Denodo HDFS Custom Wrapper

Revision 20170309

CONTENTS

1 INTRODUCTION....4

2 HDFS....5
2.1 DELIMITED TEXT FILES....6
2.2 SEQUENCEFILES....6
2.3 MAPFILES....7
2.4 AVRO FILES....7
2.5 PARQUET FILES....8

3 HDFS CUSTOM WRAPPER....10
3.1 HDFSDELIMITEDTEXTFILEWRAPPER....10
3.2 HDFSSEQUENCEFILEWRAPPER....13
3.3 HDFSMAPFILEWRAPPER....14
3.4 HDFSAVROFILEWRAPPER....16
3.5 WEBHDFSFILEWRAPPER....20
3.6 HDFSPARQUETFILEWRAPPER....22

4 HDFS COMPRESSED FILES....25

5 SECURE CLUSTER WITH KERBEROS....26

6 SOFTWARE REQUIREMENTS....29

7 TROUBLESHOOTING....30

1 INTRODUCTION

The `hdfs-customwrapper` distribution contains five Virtual DataPort custom wrappers capable of reading several file formats stored in **Hadoop Distributed File System** (HDFS).

Supported formats are:

- Delimited text files (both directly from HDFS and also via HDFS over HTTP)ncomp
- Sequence files
- Map files
- Avro files
- Parquet files

2 HDFS

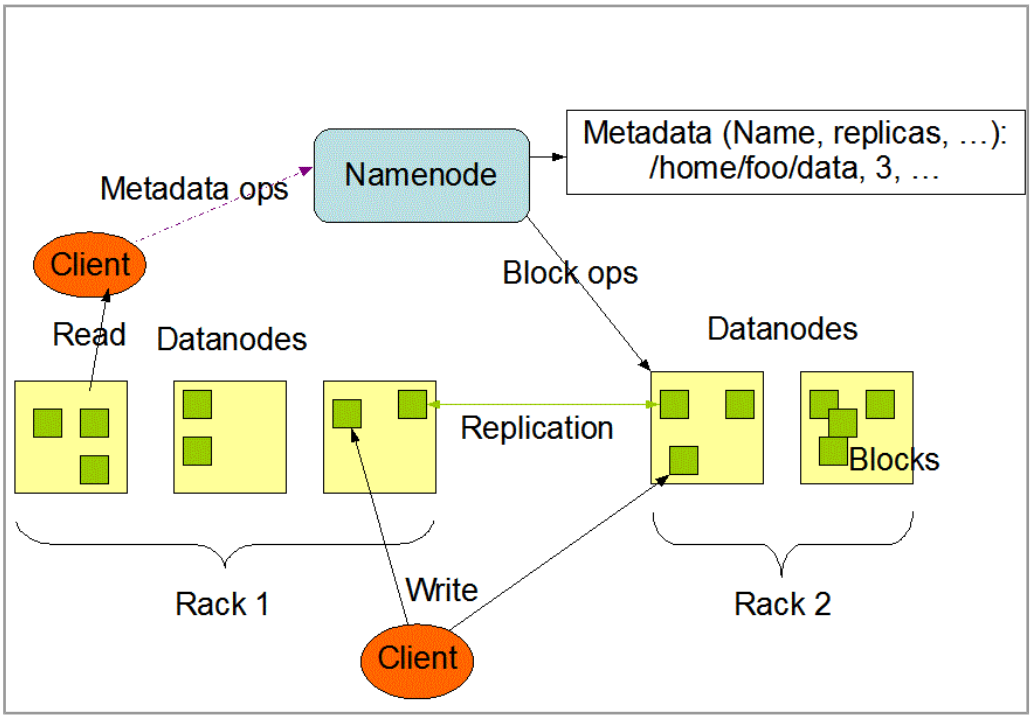
The Hadoop Distributed File System is a distributed, scalable, and portable file system used by the Hadoop platform.

In HDFS, data is divided into blocks and copies of these blocks are stored on other servers in the Hadoop cluster. That is, an individual file is actually stored as smaller blocks that are replicated across multiple servers in the entire cluster. This redundancy offers multiple benefits for Big Data processing:

- Higher availability.
- Better scalability: map and reduce functions can be executed on smaller subsets of large data sets.
- Data locality: move the computation closer to the data to reduce latency.

