more commands.

The rules of a tryRules command are processed in top-down order. If one of the commands in a rule fails, the tryRules command stops processing this rule, backtracks and tries the next rule, and so on, until a rule is found that runs all its commands to completion without failure (the rule succeeds). If a rule succeeds, the remaining rules of the current

tryRules command are skipped. If no rule succeeds the record remains unchanged, but a warning may be issued or an exception may be thrown.

Because a tryRules command is itself a command, a tryRules command can contain arbitrarily nested tryRules commands. By the same logic, a pipe command can contain arbitrarily nested tryRules commands and a tryRules command can contain arbitrarily nested pipe commands. This helps to implement complex functionality for advanced usage.

The command provides the following configuration options:

Property Name	Default	Description
catchExceptions	false	Whether Java exceptions thrown by a rule shall be caught, with processing continuing with the next rule (true), or whether such exceptions shall not be caught and consequently propagate up the call chain (false).
throwExceptionIfAllRulesFailed	true	Whether to throw a Java exception if no rule succeeds.

Example usage:

```
tryRules {
  catchExceptions : false
  throwExceptionIfAllRulesFailed : true
  rules : [
    # next rule of tryRules cmd:
    {
      commands : [
        { contains { _attachment_mimetype : [avro/binary] } }
        ... handle Avro data here
        { logDebug { format : "output record: {}", args : ["@{}"] } }
      ]
    }

    # next rule of tryRules cmd:
    {
      commands : [
        { contains { _attachment_mimetype : [text/csv] } }
        ... handle CSV data here
        { logDebug { format : "output record: {}", args : ["@{}"] } }
      ]
    }

    # if desired, the last rule can serve as a fallback mechanism
    # for records that don't match any rule:
    {
      commands : [
        { logWarn { format : "Ignoring record with unsupported input format:
 {}", args : ["@{}"] } }
        { dropRecord {} }
      ]
    }
  ]
}
```

kite-morphlines-avro

This maven module contains morphline commands for reading, extracting, and transforming Avro files and Avro objects.

readAvroContainer

The readAvroContainer command (source code) parses an InputStream or byte array that contains Apache Avro binary container file data. For each Avro datum, the command emits a morphline record containing the datum as an attachment in the field _attachment_body.

The Avro schema that was used to write the Avro data is retrieved from the Avro container. Optionally, the Avro schema that shall be used for reading can be supplied with a configuration option; otherwise it is assumed to be the same as the writer schema.



Note: Avro uses Schema Resolution if the two schemas are different, e.g. if the reader schema is a subset of the writer schema for the purpose of efficient column projection.

The input stream or byte array is read from the first attachment of the input record.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
readerSchemaFile	null	An optional Avro schema file in JSON format on the local file system to use for reading.
readerSchemaString	null	An optional Avro schema in JSON format given inline to use for reading.

Example usage:

```
# Parse Avro container file and emit a record for each avro object
readAvroContainer {
  # Optionally, require the input to match one of these MIME types:
  # supportedMimeTypes : [avro/binary]

  # Optionally, use this Avro schema in JSON format inline for reading:
  # readerSchemaString : """<json can go here>"""

  # Optionally, use this Avro schema file in JSON format for reading:
  # readerSchemaFile : /path/to/syslog.avsc
}
```

readAvro

The readAvro command (source code) parses containerless Avro. This command is the same as the readAvroContainer command except that the Avro schema that was used to write the Avro data must be explicitly supplied to the readAvro command because it expects raw Avro data without an Avro container and hence without a built-in writer schema.

Optionally, the Avro schema that shall be used for reading can be supplied with a configuration option; otherwise it is assumed to be the same as the writer schema.



Note: Avro uses Schema Resolution if the two schemas are different, e.g. if the reader schema is a subset of the writer schema for the purpose of efficient column projection.

Note: For the readAvro command to work correctly, each Avro event must have been written with the same writer schema by the ingesting app. That is, you cannot parse two Avro events with two different writer schemas A and B within the same readAvro command. The readAvroContainer command doesn't have that limitation, of course, because the writer schema comes embedded inside each Avro container, per the standard Avro container specification.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
readerSchemaFile	null	An optional Avro schema file in JSON format on the local file system to use for reading.
readerSchemaString	null	An optional Avro schema in JSON format given inline to use for reading.
writerSchemaFile	null	The Avro schema file in JSON format that was used to write the Avro data.
writerSchemaString	null	The Avro schema file in JSON format that was used to write the Avro data, given inline.
isJson	false	Whether the Avro input data is encoded as JSON or binary.

Example usage:

```
# Parse Avro and emit a record for each avro object
readAvro {
  # supportedMimeTypes : [avro/binary]
  # readerSchemaString : """<json can go here>"""
  # readerSchemaFile : test-documents/sample-statuses-20120906-141433-subsch
ema.avsc
  # writerSchemaString : """<json can go here>"""
  writerSchemaFile : test-documents/sample-statuses-20120906-141433.avsc
}
```

extractAvroTree

The extractAvroTree command (source code) converts an attached Avro datum to a morphline record by recursively walking the Avro tree and extracting all data into a single morphline record, with fields named by their path in the Avro tree.

The Avro input object is expected to be contained in the field _attachment_body, and typically placed there by an upstream readAvroContainer or readAvro command.

This kind of mapping is useful for simple Avro schemas, but for more complex schemas, this approach may be overly simplistic and expensive.

The command provides the following configuration options:

Property Name	Default	Description
outputFieldPrefix	""	A string to be prepended to each output field name.

Example usage:

```
extractAvroTree {
  outputFieldPrefix : ""
}
```

extractAvroPaths

The extractAvroPaths command (source code) extracts specific values from an Avro object, akin to a simple form of XPath. The command uses zero or more Avro path expressions to extract values from an Avro object.

The Avro input object is expected to be contained in the field _attachment_body, and typically placed there by an upstream readAvroContainer or readAvro command.

Each path expression consists of a record output field name (on the left side of the colon ':') as well as zero or more path steps (on the right hand side), each path step separated by a '/' slash, akin to a simple form of XPath. Avro arrays are traversed with the '[]' notation.

The result of a path expression is a list of objects, each of which is added to the given record output field.

The path language supports all Avro concepts, including such concepts as nested structures, records, arrays, maps, and unions. The path language supports a flatten option that collects the primitives in a subtree into a flat output list.

The command provides the following configuration options:

Property Name	Default	Description
flatten	true	Whether to collect the primitives in a subtree into a flat output list.
paths	[]	Zero or more Avro path expressions.

Example usage:

```
extractAvroPaths {
  flatten : true
  paths : {
    my_price : /price

    my_docId : /docId
    my_links : /links
    my_links_backward : "/links/backward"
    my_links_forward : "/links/forward"
    my_name_language_code : "/name[]/language[]/code"
    my_name_language_country : "/name[]/language[]/country"
    my_name : /name

    /mymapField/foo/label : /mapField/foo/label/
  }
}
```

Alternatively, if the extractAvroPaths and extractAvroTree commands don't fit your needs you can instead implement your own custom morphline command or script a java command config that uses the Generic Avro Java API to arbitrarily traverse and process the Avro tree that is emitted by the readAvroContainer and readAvro commands. For example, along the following lines:

```
{
  readAvroContainer { }
}

{
  java {
    imports : """
      import org.apache.avro.generic.GenericRecord;
      import org.kitesdk.morphline.base.Fields;
      // import com.cloudera.cdk.morphline.base.Fields; // use this for CDK
    """
    code : """
      GenericRecord root = (GenericRecord) record.getFirstValue(Fields.AT
TACHMENT_BODY);
```

```
      GenericRecord links = (GenericRecord) root.get("links"); // traverse v
ia Avro Tree API
      String forwardLinks = links.get("forward").toString(); // traverse via
 Avro Tree API
      record.put("forwardLinks", forwardLinks);
      logger.debug("My output record: {}", record);
      return child.process(record);
    """
    }
  }
}
```

toAvro

The toAvro command (source code) converts a morphline record to an Avro record of Java class org.apache.avro.generic.IndexedRecord.

The conversion supports all Avro concepts, including such concepts as nested structures, records, arrays, maps, and unions.

The Avro output record object is added to the morphline field _attachment_body.

The command provides the following configuration options:

Property Name	Default	Description
schemaFile	null	An optional Avro schema file in JSON format on the local file system to use for writing.
schemaString	null	An optional Avro schema in JSON format given inline to use for writing.
schemaField	null	An optional org.apache.avro.Schema object fetched from the given record input field. One of schemaFile or schemaString or schemaField must be present, but not more than one.
mappings	[]	An optional JSON object containing zero or more mappings from morphline record field names to Avro record field names. Each mapping consists of an Avro output field name (on the left side of the colon ':') as well as a Morphline field name (on the right hand side). Example mapping: avroPrice : morphlinePrice. Any such mappings are optional - by default data is extracted from the morphline fields that carry the same name as the Avro fields defined in the Avro schema.