HDFS has a master/slave architecture in which the **NameNode**, the master, manages the file system namespace and regulates clients access to files. And the **DataNodes**, the slaves, manage storage attached to the nodes that they run on.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



HDFS Architecture

2.1 DELIMITED TEXT FILES

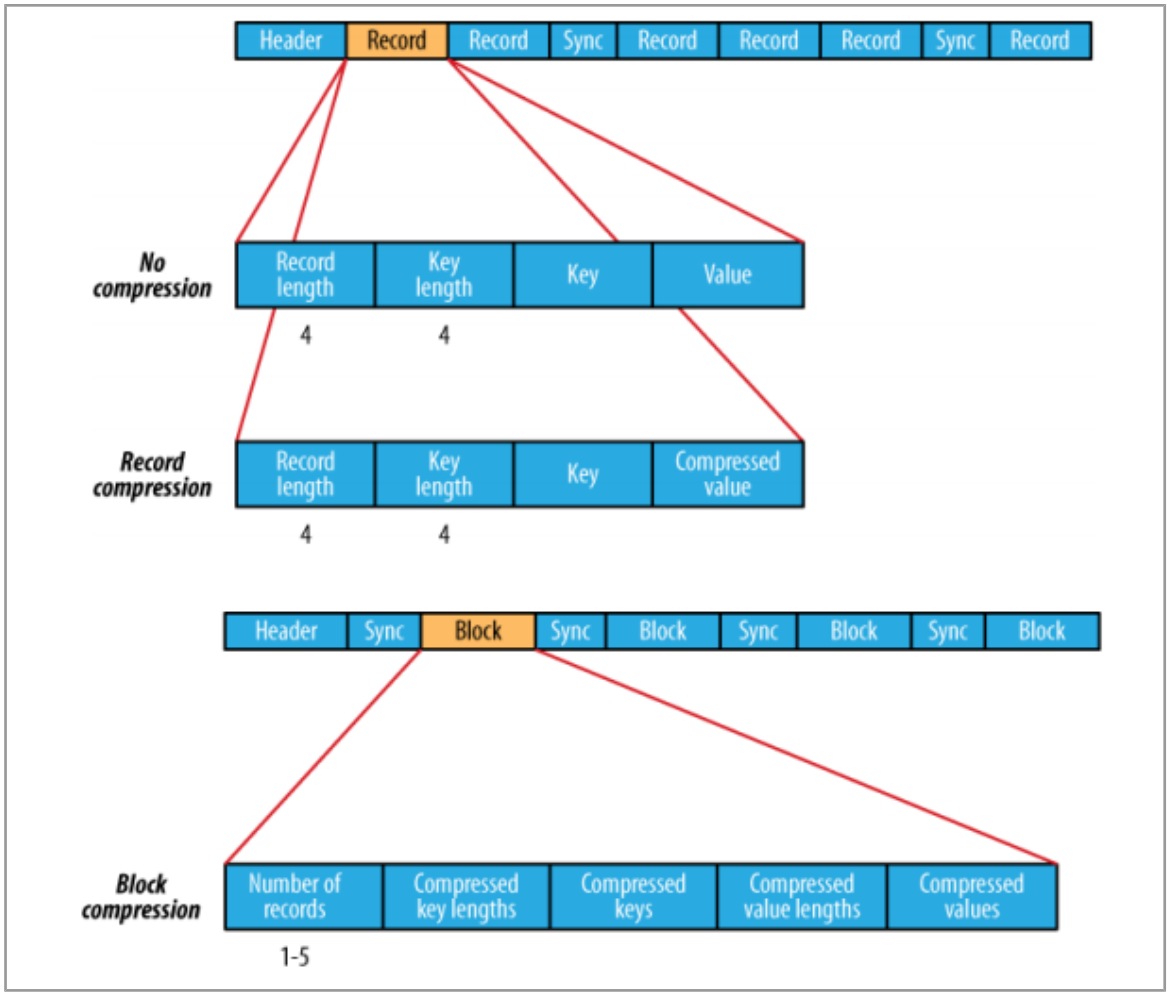
Delimited text files store plain text and each line has values separated by a delimiter, such as tab, space, comma, etc.

2.2 SEQUENCEFILES

Sequence files are binary record-oriented files, where each record has a serialized key and a serialized value.

The Hadoop framework supports compressing and decompressing sequence files transparently. Therefore, sequence files have three available formats:

- No compression.
- Record compression: only values are compressed.
- Block compression: both keys and values are collected in 'blocks' separately and compressed.



Sequence file formats

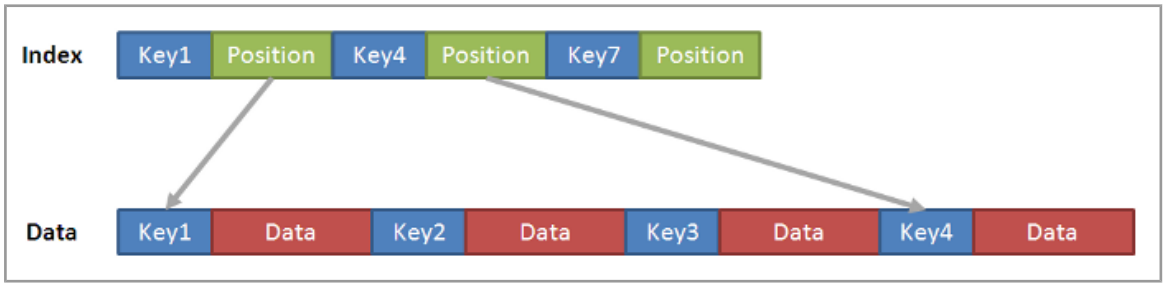
The three formats share a header that contains information which allows the reader to recognize their format:

- Version: 3 bytes of magic header SEQ, followed by 1 byte of actual version number.
- Key class name.
- Value class name.
- Compression: specifies if compression is turned on for keys/values.
- Block compression: specifies if block-compression is turned on for keys/values.
- Compression codec: `CompressionCodec` class which is used for compression of keys and/or values (if compression is enabled).
- Metadata.
- Sync: a sync marker to denote end of the header.

2.3 MAPFILES

A map is a directory containing two sequence files. The data file (`/data`) is identical to the sequence file and contains the data stored as binary key/value pairs. The index file (`/index`), which contains a key/value map with seek positions inside the data file to quickly access the data.

The index file is populated with the key and a `LongWritable` that contains the starting byte position of the record. Index does not contains all the keys but just a fraction of the keys. The index is read entirely into memory.



Map file format

2.4 AVRO FILES

Avro is a data serialization format.

Avro data files are self-describing, containing the full schema for the data in the file. An Avro schema is defined using JSON. The schema allows you to define two types of data:

- primitive data types: `string`, `integer`, `long`, `float`, `double`, `byte`, `null` and `boolean`.
- complex type definitions: a `record`, an `array`, an `enum`, a `map`, a `union` or a `fixed` type.

```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields":
    [
      {"name": "name", "type": "string"},
      {"name": "favorite_number", "type": ["int", "null"]},
      {"name": "favorite_color", "type": ["string", "null"]}
    ]
}
```

Avro schema

Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This allows each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.

2.5 PARQUET FILES

Parquet is a free and open source column-oriented data store of the Hadoop ecosystem. It provides data compression on a per-column level and encoding schemas.

The data are described by a schema. This schema starts by the word `Message` and contains a group of fields. Each field is defined by a *repetition* (required, optional, or repeated), a *type* and a *name*. An example of a file customer:

```
Message Customer {
      required int32 id;
      required binary firstname (UTF8);
      required binary lastname (UTF8);
}
```

Parquet schema

The primitives types of parquet are `boolean`, `int32`, `int64`, `int96`, `float`, `double`, `binary` and `fixed_len_byte_array`. There are not String types but there logical types which allows interpret a binary as a String, JSON or other types.

The complex types are defined by a group type, which adds a layer of nesting. LIST and MAP could be represented in a parquet file.