Example usage:

```
toAvro {
  #schemaFile : /path/to/interop.avsc
  #schemaField : _dataset_descriptor_schema
  schemaString : """
    {
      "type" : "record",
      "name" : "Rating",
      "fields" : [
        {
          "name" : "userId",
          "type" : "int"
        },
        {
          "name" : "rating",
          "type" : ["int","null"]
```

```
        },
        {
          "name" : "reviews",
          "type" : {"type": "array", "items": "string"}
        },
        {
          "name" : "history",
          "type" : ["null", {"type": "map", "values":
                              {"type": "record", "name": "Foo",
                                   "fields": [{"name": "timestamp", "type":
 "long"}]}}]
        }
      ]
    }
  """
  mappings : {
    userId : morphlineUserId
  }
}
```

writeAvroToByteArray

The writeAvroToByteArray command (source code) serializes the Avro records contained in the _attachment_body field into a byte array and replaces the _attachment_body field with that byte array. The records must share an identical Avro schema. Often, the records were originally generated by the toAvro command.

The command provides the following configuration options:

Property Name	Default	Description
format	container	Indicates the type of Avro output format that shall be written. Must be one of container (serialize into a byte array that contains Apache Avro binary container file data) or containerlessBinary (serialize into a byte array that contains Apache Avro without an Avro container and hence without a built-in writer schema) or containerlessJSON (same as containerlessBinary except that Avro output is encoded as JSON).
codec	null	Optional parameter that specifies the compression algorithm to use. Must be one of null or snappy or deflate or bzip2. This parameter only applies if format = container.

Example usage:

```
writeAvroToByteArray {
  format : container
  codec : snappy
}
```

kite-morphlines-json

This maven module contains morphline commands for reading, extracting, and transforming JSON files and JSON objects.

readJson

The readJson command (source code) parses an InputStream or byte array that contains JSON data, using the Jackson library. For each top level JSON object, the command emits a morphline record containing the top level object as an attachment in the field _attachment_body.

The input stream or byte array is read from the first attachment of the input record.

The command provides the following configuration options:

Property Name	Default	Description
outputClass	com.fasterxml.jackson.databind.JsonNode	The fully qualified name of a Java class that Jackson shall convert to.

Example usage:

```
readJson {}
```

Example usage with conversion from JSON to java.util.Map objects:

```
readJson {
  outputClass : java.util.Map
}
```

extractJsonPaths

The extractJsonPaths command (source code) extracts specific values from a JSON object, akin to a simple form of XPath. The command uses zero or more JSON path expressions to extract values from a Jackson JSON object of outputClass com.fasterxml.jackson.databind.JsonNode.

The JSON input object is expected to be contained in the field _attachment_body, and typically placed there by an upstream readJson command with outputClass : com.fasterxml.jackson.databind.JsonNode.

Each path expression consists of a record output field name (on the left side of the colon ':') as well as zero or more path steps (on the right hand side), each path step separated by a '/' slash, akin to a simple form of XPath. JSON arrays are traversed with the '[]' notation.

The result of a path expression is a list of objects, each of which is added to the given record output field.

The path language supports all JSON concepts, including such concepts as nested objects, arrays, etc. The path language supports a flatten option that collects the primitives in a subtree into a flat output list.

The command provides the following configuration options:

Property Name	Default	Description
flatten	true	Whether to collect the primitives in a subtree into a flat output list.
paths	[]	Zero or more JSON path expressions.

Example usage:

```
extractJsonPaths {
  flatten : true
  paths : {
    my_price : /price

    my_docId : /docId
    my_links : /links
    my_links_backward : "/links/backward"
    my_links_forward : "/links/forward"
    my_name_language_code : "/name[]/language[]/code"
```

```
    my_name_language_country : "/name[]/language[]/country"
    my_name : /name
  }
}
```

Alternatively, if the extractJsonPaths command doesn't fit your needs you can instead implement your own Custom Morphline Command or script a java command config that uses the com.fasterxml.jackson.databind.JsonNode Java API to arbitrarily traverse and process the Jackson Json tree that is emitted by the readJson command. For example, along the following lines:

```
{
  readJson { }
}
{
  java {
    imports : """
      import com.fasterxml.jackson.databind.JsonNode;
      import org.kitesdk.morphline.base.Fields;
      // import com.cloudera.cdk.morphline.base.Fields; // use this for CDK
    """
    code : """
      JsonNode rootNode = (JsonNode) record.getFirstValue(Fields.ATTACHMENT_
BODY);
      String forwardLinks = rootNode.get("links").get("forward").asText()
; // traverse via Jackson Tree API
      record.put("forwardLinks", forwardLinks);
      logger.debug("My output record: {}", record);
      return child.process(record);
    """
    }
  }
}
```

kite-morphlines-hadoop-core

downloadHdfsFile

The downloadHdfsFile command (source code) downloads, on startup, zero or more files or directory trees from HDFS to the local file system. These files are typically static configuration files that are required by downstream morphline commands, e.g. Avro schema files, XML join tables, grok dictionaries, etc. Storing such configuration files in HDFS can help with consistent centralized configuration management across a set of cluster nodes.

The output directory on the local file system defaults to the current working directory of the current process. If the effective output file or directory already exists it will be deleted and overwritten.

The command provides the following configuration options:

Property Name	Default	Description
inputFiles		The HDFS files or directories to download, in the form of a list of HDFS URIs.
outputDir	"."	The relative or absolute path of the destination directory on the local file system. Parent directories of that directory will be created automatically. Defaults to the current working directory of the current process.

Example usage:

```
downloadHdfsFile {
  inputFiles : ["hdfs://c2202.mycompany.com/user/foo/configs/sample-schem
a.avsc"]
  outputDir : "myconfigs"
}
```

openHdfsFile

The openHdfsFile command (source code) opens an HDFS file for read and returns a corresponding Java InputStream.

The morphline record input field _attachment_body must contain the HDFS Path of the file to read. The command replaces the HDFS Path in this field with the corresponding Java InputStream. Said InputStream can then be parsed with other commands, such as readLine on page 13 or similar.

The command automatically handles gzip files if the file path ends with the ".gz" file name extensions.

Example usage:

```
openHdfsFile {}
```

kite-morphlines-hadoop-parquet-avro

This maven module contains morphline commands for handling Hadoop Avro Parquet files.

readAvroParquetFile

The readAvroParquetFile command (source code) parses a Hadoop Parquet file and emits a morphline record for each contained Avro datum.

The morphline record input field file_upload_url must contain the HDFS Path of the Parquet file to read. (This field is already provided out of the box with MapReduceIndexerTool).

For each Avro datum, the command emits a morphline record containing the datum as an attachment in the field _attachment_body. Typically, the emitted Avro datum is further post-processed with downstream commands such as extractAvroPaths.

Optionally, an Avro schema that shall be used for projecting parquet columns can be supplied with a configuration option.

The command provides the following configuration options:

Property Name	Default	Description
decimalConversionEnabled	false	When set to true, decimal Parquet data is correctly read instead of returning raw bytes.
projectionSchemaFile	null	An optional Avro schema file in JSON format on the local file system to use for projection. This Avro schema is converted to a parquet schema before applying the projection.
projectionSchemaString	null	An optional Avro schema in JSON format given inline to use for projection. This Avro schema is converted to a parquet schema before applying the projection.

Property Name	Default	Description
readerSchemaFile	null	This optional parameter is available in CDH 5.0 and beyond. This optional parameter specifies an Avro schema file in JSON format on the local file system to use for reading, as discussed above.
readerSchemaString	null	This optional parameter is available in CDH 5.0 and beyond. This optional parameter specifies an optional Avro schema in JSON format given inline to use for reading. Has identical behaviour as the readerSchemaFile parameter described above.

Example usage:

```
readAvroParquetFile {
  # Optionally, use this Avro schema in JSON format inline for projection:
  # projectionSchemaString : """<json can go here>"""

  # Optionally, use this Avro schema file in JSON format for projection:
  # projectionSchemaFile : /path/to/syslog.avsc
}
```

kite-morphlines-hadoop-rcfile

readRCFile

The readRCFile command (source code) parses an Apache Hadoop RCFile. An RCFile can be read Row-wise or Column-wise, as follows:

- Row-wise: One morphline record is emitted for each row in the RCFile. Each record will contain fields for all the columns configured in the columns parameter. For example, with an RCFile with 10 rows and 5 columns, Row-wise mode would emit 10 morphline records.
- Column-wise: For every row-split (block) in the RC File, Emits one record for each column specified in the colu mns parameter. This record will contain a list of row values for that column as a list. The order of columns is as specified in the columns parameter. For example, with an RCFile with 10 rows and 5 columns, Column-wise mode would emit 5 morphline records each with a list of 10 values assuming there are no row-splits.

The InputStream of the RCFile is read from the _attachment_body field of the input record. Optionally, the name of the RCFile is read from the _attachment_name field of the input record. Providing a name for the InputStream will, in case of errors, result in error messages containing said name for better debugging and diagnostics.

The command automatically handles compressed RCFiles.

The command provides the following configuration options:

Property Name	Default	Description
readMode	row	Valid values: row or column. Defines the reading strategy. Affects the structure and number of output records that will be emitted, as discussed above.
includeMetaData	false	Whether or not the RCFile metadata shall be included in the output record.
columns	n/a	A list of column configurations. Since an RCFile does not store meta information about the columns itself, this configuration is necessary to read the RCFile.

The columns configuration has the following configuration options:

Property Name	Default	Description
inputField	n/a	Non-negative integer index of the RCFile column to read.
outputField	n/a	The name of the field to add output values to. The output record will have the value of the column added to this field.
writableClass	n/a	Fully Qualified Class Name of a sub-class of org.apache.hadoop.io.Writable. Instances of this class are used to read the value of the column bytes, and said instances are added to the outputField. For example org.apache.hadoop.io.Text or org.apache.hadoop.io.LongWritable or org.apache.hadoop.hive.serde2.columnar.BytesRefWritable

Example usage:

```
readRCFile {
  readMode: row
  includeMetaData: false
  columns: [
    {
      inputField: 0
      outputField: name
      writableClass: "org.apache.hadoop.io.Text"
    }
    {
      inputField: 3
      outputField: age
      writableClass: "org.apache.hadoop.io.LongWritable"
    }
    {
      inputField: 1000
      outputField: photo
      writableClass: "org.apache.hadoop.hive.serde2.columnar.BytesRefWrita
ble"
    }
  ]
}
```

kite-morphlines-hadoop-sequencefile

readSequenceFile

The readSequenceFile command (source code) parses an Apache Hadoop SequenceFile and emits a morphline record for each contained key-value pair. The sequence file is read from the input stream of the first attachment of the record.

The command automatically handles Record-Compressed and Block-Compressed SequenceFiles.

The command provides the following configuration options:

Property Name	Default	Description
keyField	_attachment_name	The name of the output field to store the SequenceFile Record key.

Property Name	Default	Description
valueField	_attachment_body	The name of the output field to store the SequenceFile Record value.

Example usage:

```
readSequenceFile {
  keyField : "key"
  valueField : "value"
}
```

kite-morphlines-maxmind

geoIP

The geoIP command (source code) returns Geolocation information for a given IP address, using an efficient in-memory Maxmind database lookup. The command stores a corresponding Jackson JsonNode Java object into the _attachment_body record field. The most recent version of the Maxmind GeoLite2 database can be downloaded as a flat data file from Maxmind.

Often, the geoIP command is combined with commands such as extractJsonPaths.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field that contains zero or more IP addresses.
database	GeoLite2-City.mmdb	The relative or absolute path of a Maxmind database file on the local file system. Example: /path/to/GeoLite2-City.mmdb

Example usage:

```
# extract geolocation info into a Jackson JsonNode Java object
# and store it into the _attachment_body field:
geoIP {
  inputField : ip
  database : "target/test-classes/GeoLite2-City.mmdb"
}

# extract parts of the geolocation info from the Jackson JsonNode Java
# object contained in the _attachment_body field and store the parts in
# the given record output fields:
extractJsonPaths {
  flatten : false
  paths : {
    /country/iso_code : /country/iso_code
    /country/names/en : /country/names/en
    /country/names/zh-CN : /country/names/zh-CN
    "/subdivisions[]/names/en" : "/subdivisions[]/names/en"
    "/subdivisions[]/iso_code" : "/subdivisions[]/iso_code"
    /city/names/en : /city/names/en
    /postal/code : /postal/code
    /location/latitude : /location/latitude
    /location/longitude : /location/longitude
    /location/latitude_longitude : /location/latitude_longitude
    /location/longitude_latitude : /location/longitude_latitude
```

```
  }
}
```

Example geoIP JSON output with extractJsonPaths:

Input: ip: 128.101.101.101

Expected output:

```
ip: 128.101.101.101
/country/iso_code: US
/country/names/en: United States
/country/names/zh-CN: ##
/subdivisions[]/names/en: Minnesota
/subdivisions[]/iso_code: MN
/city/names/en: Minneapolis
/postal/code: 55455
/location/latitude: 44.9733
/location/longitude: -93.2323
/location/latitude_longitude: 44.9733,-93.2323
/location/longitude_latitude: -93.2323,44.9733
```

Example geoIP JSON output:

Input: ip: 128.101.101.101

Expected output:

```
{
   "city":{
      "geoname_id":5037649,
      "names":{
         "de":"Minneapolis",
         "en":"Minneapolis",
         "es":"Mineápolis",
         "fr":"Minneapolis",
         "ja":"######",
         "pt-BR":"Minneapolis",
         "ru":"###########",
         "zh-CN":"######"
      }
   },
   "continent":{
      "code":"NA",
      "geoname_id":6255149,
      "names":{
         "de":"Nordamerika",
         "en":"North America",
         "es":"Norteamérica",
         "fr":"Amérique du Nord",
         "ja":"#####",
         "pt-BR":"América do Norte",
         "ru":"######## #######",
         "zh-CN":"###"
      }
   },
   "country":{
      "geoname_id":6252001,
      "iso_code":"US",
      "names":{
         "de":"USA",
         "en":"United States",
         "es":"Estados Unidos",
         "fr":"États-Unis",
```

```
            "ja":"#######",
            "pt-BR":"Estados Unidos",
            "ru":"###",
            "zh-CN":"##"
         }
      },
      "location":{
         "latitude":44.9733,
         "longitude":-93.2323,
         "metro_code":"613",
         "time_zone":"America/Chicago"
         "latitude_longitude":"44.9733,-93.2323"
         "longitude_latitude":"-93.2323,44.9733"
      },
      "postal":{
         "code":"55455"
      },
      "registered_country":{
         "geoname_id":6252001,
         "iso_code":"US",
         "names":{
            "de":"USA",
            "en":"United States",
            "es":"Estados Unidos",
            "fr":"États-Unis",
            "ja":"#######",
            "pt-BR":"Estados Unidos",
            "ru":"###",
            "zh-CN":"##"
         }
      },
      "subdivisions":[
         {
            "geoname_id":5037779,
            "iso_code":"MN",
            "names":{
               "en":"Minnesota",
               "es":"Minnesota",
               "ja":"#####",
               "ru":"#########"
            }
         }
      ]
   }
```


kite-morphlines-metrics-servlets

registerJVMMetrics

The registerJVMMetrics command (source code) registers metrics that are related to the Java Virtual Machine with the MorphlineContext of the morphline. For example, this includes metrics for garbage collection events, buffer pools, threads and thread deadlocks.

Often, the registerJVMMetrics command is combined with commands such as startReportingMetricsToHTTP or startReportingMetricsToJMX or startReportingMetricsToSLF4J or startReportingMetricsToCSV.

Example usage:

```
registerJVMMetrics {}
```

startReportingMetricsToHTTP

Status: EXPERIMENTAL

The startReportingMetricsToHTTP command (source code) exposes liveness status, health check status, metrics state and thread dumps via a set of HTTP URLs served by Jetty, using the AdminServlet.

On startup, a Jetty HTTP server is created that listens on a configurable port. If an HTTP server isn't required for your use case, and reporting metrics to JMX (or SLF4J or CSV) is sufficient, consider command such as startReportingMetricsToJMX or startReportingMetricsToSLF4J or startReportingMetricsToCSV.

Often, the startReportingMetricsToHTTP command is combined with the registerJVMMetrics command.

The following HTTP URLs are provided:

URL Path	Servlet	Description
/	AdminServlet	an HTML admin menu, for example at http://foo.com:8080/, with links to the following servlets:
/ping	PingServlet	PingServlet responds to GET requests with a text/plain/200 OK response of pong. This is useful for determining liveness for load balancers, etc.
/healthcheck	HealthCheckServlet	HealthCheckServlet responds to GET requests by running all the health checks registered with the morphline context, and returning 501 Not Implemented if no health checks are registered, 200 OK if all pass, or 500 Internal Service Error if one or more fail. The results are returned as a human-readable text/plain JSON entity.
/metrics	MetricsServlet	MetricsServlet exposes the state of the metrics registered with the morphline context as a JSON object.
/threads	ThreadDumpServlet	ThreadDumpServlet responds to GET requests with a text/plain representation of all the live threads in the JVM, their states, their stack traces, and the state of any locks they may be waiting for.