3 HDFS CUSTOM WRAPPER

`hdfs-customwrapper` library includes five custom wrappers:

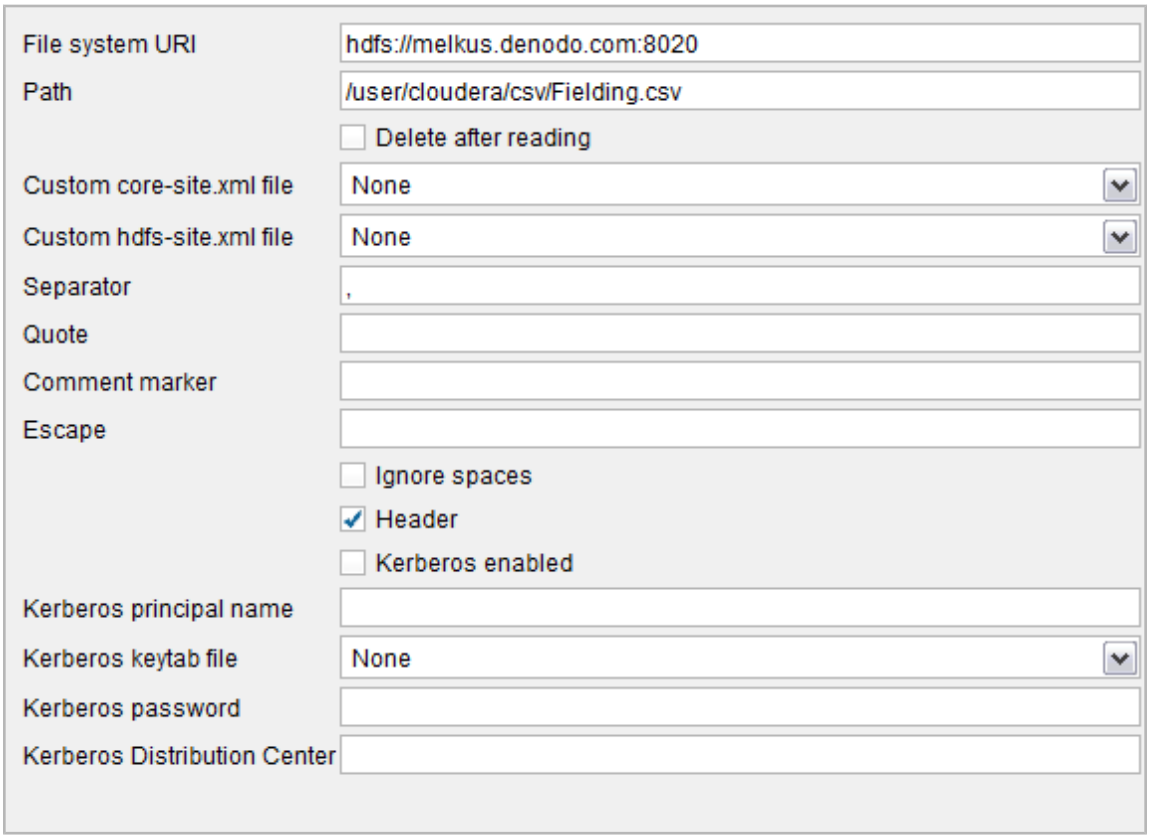
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSDelimitedTextFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSSequenceFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSMapFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSAvroFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.WebHDFSFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSParquetFileWrapper`

3.1 HDFSDELIMITEDTEXTFILEWRAPPER

Custom wrapper for reading delimited text files stored in HDFS.

The base views created from the `HDFSDelimitedTextFileWrapper` need the following parameters:

- `File system URI`: A URI whose scheme and authority identify the file system.The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
  - For HDFS the URI has the form `hdfs://<ip>:<port>`.
  - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash `/` should be replaced with the string `%2F`).
- `Path`: input path for the delimited file or the directory containing the files.
- `Delete after reading`: Requests that the file or directory denoted by the path be deleted when the wrapper terminates.
- `Custom core-site.xml file`: configuration file that overrides the default core parameters (common to HDFS and MapReduce). Optional.
- `Custom hdfs-site xml file`: configuration file that overrides the default hdfs parameters. Optional.
- `Separator`: delimiter between values. Default is the `comma`. Optional.
- `Quote`: Character used to encapsulate values containing special characters. Default is
  `Quote`. Optional.
- `Comment marker`: Character marking the start of a line comment. Comments are disable by default. Optional.
- `Escape`: Escape character. Escapes are disable by default. Optional.
- `Ignore spaces`: Whether spaces around values are ignored. `False` by default.
- `Header`: Whether the file has a header or not. `True` by default.