The command provides the following configuration options:

Property Name	Default	Description
port	8080	The port on which the HTTP server shall listen.
defaultDurationUnit	milliseconds	Report output durations in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days.
defaultRateUnit	seconds	Report output rates in the given time unit. One of nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days. Example output: events/second

Example startReportingMetricsToHTTP Usage:

```
startReportingMetricsToHTTP {
  port : 8080
}
```

Example startReportingMetricsToHTTP ping output:

Here is the response to an HTTP GET to http://localhost:8080/ping for liveness check:

```
pong
```

Example startReportingMetricsToHTTP healthcheck output:

Example output of running healthchecks via a HTTP GET to http://localhost:8080/healthcheck

```
{"deadlocks":{"healthy":true}}
```

Example startReportingMetricsToHTTP metrics output:

For an example on how to update user defined custom metrics such as counters, meters, timers and histograms, see the java command. Here is an example output of the JSON metrics reported by an HTTP GET to http://localhost:8080/metrics?pretty=true

```
{
 "version" : "3.0.0",
 "gauges" : {
   "jvm.gc.ConcurrentMarkSweep.count" : {
     "value" : 0
   },
   "jvm.gc.ConcurrentMarkSweep.time" : {
     "value" : 0
   },
   "jvm.gc.ParNew.count" : {
     "value" : 4
   },
   "jvm.gc.ParNew.time" : {
     "value" : 29
   },
   "jvm.memory.heap.committed" : {
     "value" : 85000192
   },
   "jvm.memory.heap.init" : {
     "value" : 0
   },
   "jvm.memory.heap.max" : {
     "value" : 129957888
   },
   "jvm.memory.heap.usage" : {
     "value" : 0.1319703810514372
   },
   "jvm.memory.heap.used" : {
     "value" : 17150592
   },
   "jvm.memory.non-heap.committed" : {
     "value" : 24711168
   },
   "jvm.memory.non-heap.init" : {
     "value" : 24317952
   },
   "jvm.memory.non-heap.max" : {
     "value" : 136314880
   },
   "jvm.memory.non-heap.usage" : {
     "value" : 0.16856530996469352
   },
   "jvm.memory.non-heap.used" : {
     "value" : 22978104
   },
   "jvm.memory.pools.CMS-Old-Gen.usage" : {
     "value" : 0.05705025643464222
```

```
    },
    "jvm.memory.pools.CMS-Perm-Gen.usage" : {
      "value" : 0.25629341311571074
    },
    "jvm.memory.pools.Code-Cache.usage" : {
      "value" : 0.018703460693359375
    },
    "jvm.memory.pools.Par-Eden-Space.usage" : {
      "value" : 0.5095581972509399
    },
    "jvm.memory.pools.Par-Survivor-Space.usage" : {
      "value" : 0.9115804036458334
    },
    "jvm.memory.total.committed" : {
      "value" : 109711360
    },
    "jvm.memory.total.init" : {
      "value" : 24317952
    },
    "jvm.memory.total.max" : {
      "value" : 266272768
    },
    "jvm.memory.total.used" : {
      "value" : 40248344
    },
    "jvm.threads.blocked.count" : {
      "value" : 2
    },
    "jvm.threads.count" : {
      "value" : 22
    },
    "jvm.threads.daemon.count" : {
      "value" : 4
    },
    "jvm.threads.deadlocks" : {
      "value" : [ ]
    },
    "jvm.threads.new.count" : {
      "value" : 0
    },
    "jvm.threads.runnable.count" : {
      "value" : 10
    },
    "jvm.threads.terminated.count" : {
      "value" : 0
    },
    "jvm.threads.timed_waiting.count" : {
      "value" : 8
    },
    "jvm.threads.waiting.count" : {
      "value" : 2
    }
  },
  "counters" : {
    "myMetrics.myCounter" : {
      "count" : 1
    }
  },
  "histograms" : {
    "myMetrics.myHistogram" : {
      "count" : 1,
      "max" : 100,
      "mean" : 100.0,
      "min" : 100,
```

```
    "p50" : 100.0,
    "p75" : 100.0,
    "p95" : 100.0,
    "p98" : 100.0,
    "p99" : 100.0,
    "p999" : 100.0,
    "stddev" : 0.0
  }
},
"meters" : {
  "morphline.java.numNotifyCalls" : {
    "count" : 1,
    "m15_rate" : 0.16929634497812282,
    "m1_rate" : 0.19779007785878447,
    "m5_rate" : 0.1934432200964012,
    "mean_rate" : 0.06666243138019297,
    "units" : "events/second"
  },
  "morphline.java.numProcessCalls" : {
    "count" : 1,
    "m15_rate" : 0.16929634497812282,
    "m1_rate" : 0.19779007785878447,
    "m5_rate" : 0.1934432200964012,
    "mean_rate" : 0.06666191145031655,
    "units" : "events/second"
  },
  "morphline.logDebug.numNotifyCalls" : {
    "count" : 3,
    "m15_rate" : 0.5078890349343685,
    "m1_rate" : 0.5933702335763534,
    "m5_rate" : 0.5803296602892035,
    "mean_rate" : 0.1999690981087056,
    "units" : "events/second"
  },
  "morphline.logDebug.numProcessCalls" : {
    "count" : 3,
    "m15_rate" : 0.5078890349343685,
    "m1_rate" : 0.5933702335763534,
    "m5_rate" : 0.5803296602892035,
    "mean_rate" : 0.19996765856402157,
    "units" : "events/second"
  },
  "morphline.logWarn.numNotifyCalls" : {
    "count" : 2,
    "m15_rate" : 0.33859268995624564,
    "m1_rate" : 0.39558015571756894,
    "m5_rate" : 0.3868864401928024,
    "mean_rate" : 0.11979779569659962,
    "units" : "events/second"
  },
  "morphline.logWarn.numProcessCalls" : {
    "count" : 2,
    "m15_rate" : 0.33859268995624564,
    "m1_rate" : 0.39558015571756894,
    "m5_rate" : 0.3868864401928024,
    "mean_rate" : 0.11979702071997292,
    "units" : "events/second"
  },
  "morphline.pipe.numNotifyCalls" : {
    "count" : 1,
    "m15_rate" : 0.16929634497812282,
    "m1_rate" : 0.19779007785878447,
    "m5_rate" : 0.1934432200964012,
    "mean_rate" : 0.05774150456461029,
```

```
      "units" : "events/second"
    },
    "morphline.pipe.numProcessCalls" : {
      "count" : 1,
      "m15_rate" : 0.16929634497812282,
      "m1_rate" : 0.19779007785878447,
      "m5_rate" : 0.1934432200964012,
      "mean_rate" : 0.05766282424671017,
      "units" : "events/second"
    },
    "morphline.registerJVMMetrics.numNotifyCalls" : {
      "count" : 1,
      "m15_rate" : 0.16929634497812282,
      "m1_rate" : 0.19779007785878447,
      "m5_rate" : 0.1934432200964012,
      "mean_rate" : 0.059902898599428295,
      "units" : "events/second"
    },
    "morphline.registerJVMMetrics.numProcessCalls" : {
      "count" : 1,
      "m15_rate" : 0.16929634497812282,
      "m1_rate" : 0.19779007785878447,
      "m5_rate" : 0.1934432200964012,
      "mean_rate" : 0.059902518235973874,
      "units" : "events/second"
    },
    "morphline.startReportingMetricsToHTTP.numNotifyCalls" : {
      "count" : 3,
      "m15_rate" : 0.5078890349343685,
      "m1_rate" : 0.5933702335763534,
      "m5_rate" : 0.5803296602892035,
      "mean_rate" : 0.19066147711547488,
      "units" : "events/second"
    },
    "morphline.startReportingMetricsToHTTP.numProcessCalls" : {
      "count" : 3,
      "m15_rate" : 0.5078890349343685,
      "m1_rate" : 0.5933702335763534,
      "m5_rate" : 0.5803296602892035,
      "mean_rate" : 0.19066014422550162,
      "units" : "events/second"
    },
    "myMetrics.myMeter" : {
      "count" : 1,
      "m15_rate" : 0.18400888292586468,
      "m1_rate" : 0.19889196960097935,
      "m5_rate" : 0.1966942907643235,
      "mean_rate" : 0.0698670918312095,
      "units" : "events/second"
    }
  },
  "timers" : {
    "myMetrics.myTimer" : {
      "count" : 1,
      "max" : 1.4000000000000001E-5,
      "mean" : 1.4000000000000001E-5,
      "min" : 1.4000000000000001E-5,
      "p50" : 1.4000000000000001E-5,
      "p75" : 1.4000000000000001E-5,
      "p95" : 1.4000000000000001E-5,
      "p98" : 1.4000000000000001E-5,
      "p99" : 1.4000000000000001E-5,
      "p999" : 1.4000000000000001E-5,
      "stddev" : 0.0,
```

```
      "m15_rate" : 0.18400888292586468,
      "m1_rate" : 0.19889196960097935,
      "m5_rate" : 0.1966942907643235,
      "mean_rate" : 0.0698417274708746,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "myMetrics.myTimer2" : {
      "count" : 1,
      "max" : 0.0,
      "mean" : 0.0,
      "min" : 0.0,
      "p50" : 0.0,
      "p75" : 0.0,
      "p95" : 0.0,
      "p98" : 0.0,
      "p99" : 0.0,
      "p999" : 0.0,
      "stddev" : 0.0,
      "m15_rate" : 0.18400888292586468,
      "m1_rate" : 0.19889196960097935,
      "m5_rate" : 0.1966942907643235,
      "mean_rate" : 0.0698418152725891,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
  }
}
```

Example startReportingMetricsToHTTP thread dump output:

Here is the response to an HTTP GET to http://localhost:8080/threads for a thread dump:

```
main id=1 state=TIMED_WAITING
    at java.lang.Thread.sleep(Native Method)
    at org.kitesdk.morphline.metrics.servlets.HttpMetricsMorphlineTest.te
stBasic(HttpMetricsMorphlineTest.java:51)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl
.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcc
essorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(Framew
orkMethod.java:45)
    at org.junit.internal.runners.model.ReflectiveCallable.run(Reflective
Callable.java:15)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(Framework
Method.java:42)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMet
hod.java:20)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores
.java:28)
    at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.
java:30)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassR
unner.java:68)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4Class
Runner.java:47)
    at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
```

```
    at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:300)
    at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUn
it4TestReference.java:50)
    at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecuti
on.java:38)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(Remot
eTestRunner.java:467)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(Rem
oteTestRunner.java:683)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTest
Runner.java:390)
    at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTe
stRunner.java:197)

Reference Handler id=2 state=WAITING
    - waiting on <0x7418e252> (a java.lang.ref.Reference$Lock)
    - locked <0x7418e252> (a java.lang.ref.Reference$Lock)
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:485)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:116)

... and so on
```

kite-morphlines-protobuf

This maven module contains morphline commands for reading, extracting, and transforming protocol buffer objects.

readProtobuf

The readProtobuf command (source code) parses an InputStream or byte array that contains protobuf data. For each protobuf object, the command emits a morphline record containing the top level object as an attachment in the field _attachment_body.

The input stream or byte array is read from the first attachment of the input record.

The command provides the following configuration options:

Property Name	Default	Description
protobufClass	[]	The fully qualified name of a Java class that was generated by the protoc compiler. This Java class contains protobuf message definitions.
outputClass	[]	The name of an inner Java class (within protobufClass) for deserializing data to.

Example usage:

```
readProtobuf {
  protobufClass : org.kitesdk.morphline.protobuf.Protos
  outputClass : RepeatedLongs
}
```

And protobuf schema for protoc:

```
option java_package = "org.kitesdk.morphline.protobuf";
option java_outer_classname = "Protos";
option java_generate_equals_and_hash = true;
```

```
option optimize_for = SPEED;

message RepeatedLongs {
  repeated sint64 longVal = 1;
}
message Complex {

  message Name {
    optional uint32 intVal = 1;
    optional uint64 longVal = 2;
    optional double doubleVal = 3;
    optional float floatVal = 4;
    repeated string stringVal = 5;
    optional RepeatedLongs repeatedLong = 6;
  }

  message Link {
    repeated string language = 1;
    required string url = 2;
  }

  enum Type {
    QUERY = 1;
    UPDATE = 2;
  }

  required sint32 docId = 1;
  required Name name = 2;
  repeated Link link = 3;
  required Type type = 4;
}
```

extractProtobufPaths

The extractProtobufPaths command (source code) extracts specific values from a protobuf object, akin to a simple form of XPath. The command uses zero or more path expressions to extract values from a protobuf instance object.

The protobuf input object is expected to be contained in the field _attachment_body, and typically placed there by an upstream readProtobuf command.

Each path expression consists of a record output field name (on the left side of the colon ':') as well as zero or more path steps (on the right hand side), each path step separated by a '/' slash, akin to a simple form of XPath. Repeated values(Lists) are traversed with the '[]' notation.

The result of a path expression is a list of objects, each of which is added to the given record output field. To check if the property is set and serialized in protobuf message is used the has<PropertyName>() method and if the property isn't set then there is no result of a path expression. That means the output field is not passed to next command.

The command provides the following configuration options:

Property Name	Default	Description
objectExtractMethod	toByteArray	Java method that is called on the protobuf object to get a value to pass to the next command if the type of value on a path is a protobuf object. Options are: toByteArray - the "toByteArray()" method is called to get serialized bytes from the protobuf object. toString - the "toString()" method is called to get a String representation of a protobuf object. none - no method is called and the whole protobuf object is passed to the next command.

Property Name	Default	Description
enumExtractMethod	name	Java method that is called to get a value to pass to the next command if the type of value on a path is an enum object. Options are: name - the "name()" method is called to get a String representation of the enum object. getNumber - the "getNumber()" method is called to get an int representation of the enum object, none - no method is called and the whole enum object is passed to the next command.
paths	[]	Zero or more protobuf path expressions.

Example usage:

```
extractProtobufPaths {
  objectExtractMethod : toByteArray
  enumExtractMethod : name
  paths : {
    "docId" : "/docId"
    "name" : "/name"
    "intVal" : "/name/intVal"
    "longVal" : "/name/longVal"
    "doubleVal" : "/name/doubleVal"
    "floatVal" : "/name/floatVal"
    "stringVals" : "/name/stringVal[]"
    "longVals" : "/name/repeatedLong/longVal[]"
    "links" : "/link[]"
    "languages" : "/link[]/language"
    "urls" : "/link[]/url"
    "type" : "/type"
  }
}
```

kite-morphlines-tika-core

This maven module contains morphline commands for autodetecting MIME types from binary data. Depends on tika-core.

detectMimeType

The detectMimeType command (source code) uses Apache Tika to autodetect the MIME type of the first attachment from the binary data. The detected MIME type is assigned to the _attachment_mimetype field.

The command provides the following configuration options:

Property Name	Default	Description
includeDefaultMimeTypes	true	Whether to include the Tika default MIME types file that ships embedded in tika-core.jar (see http://github.com/apache/tika/blob/trunk/tika-core/src/main/resources/org/apache/tika/mime/tika-mimetypes.xml)
mimeTypesFiles	[]	The relative or absolute path of zero or more Tika custom-mimetypes.xml files to include.
mimeTypesString	null	The content of an optional custom-mimetypes.xml file embedded directly inside of this morphline configuration file.

Property Name	Default	Description
preserveExisting	true	Whether to preserve the _attachment_mimetype field value if one is already present.
includeMetaData	false	Whether to pass the record fields to Tika to assist in MIME type detection.
excludeParameters	true	Whether to remove MIME parameters from output MIME type.

Example usage:

```
detectMimeType {
  includeDefaultMimeTypes : false
  #mimeTypesFiles : [src/test/resources/custom-mimetypes.xml]
  mimeTypesString :
    """
    <mime-info>
      <mime-type type="text/space-separated-values">
        <glob pattern="*.ssv"/>
      </mime-type>

      <mime-type type="avro/binary">
        <magic priority="50">
          <match value="0x4f626a01" type="string" offset="0"/>
        </magic>
        <glob pattern="*.avro"/>
      </mime-type>

      <mime-type type="mytwittertest/json+delimited+length">
        <magic priority="50">
          <match value="[0-9]+(\r)?\n\\{&quot;" type="regex" offset="0:16"/>
        </magic>
      </mime-type>
    </mime-info>
    """
}
```

kite-morphlines-tika-decompress

This maven module contains morphline commands for decompressing and unpacking files. Depends on tika-core and commons-compress.

decompress

The decompress command (source code) decompresses the first attachment, and supports gzip and bzip2 format.

Example usage:

```
decompress {}
```

unpack

The unpack command (source code) unpacks the first attachment, and supports tar, zip, and jar format. The command emits one record per contained file.

Example usage:

```
unpack {}
```

kite-morphlines-saxon

This maven module contains morphline commands for reading, extracting and transforming XML and HTML with XPath, XQuery and XSLT.

convertHTML

The convertHTML command (source code) converts any HTML to XHTML, using the TagSoup Java library.

Instead of parsing well-formed or valid XML, this command parses HTML as it is found in the wild: poor, nasty and brutish, though quite often far from short. TagSoup (and hence this command) is designed for people who have to process this stuff using some semblance of a rational application design. By providing this converter, it allows standard XML tools to be applied to even the worst malformed HTML.

The command reads an InputStream or byte array from the first attachment (field _attachment_body) of the input record, parses it as HTML and replaces the field with UTF-8 encoded XHTML.

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
charset	null	The character encoding to use for parsing input, for example, UTF-8. If none is specified the charset specified in the _attachment_charset input field is used instead.
noNamespaces	true	A value of false indicates namespace URIs and unprefixed local names for element and attribute names will be available.
noCDATA	false	A value of true indicates that the parser will treat CDATA elements specially.
noBogons	false	A value of true indicates that the parser will ignore unknown elements.
emptyBogons	false	A value of true indicates that the parser will give unknown elements a content model of EMPTY; a value of false, a content model of ANY.
noRootBogons	false	A value of true indicates that the parser will allow unknown elements to be the root element.
noDefaultAttributes	false	A value of true indicates that the parser will return default attribute values for missing attributes that have default values.
noColons	false	A value of true indicates that the parser will translate colons into underscores in names.
noRestart	false	A value of true indicates that the parser will attempt to restart the restartable elements.
suppressIgnorableWhitespace	true	A value of false indicates that the parser will transmit whitespace in element-only content via the SAX ignorableWhitespace callback.

Example usage:

```
convertHTML {
  charset : UTF-8
}
```

xquery

The xquery command (source code) parses an InputStream that contains an XML document and runs the given W3C XQuery over the XML document, using the Saxon Java library. For each item in the query result sequence, the command emits a corresponding morphline record.