HDFSDelimitedTextFileWrapper base view edition

View schema:

Field Name	Field Type
playerid	text
yearid	text
stint	text
teamid	text
lgid	text
pos	text
g	text
gs_0	text
innouts	text
po	text
a	text
e	text
dp	text
pb	text
wp	text
sb	text
cs	text
zr	text

View schema

The execution of the wrapper returns the values contained in the file or group of files, if the Path input parameter denotes a directory.

Total rows received: 167938 (shown 150)

playerid	yearid	stint	teamid	lgid	pos	g	gs
abercda01	1871	1	TRO	NA	SS	1	
addybo01	1871	1	RC1	NA	2B	22	
addybo01	1871	1	RC1	NA	SS	3	
allisar01	1871	1	CL1	NA	2B	2	
allisar01	1871	1	CL1	NA	OF	29	
allisdo01	1871	1	WS3	NA	C	27	
ansonca01	1871	1	RC1	NA	1B	1	
ansonca01	1871	1	RC1	NA	2B	2	
ansonca01	1871	1	RC1	NA	3B	20	
ansonca01	1871	1	RC1	NA	C	5	
ansonca01	1871	1	RC1	NA	OF	1	

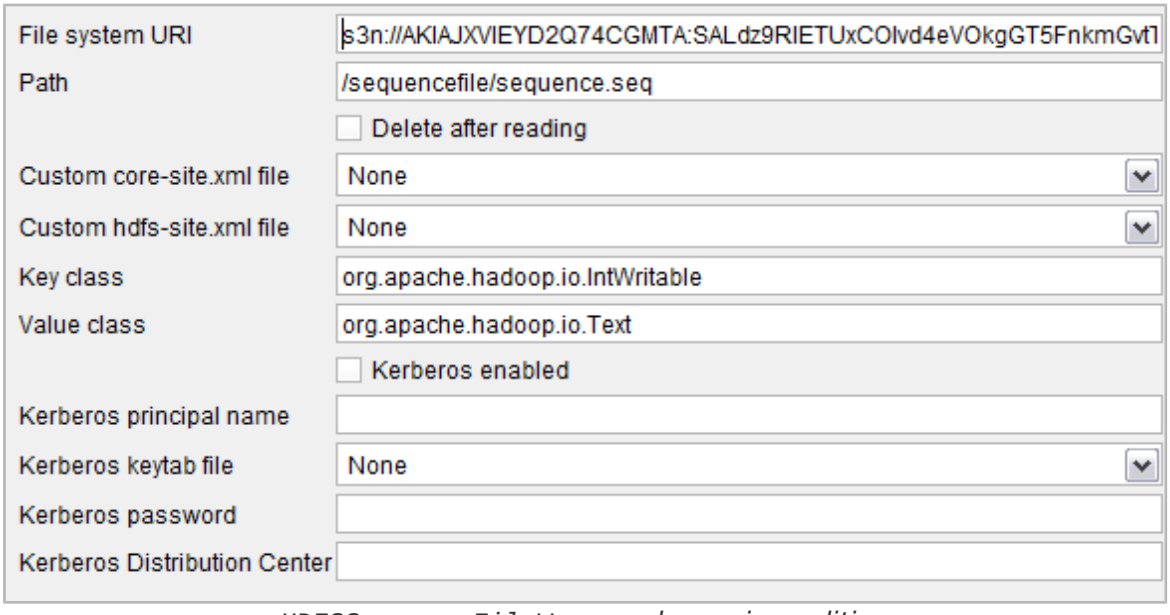
View results

3.2 HDFSSEQUENCEFILEWRAPPER

Custom wrapper for reading sequence files stored in HDFS.

The base views created from the `HDFSSequenceFileWrapper` need the following parameters:

- `File system URI`: A URI whose scheme and authority identify the file system.The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
  - For HDFS the URI has the form `hdfs://<ip>:<port>`.
  - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash `/` should be replaced with the string `%2F`).
- `Path`: input path for the sequence file or the directory containing the files.
- `Delete after reading`: Requests that the file or directory denoted by the path be deleted when the wrapper terminates.
- `Custom core-site.xml file`: configuration file that overrides the default core parameters (common to HDFS and MapReduce). Optional.
- `Custom hdfs-site xml file`: configuration file that overrides the default hdfs parameters. Optional.
- `Key class`: key class name implementing `org.apache.hadoop.io.Writable` interface.
- `Value class`: value class name implementing `org.apache.hadoop.io.Writable` interface.



`HDFSSequenceFileWrapper` base view edition

View schema:

Field Name	Field Type
key	int
value	text

View schema

The execution of the wrapper returns the key/value pairs contained in the file or group of files, if the Path input parameter denotes a directory.

Total rows received: 100 (shown 100)

key	value
100	One, two, buckle my shoe
99	Three, four, shut the door
98	Five, six, pick up sticks
97	Seven, eight, lay them straight
96	Nine, ten, a big fat hen
95	One, two, buckle my shoe
94	Three, four, shut the door
93	Five, six, pick up sticks
92	Seven, eight, lay them straight
91	Nine, ten, a big fat hen
90	One, two, buckle my shoe
89	Three, four, shut the door

View results

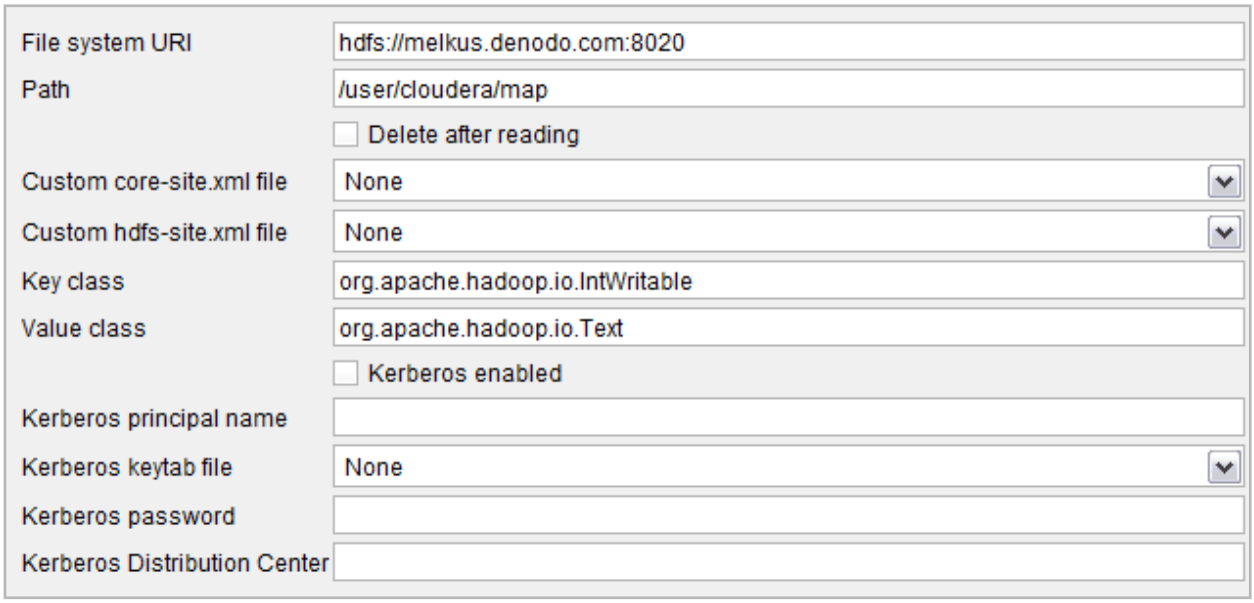
3.3 HDFSMAPFILEWRAPPER

Custom wrapper for reading map files stored in HDFS.

The base views created from the `HDFSMapFileWrapper` need the following parameters:

- `File system URI`: A URI whose scheme and authority identify the file system.The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
  - For HDFS the URI has the form `hdfs://<ip>:<port>`.
  - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash `/` should be replaced with the string `%2F`).
- `Path`: input path for the directory containing the map file. Also the path to the index or data file could be specified. When using **Amazon S3**, a flat file system where there is no folder concept, the path to the index or data should be used.
- `Delete after reading`: Requests that the file or directory denoted by the path be deleted when the wrapper terminates.
- `Custom core-site.xml file`: configuration file that overrides the default core parameters (common to HDFS and MapReduce). Optional.

- `Custom hdfs-site xml file`: configuration file that overrides the default hdfs parameters. Optional.
- `Key class`: key class name implementing the `org.apache.hadoop.io.WritableComparable` interface. `WritableComparable` is used because records are sorted in **key order**.
- `Value class`: value class name implementing the `org.apache.hadoop.io.Writable` interface.