The command reads an InputStream or byte array from the first attachment (field _attachment_body) of the input record.

Per the W3C specs, every valid XPath (e.g. //tweets/tweet[@color='blue']) is also a valid XQuery. If you are comfortable with XPath you are already almost there.

An XQuery result sequence contains zero or more items such as element nodes, attribute nodes, text nodes, atomic values, etc. For each item in the query result sequence, the morphline command converts the item to a record and pipes that record to the next morphline command. For an attribute node the attribute's XPath string value is filled into the record field named after the attribute name. For an element node the attributes and children of the element are treated as follows: The XPath string value of the attribute or child is filled into the record field named after the child's name.

For example, in order to generate two morphline records, the first morphline record with a firstName field that contains Joe, as well as a lastName field that contains Bubblegum, and the second morphline record with a firs tName field that contains Alice, as well as a lastName field that contains Pellegrino, your xquery command should be formulated such that it outputs two XML fragments like this:

```
<record>
 <firstName>Joe</firstName>
 <lastName>Bubblegum</lastName>
</record>

<record>
 <firstName>Alice</firstName>
 <lastName>Pellegrino</lastName>
</record>
```

The xquery command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
languageVersion	"1.0"	Must be "1.0" for XQuery 1.0 or "3.0" for XQuery 3.0.
features	null	An optional JSON object containing zero or more name-value pairs that represent configuration properties for Saxon features.
extensionFunctions	[]	An optional list of Java class names that implement custom Saxon extension functions. Each such Java class must implement net.sf.saxon.s9api.ExtensionFunction as described in the Saxon documentation.
fragments	n/a	An array containing exactly one fragment JSON object, as described below.
Each fragment provides the following configuration options:		

Property Name	Default	Description
fragmentPath	n/a	Currently must be "/"
externalVariables	null	An optional JSON object containing zero or more name-value pairs that are bound and passed in as external variables to the query. Example: myVar : "hello world"
externalFileVariables	null	An optional JSON object containing zero or more name-path pairs that refer to XML files on the local file system, and are bound and passed in as external variables to the query. These files are loaded once on program startup and subsequently remain memory resident across queries. This can be used for efficient joins where the join table is static and fits into main memory. Example: myDoc : src/test/resources/testdocuments/helloworld.xml
queryFile	null	A relative or absolute path of a local file from which to load the query.
queryString	null	An inline string from which to load the query. One of queryFile or queryString must be present, but not both. Example: """/tweets/tweet"""



Note: The morphline configuration file is implemented using the HOCON format (Human Optimized Config Object Notation). HOCON is basically JSON slightly adjusted for the configuration file use case. HOCON syntax is defined at HOCON github page and as such, multi-line strings are similar to Python or Scala, using triple quotes. If the three-character sequence """ appears, then all Unicode characters until a closing """ sequence are used unmodified to create a string value.

Example usage:

```
xquery {
  fragments : [
    {
      fragmentPath : "/"
      externalVariables : {
        myVariable : "hello world"
      }
      queryString : """
        (: Example test xquery :)

        declare variable $myVariable as xs:string external;

        for $tweet in /tweets/tweet
        return
        <record>
          {$tweet/@id}
          {$tweet/user/@screen_name}
          <myStatusCounts>{string($tweet/user/@statuses_count)}</myStatusC
ounts>
          <text>{$tweet/text}</text>
          <greeting>{$myVariable}</greeting>
        </record>
      """
    }
  ]
}
```

Here is an example output record for the query above:

```
id:11111112
```

```
screen_name:fake_user1
myStatusCounts:11111
text:Come, see new hot tub under Redwood Tree!
greeting:hello world
```

More example usage:

```
xquery {
  fragments : [
    {
      fragmentPath : "/"
      queryString : """
        (: Example xquery :)
        for $req in /request
        return
          <record>
            <date> { string($req/data/agreementDate) } </date>
            <tradeId> { string($req/trade/@tradeId) } </tradeId>

            <partyId>
              {
                for $keyword in $req/trade/keyWords/keyword
                  where $keyword/name = "memberId"
                    return string($keyword/value)
              }
            </partyId>
            <fullText> { $req } </fullText>
          </record>
      """
    }
  ]
}
```

More examples can be found in the unit tests.

Here is an example extension function along with a corresponding example xquery.

For more background, see resources such as the XQuery Primer and XQuery FLOWR Tutorial and XQuery: A Guided Tour and Wikipedia.

xslt

The xslt command (source code) parses an InputStream that contains an XML document and runs the given W3C XSL Transform over the XML document, using the Saxon Java library. For each item in the query result sequence, the command emits a corresponding morphline record.

The command reads an InputStream or byte array from the first attachment (field _attachment_body) of the input record.

An XSLT result sequence contains zero or more items such as element nodes, attribute nodes, text nodes, atomic values, etc. For each item in the query result sequence, the morphline command converts the item to a record and pipes that record to the next morphline command. For an attribute node the attribute's XPath string value is filled into the record field named after the attribute name. For an element node the attributes and children of the element are treated as follows: The XPath string value of the attribute or child is filled into the record field named after the child's name.

For example, in order to generate two morphline records, the first morphline record with a firstName field that contains Joe, as well as a lastName field that contains Bubblegum, and the second morphline record with a firs tName field that contains Alice, as well as a lastName field that contains Pellegrino, your xslt command should be formulated such that it outputs two XML fragments like this:

```
<record>
```

```
 <firstName>Joe</firstName>
 <lastName>Bubblegum</lastName>
</record>

<record>
 <firstName>Alice</firstName>
 <lastName>Pellegrino</lastName>
</record>
```

The command provides the following configuration options:

Property Name	Default	Description
supportedMimeTypes	null	Optionally, require the input record to match one of the MIME types in this list.
features	null	An optional JSON object containing zero or more name-value pairs that represent configuration properties for Saxon features.
extensionFunctions	[]	An optional list of Java class names that implement custom Saxon extension functions. Each such Java class must implement net.sf.saxon.s9api.ExtensionFunction as described in the Saxon documentation.
fragments	n/a	An array containing exactly one fragment JSON object, as described below.
Each fragment provides the following configuration options:		
fragmentPath	n/a	Currently must be "/"
parameters	null	An optional JSON object containing zero or more name-value pairs that are bound and passed in as XSLT parameters to the query. Example: myVar : "hello world"
fileParameters	null	An optional JSON object containing zero or more name-path pairs that refer to XML files on the local file system, and are bound and passed in as external variables to the query. These files are loaded once on program startup and subsequently remain memory resident across queries. This can be used for efficient joins where the join table is static and fits into main memory. Example: myDoc : src/test/resources/testdocuments/helloworld.xml
queryFile	null	A relative or absolute path of a local file from which to load the query.
queryString	null	An inline string from which to load the query. One of queryFile or queryString must be present, but not both.

Note: The morphline configuration file is implemented using the HOCON format (Human Optimized Config Object Notation). HOCON is basically JSON slightly adjusted for the configuration file use case. HOCON syntax is defined at HOCON github page and as such, multi-line strings are similar to Python or Scala, using triple quotes. If the three-character sequence """ appears, then all Unicode characters until a closing """ sequence are used unmodified to create a string value.

Example usage:

```
xslt {
  fragments : [
    {
      fragmentPath : "/"
      parameters : {
```

```
        myVariable : "hello world"
      }
      queryString : """
        <!-- Example XSLT identity transformation -->
        <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver
sion="1.0">

        <xsl:template match="@*|node()">
          <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
          </xsl:copy>
        </xsl:template>

        </xsl:stylesheet>
      """
    }
  ]
}
```

More examples can be found in the unit tests.

For more background, see resources such as the XSLT Tutorial and Wikipedia.

kite-morphlines-solr-core

This maven module contains morphline commands for Solr that higher level modules such as kite-morphlines-solr-cell, search-mr, and search-flume depend on for indexing.

solrLocator

A solrLocator is a set of configuration parameters that identify the location and schema of a Solr server or SolrCloud. Based on this information a morphline Solr command can fetch the Solr index schema and send data to Solr. A solr Locator is not actually a command but rather a common parameter of many morphline Solr commands, and thus described separately here.

Example usage:

```
solrLocator : {
  # Name of solr collection
  collection : collection1

  # ZooKeeper ensemble
  zkHost : "127.0.0.1:2181/solr"
  # Max number of documents to pass per RPC from morphline to Solr Server
  # batchSize : 10000
}
```

loadSolr

The loadSolr command (source code) inserts, updates or deletes records into a Solr server or MapReduce Reducer.

The command provides the following configuration options:

Property Name	Default	Description
solrLocator	n/a	Solr location parameters as described separately above.

Property Name	Default	Description
boosts	[]	An optional JSON object containing zero or more fieldName-boostValue mappings where the fieldName is a String and the boostValue is a float. The default boost is 1.0.