`HDFSMapFileWrapper` base view edition

View schema

The execution of the wrapper returns the key/value pairs contained in the file or group of files, if the `Path` input parameter denotes a directory.

Total rows received: 1024 (shown 150)

key	value
1	One, two, buckle my shoe
2	Three, four, shut the door
3	Five, six, pick up sticks
4	Seven, eight, lay them straight
5	Nine, ten, a big fat hen
6	One, two, buckle my shoe
7	Three, four, shut the door
8	Five, six, pick up sticks
9	Seven, eight, lay them straight
10	Nine, ten, a big fat hen
11	One, two, buckle my shoe
12	Three, four, shut the door

View results

3.4 HDFSAVROFILEWRAPPER

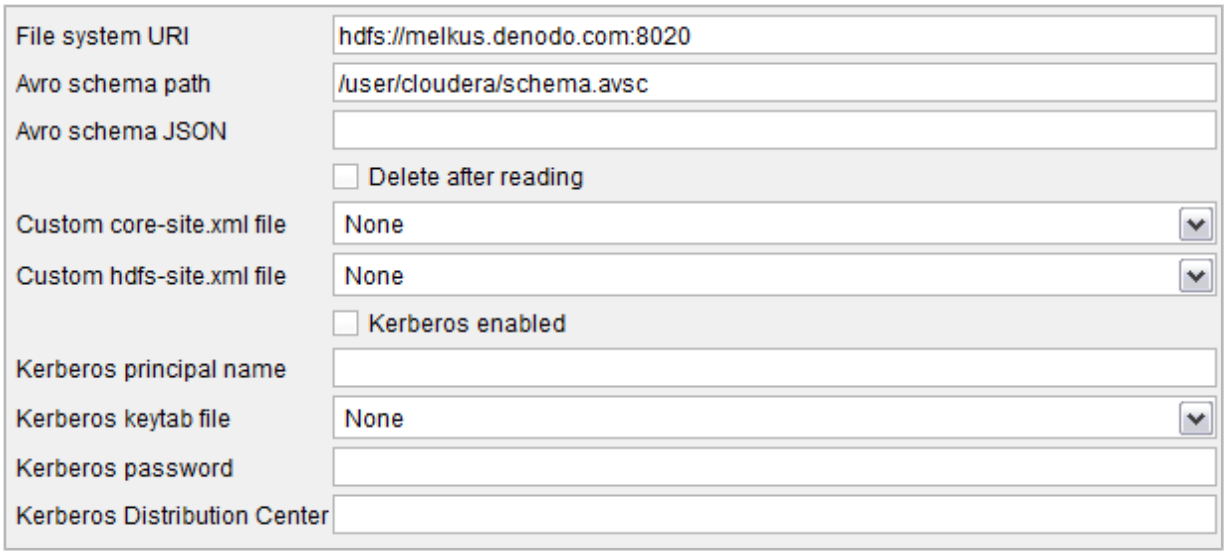
Custom wrapper for reading Avro files stored in HDFS.

The base views created from the `HDFSAvroFileWrapper` need the following parameters:

- `File system URI`: A URI whose scheme and authority identify the file system.The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
  - For HDFS the URI has the form `hdfs://<ip>:<port>`.
  - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash `/` should be replaced with the string `%2F`).
- `Delete after reading`: Requests that the file denoted by the path be deleted when the wrapper terminates.
- `Custom core-site.xml file`: configuration file that overrides the default core parameters (common to HDFS and MapReduce). Optional.
- `Custom hdfs-site xml file`: configuration file that overrides the default hdfs parameters. Optional.

There is also two parameters that are **mutually exclusive**:

- `Avro schema path`: input path for the Avro schema file.
- `Avro schema JSON`: JSON of the Avro schema.