Examples:

- loadSolrUpdate
- loadSolrPartialUpdate
- loadSolrDeleteById
- loadSolrDeleteByQuery
- loadSolrChildDocuments

Example loadSolr usage to insert a document or update an existing document stored in Solr ("update")

```
loadSolr {
  solrLocator : {
    # Name of solr collection
    collection : collection1
    # ZooKeeper ensemble
    zkHost : "127.0.0.1:2181/solr"

    # Max number of docs to pass per RPC from morphline to Solr Server
    # batchSize : 10000
  }
  boosts : {
    id : 2.0 # assign to the id field a boost value 2.0
  }
}
```

Example loadSolr usage to update a subset of fields of an existing document stored in Solr ("partial document update"):

```
java { code : """
  // specify the unique key of the document stored in Solr that shall be
updated
  record.put("id", 123);

  // set "first_name" field of stored Solr document to "Nadja"; retain ot
her fields as-is
  record.put("first_name", Collections.singletonMap("set", "Nadja"));

  // set "tags" field of stored Solr document to multiple values ["smart", "
creative"]; retain other fields as-is
  record.put("tags", Collections.singletonMap("set", Arrays.asList("smart",
 "creative")));

  // add "San Francisco" to the existing values of the cities field of the
 stored Solr document; retain other fields as-is
  record.put("cities", Collections.singletonMap("add", "San Francisco"));

  // remove the "text" field from a document stored in Solr; retain other
fields as-is
  record.put("text", Collections.singletonMap("set", null));

  // increment user_friends_count by 5; retain other fields of stored Solr
document as-is
  record.put("user_friends_count", Collections.singletonMap("inc", 5));

  // pass record to next command in chain
```

```
  return child.process(record);
              """
}

loadSolr {
  <solrLocator goes here>
}
```

Note:

A partial document update requires that all Solr fields be configured as stored="true" in schema.xml, not just the field that shall be updated. This is because this kind of "update" is implemented as a document delete followed by a document insert: the old values of all fields are fetched from Solr (requires stored="true" markers), then the old document is deleted and then a new document is inserted with the old values plus the new field value. For details see this article and http://stackoverflow.com/questions/12183798/solrj-api-for-partial-document-update

Example loadSolr usage for deleteById:

```
# Tell loadSolr command to delete the documents for which the unique key fie
ld equals 123 or 456.
setValues {
  _loadSolr_deleteById:[123, 456]
}

loadSolr {
  <solrLocator goes here>
}
```

Example loadSolr usage for deleteByQuery:

```
# Tell loadSolr command to delete all documents for which the following cond
itions hold:
# The city field starts with "Paris" AND the color field equals "blue" OR
# The city field starts with "London" AND the color field equals "purple"
setValues {
  _loadSolr_deleteByQuery:["(city:Paris*)AND(color:blue)", "(city:London*)
AND(color:purple)"]
}

loadSolr {
  <solrLocator goes here>
}
```

Example loadSolr usage for child documents (aka nested documents):

A record can contain (arbitrarily nested) child documents (aka nested documents aka nested records) in the "_loadSolr_childDocuments" morphline record field. If present, these are recognized and indexed by the loadSolr command, and the parent-child relationships become available to Solr queries, as shown below:

```
java {
  code: """
    // Index a document that has a foo child document, which in turn has a b
ar child document
    record.put("id", "12345");
    record.put("content_type", "parent");
    Record childDoc = new Record();
    childDoc.put("id", "foo");
    childDoc.put("content_type", "child");
    Record childDoc2 = new Record();
    childDoc2.put("id", "bar");
    childDoc2.put("content_type", "child");
```

```
    childDoc.put("_loadSolr_childDocuments", childDoc2); // mark as child
doc
    record.put("_loadSolr_childDocuments", childDoc); // mark as child doc
    return child.process(record);
        """
  }
}
loadSolr {
  <solrLocator goes here>
}
```

Example Solr parent block join that returns the parent records for records where the child documents contain "bar" in the id field:

```
{!parent which='content_type:parent'}id:bar
```

For more background see this article.

generateSolrSequenceKey

The generateSolrSequenceKey command (source code) assigns a record unique key that is the concatenation of the given baseIdField record field, followed by a running count of the record number within the current session. The count is reset to zero whenever a startSession notification is received.

For example, assume a CSV file containing multiple records but no unique ids, and the base_id field is the filesystem path of the file. Now this command can be used to assign the following record values to Solr's unique key field: $path#0, $path#1, ... $path#N.

The name of the unique key field is fetched from Solr's schema.xml file, as directed by the solrLocator configuration parameter.

The command provides the following configuration options:

Property Name	Default	Description
solrLocator	n/a	Solr location parameters as described separately above.
baseIdField	baseid	The name of the input field to use for prefixing keys.
preserveExisting	true	Whether to preserve the field value if one is already present.solrLocator n/a Solr location parameters as described separately above. baseIdField baseid The name of the input field to use for prefixing keys. preserveExisting true Whether to preserve the field value if one is already present.

Example usage:

```
generateSolrSequenceKey {
  baseIdField: ignored_base_id
  solrLocator : ${SOLR_LOCATOR}
}
```

sanitizeUnknownSolrFields

The sanitizeUnknownSolrFields command (source code) sanitizes record fields that are unknown to Solr schema.xml by either deleting them (renameToPrefix parameter is absent or a zero length string) or by moving them to a field prefixed with the given renameToPrefix (for example, to use typical dynamic Solr fields).

Recall that Solr throws an exception on any attempt to load a document that contains a field that is not specified in schema.xml.

The command provides the following configuration options:

Property Name	Default	Description
solrLocator	n/a	Solr location parameters as described separately above.
renameToPrefix	""	Output field prefix for unknown fields.

Example usage:

```
sanitizeUnknownSolrFields {
  solrLocator : ${SOLR_LOCATOR}
}
```

tokenizeText

The tokenizeText command (source code) uses the embedded Solr/Lucene Analyzer library to generate tokens from a text string, without sending data to a Solr server.

This is useful for prototyping and debugging Solr applications. It is also useful for standalone usage outside of Solr, e.g. for extracting text features from documents for use with recommendation systems, clustering and classification applications.

The command provides the following configuration options:

Property Name	Default	Description
solrLocator	n/a	Solr location parameters as described separately above.
inputField	n/a	The name of the input field.
outputField	n/a	The name of the field to add output values to.
solrFieldType	n/a	The name of the Solr field type in schema.xml to use for text analysis and tokenization. This parameter specifies the algorithmic extraction rules. Example: "text_en"

Example usage:

```
tokenizeText {
  inputField : message
  outputField : tokens
  solrFieldType : text_en
  solrLocator : {
    # Name of solr collection
    collection : collection1
    # ZooKeeper ensemble
    zkHost : "127.0.0.1:2181/solr"

    # solrHomeDir : "example/solr/collection1"
  }
}
```

For example, given the input field message with the value Hello World!\nFoo@Bar.com #%()123 the expected output record is:

```
tokens:hello
tokens:world
tokens:foo
```

```
tokens:bar.com
tokens:123
```

This example assumes the Solr field type named "text_en" is defined in schema.xml as shown in the following snippet:

```
...
<fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory"
       ignoreCase="true"
       words="lang/stopwords_en.txt"
       enablePositionIncrements="true"
    />
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EnglishPossessiveFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.tx
t"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
</fieldType>
```

kite-morphlines-solr-cell

This maven module contains morphline commands for using SolrCell with Tika parsers. This includes support for types including HTML, XML, PDF, Word, Excel, Images, Audio, and Video.

solrCell

The solrCell command (source code) pipes the first attachment of a record into one of the given Apache Tika parsers, then maps the Tika output back to a record using Apache SolrCell.

The Tika parser is chosen from the configurable list of parsers, depending on the MIME type specified in the input record. Typically, this requires an upstream detectMimeType command.

The command provides the following configuration options:

Property Name	Default	Description
solrLocator	n/a	Solr location parameters as described separately above.
capture	[]	List of XHTML element names to extract from the Tika output. For instance, it could be used to grab paragraphs (<p>) and index them into a separate field. Note that content is also still captured into the overall "content" field.
fmaps	[]	Maps (moves) one field name to another. See the example below.

Property Name	Default	Description
uprefix	null	The uprefix option indicates that the command shall prefix all fields that are not defined in the Solr schema.xml with the given prefix. Recall that Solr throws an exception on any attempt to load a document that contains a field that is not specified in schema.xml. The uprefix option is very useful when combined with dynamic field definitions. For example, uprefix : ignored_ would effectively ignore all unknown fields generated by Tika if the schema.xml contains the following dynamic field definition: dynamicField name="ignored_*" type="ignored"
captureAttr	false	Whether to index attributes of the Tika XHTML elements into separate fields, named after the element. For example, when extracting from HTML, Tika can return the href attributes in <a> tags as fields named "a".
xpath	null	When extracting, only return Tika XHTML content that satisfies the XPath expression. See http://tika.apache.org/1.4/parser.html for details on the format of Tika XHTML. See also http://wiki.apache.org/solr/TikaExtractOnlyExampleOutput.
lowernames	false	Map all field names to lowercase with underscores. For example, Content-Type would be mapped to content_type.
solrContentHandlerFactory	org.kitesdk.morphline.solrcell.TrimSolrContentHandlerFactory	A Java class to handle bridging from Tika to SolrCell.
parsers	[]	List of fully qualified Java class names of one or more Tika parsers.

Example usage:

```
solrCell {
  solrLocator : ${SOLR_LOCATOR}

  # extract some fields
  capture : [content, a, h1, h2]

  # rename exif_image_height field to text field
  # rename a field to anchor field
  # rename h1 field to heading1 field
  fmap : { exif_image_height : text, a : anchor, h1 : heading1 }

  # xpath : "/xhtml:html/xhtml:body/xhtml:div/descendant:node()"

  parsers : [ # one or more nested Tika parsers
    { parser : org.apache.tika.parser.jpeg.JpegParser }
  ]
}
```

Here is a complex morphline that demonstrates integrating multiple heterogenous input file formats via a tryR ules command, including Avro and SolrCell, using auto detection of MIME types via detectMimeType command, recursion via the callParentPipe command for unwrapping container formats, and automatic UUID generation:

```
morphlines : [
  {
    id : morphline1
    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
```

```
commands : [
  {
    # emit one output record for each attachment in the input
    # record's list of attachments. The result is a list of
    # records, each of which has at most one attachment.
    separateAttachments {}
  }

  {
    # used for auto-detection if MIME type isn't explicitly supplied
    detectMimeType {
      includeDefaultMimeTypes : true
      mimeTypesFiles : [target/test-classes/custom-mimetypes.xml]
    }
  }

  {
    tryRules {
      throwExceptionIfAllRulesFailed : true
      rules : [
        # next rule of tryRules cmd:
        {
          commands : [
            { logDebug { format : "hello unpack" } }
            { unpack {} }
            { generateUUID {} }
            { callParentPipe {} }
          ]
        }

        # next rule of tryRules cmd:
        {
          commands : [
            { logDebug { format : "hello decompress" } }
            { decompress {} }
            { callParentPipe {} }
          ]
        }
        # next rule of tryRules cmd:
        {
          commands : [
            {
              readAvroContainer {
                supportedMimeTypes : [avro/binary]
                # optional, avro json schema blurb for getSchema()
                # readerSchemaString : "<json can go here>"
                # readerSchemaFile : /path/to/syslog.avsc
              }
            }

            { extractAvroTree {} }

            {
              setValues {
                id : "@{/id}"
                user_screen_name : "@{/user_screen_name}"
                text : "@{/text}"
              }
            }

            {
              sanitizeUnknownSolrFields {
                solrLocator : ${SOLR_LOCATOR}
```

```
                }
              }
            ]
          }

          # next rule of tryRules cmd:
          {
            commands : [
              {
                readJsonTestTweets {
                  supportedMimeTypes : ["mytwittertest/json+delimited+len
gth"]
                }
              }
              {
                sanitizeUnknownSolrFields {
                  solrLocator : ${SOLR_LOCATOR}
                }
              }
            ]
          }

          # next rule of tryRules cmd:
          {
            commands : [
              { logDebug { format : "hello solrcell" } }
              {
                # wrap SolrCell around an Tika parsers
                solrCell {
                  solrLocator : ${SOLR_LOCATOR}

                  capture : [
                    # twitter feed schema
                    user_friends_count
                    user_location
                    user_description
                    user_statuses_count
                    user_followers_count
                    user_name
                    user_screen_name
                    created_at
                    text
                    retweet_count
                    retweeted
                    in_reply_to_user_id
                    source
                    in_reply_to_status_id
                    media_url_https
                    expanded_url
                   ]

                  # rename "content" field to "text" fields
                  fmap : { content : text, content-type : content_type }

                  lowernames : true
                  # Tika parsers to be registered:
                  parsers : [
                    # { parser : org.apache.tika.parser.AutoDetectParser }
                    { parser : org.apache.tika.parser.asm.ClassParser }
                    { parser : org.gagravarr.tika.FlacParser }
                    { parser : org.apache.tika.parser.audio.AudioParser }
                    { parser : org.apache.tika.parser.audio.MidiParser }
                    { parser : org.apache.tika.parser.crypto.Pkcs7Parser }
                    { parser : org.apache.tika.parser.dwg.DWGParser }
```

```
                        { parser : org.apache.tika.parser.epub.EpubParser }
                        { parser : org.apache.tika.parser.executable.Executabl
eParser }
                        { parser : org.apache.tika.parser.feed.FeedParser }
                        { parser : org.apache.tika.parser.font.AdobeFontMetric
Parser }
                        { parser : org.apache.tika.parser.font.TrueTypeParser
 }
                        { parser : org.apache.tika.parser.xml.XMLParser }
                        { parser : org.apache.tika.parser.html.HtmlParser }
                        { parser : org.apache.tika.parser.image.ImageParser }
                        { parser : org.apache.tika.parser.image.PSDParser }
                        { parser : org.apache.tika.parser.image.TiffParser }
                        { parser : org.apache.tika.parser.iptc.IptcAnpaParse
r }
                        { parser : org.apache.tika.parser.iwork.IWorkPackagePa
rser }
                        { parser : org.apache.tika.parser.jpeg.JpegParser }
                        { parser : org.apache.tika.parser.mail.RFC822Parser }
                        { parser : org.apache.tika.parser.mbox.MboxParser,
                            additionalSupportedMimeTypes : [message/x-emlx] }
                        { parser : org.apache.tika.parser.microsoft.OfficePa
rser }
                        { parser : org.apache.tika.parser.microsoft.TNEFPar
ser }
                        { parser : org.apache.tika.parser.microsoft.ooxml.OO
XMLParser }
                        { parser : org.apache.tika.parser.mp3.Mp3Parser }
                        { parser : org.apache.tika.parser.mp4.MP4Parser }
                        { parser : org.apache.tika.parser.hdf.HDFParser }
                        { parser : org.apache.tika.parser.netcdf.NetCDFParser
}
                        { parser : org.apache.tika.parser.odf.OpenDocumentPa
rser }
                        { parser : org.apache.tika.parser.pdf.PDFParser }
                        { parser : org.apache.tika.parser.pkg.CompressorPar
ser }
                        { parser : org.apache.tika.parser.pkg.PackageParser }
                        { parser : org.apache.tika.parser.rtf.RTFParser }
                        { parser : org.apache.tika.parser.txt.TXTParser }
                        { parser : org.apache.tika.parser.video.FLVParser }
                        { parser : org.apache.tika.parser.xml.DcXMLParser }
                        { parser : org.apache.tika.parser.xml.FictionBookPar
ser }
                        { parser : org.apache.tika.parser.chm.ChmParser }
                    ]
                  }
                }

                { generateUUID { field : ignored_base_id } }
                {
                  generateSolrSequenceKey {
                    baseIdField: ignored_base_id
                    solrLocator : ${SOLR_LOCATOR}
                  }
                }
              ]
            }
          ]
        }
      }

      {
        loadSolr {
```

```
          solrLocator : ${SOLR_LOCATOR}
        }
      }

      {
        logDebug {
          format : "My output record: {}"
          args : ["@{}"]
        }
      }
    ]
  }
]
```

Note: More information on SolrCell can be found here: http://wiki.apache.org/solr/ExtractingRequestHandler

kite-morphlines-useragent

userAgent

The userAgent command (source code) parses a user agent string and returns structured higher level data like user agent family, operating system, version, and device type, using the underlying API and regexes.yaml BrowserScope database from ua-parser.

The command provides the following configuration options:

Property Name	Default	Description
inputField	n/a	The name of the input field that contains zero or more user agent strings.
outputFields	[]	A JSON object containing zero or more user agent mappings. Each mapping consists of a record output field name (on the left side of the colon ':') as well as an expression (on the right hand side). An expression consists of a concatenation of zero or more literal strings or components of the form @{componentName}. Example mapping: myOutputField : "@{ua_family}/@{ua_major}.@{ua_minor}.@{ua_patch}". The following components are available: ua_family, ua_major, ua_minor, ua_patch, os_family, os_major, os_minor, os_patch, os_patch_minor, device_family. If a component resolves to null or the empty string the preceding string separator, if any, is suppressed.
database	null	The (optional) relative or absolute path of a regexes.yaml database file on the local file system. The default is to use the standard regexes.yaml database file that ships embedded inside of the ua-parser-*.jar. Example: /path/to/regexes.yaml

Example usage:

```
userAgent {
  inputField : user_agents
  outputFields : {
    ua_family : "@{ua_family}"
    device_family : "@{device_family}"
```

```
    ua_family_and_version : "@{ua_family}/@{ua_major}.@{ua_minor}.@{ua_patch
}"
    os_family_and_version : "@{os_family} @{os_major}.@{os_minor}.@{os_pa
tch}"
  }
}
```

Example input:

```
user_agents : Mozilla/5.0 (iPhone; CPU iPhone OS 5_1_1 like Mac OS X) AppleW
ebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9B206 Safari/7534.48.3
```

Expected output:

```
ua_family : Mobile Safari
device_family : iPhone
ua_family_and_version : Mobile Safari/5.1
os_family_and_version : iOS 5.1.1
```