HDFSAvroFileWrapper base view edition

```
{"type" : "record",
  "name" : "Doc",
  "doc" : "adoc",
  "fields" : [ {
    "name" : "id",
    "type" : "string"
  }, {
    "name" : "user_friends_count",
    "type" : [ "int", "null" ]
  }, {
    "name" : "user_location",
    "type" : [ "string", "null" ]
  }, {
    "name" : "user_description",
    "type" : [ "string", "null" ]
  }, {
    "name" : "user_statuses_count",
    "type" : [ "int", "null" ]
  }, {
    "name" : "user_followers_count",
    "type" : [ "int", "null" ]
  }, {
    "name" : "user_name",
    "type" : [ "string", "null" ]
  }, {
    "name" : "user_screen_name",
    "type" : [ "string", "null" ]
  }, {
    "name" : "created_at",
    "type" : [ "string", "null" ]
  }, {
    "name" : "text",
    "type" : [ "string", "null" ]
  }, {
    "name" : "retweet_count",
    "type" : [ "int", "null" ]
  }, {
    "name" : "retweeted",
    "type" : [ "boolean", "null" ]
  }, {
    "name" : "in_reply_to_user_id",
    "type" : [ "long", "null" ]
  }, {
    "name" : "source",
    "type" : [ "string", "null" ]
  }, {
    "name" : "in_reply_to_status_id",
    "type" : [ "long", "null" ]
  }, {
    "name" : "media_url_https",
    "type" : [ "string", "null" ]
  }, {
    "name" : "expanded_url",
    "type" : [ "string", "null" ]
  } ] }
```

Avro schema

View schema:

Field Name	Field Type
avrofilepath	text
doc	avro_ds_doc
id	text
user_friends_count	int
user_location	text
user_description	text
user_statuses_count	int
user_followers_count	int
user_name	text
user_screen_name	text
created_at	text
text	text
retweet_count	int
retweeted	boolean
in_reply_to_user_id	long
source	text
in_reply_to_status_id	long
media_url_https	text
expanded_url	text

View schema

The execution of the view returns the values contained in the Avro file specified in the WHERE clause of the VQL sentence:

```
SELECT * FROM avro_ds_file WHERE avrofilepath =
   '/user/cloudera/file.avro'
```

Total rows received: 2104 (shown 150)

avrofilepath	doc
/user/cloudera/file.avro	[Register]...
/user/cloudera/file.avro	[Register]...
/user/cloudera/file.avro	[Register]

RESU... -> doc

id	user_frien...	user_loca...	user_des...	user_stat...	user_follo...	user_name	user_scre...	created_at	text	re
10000	10000	location1...		1	1	fake user...	fake_user...	1985-05-...	tweet text ...	0

View results

After applying a flattening operation results are as follows.

Total rows received: 2104 (shown 150)

avrofilepath	id	user_frien...	user_locati...	user_desc...	user_statu...	user_follo...	user_name	user_scre...	cre
/user/cloud...	10000	10000	location10...		1	1	fake user1...	fake_user1...	198
/user/cloud...	10001	10001	location10...		1	1	fake user1...	fake_user1...	198

Flattened results

3.5 WEBHDFSFILEWRAPPER

Custom wrapper for reading delimited text files stored in HDFS using the **WebHDFS**.

3.5.1 About WebHDFS

WebHDFS provides HTTP REST access to HDFS. It supports all HDFS user operations including reading files, writing to files, making directories, changing permissions and renaming.

The advantage of WebHDFS are:

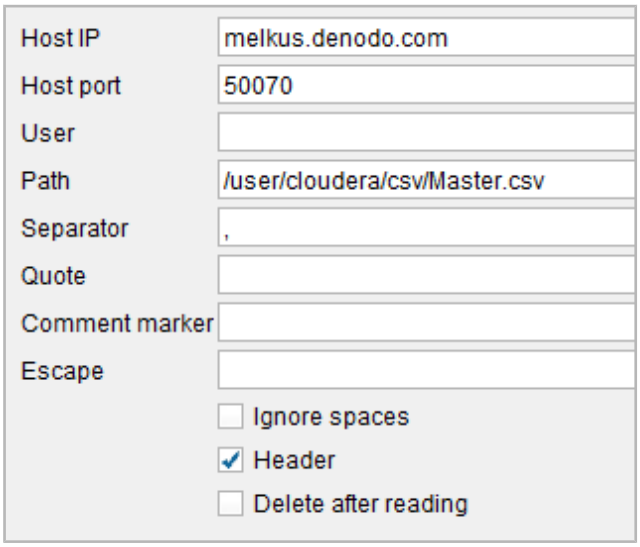
- **Version-independent** REST-based protocol which means that can be read and written to/from Hadoop clusters no matter their version. This addresses the issue of using the Java API (RPC-based) that requires both the client and the Hadoop cluster to share the same version. Upgrading one without the other causes serialization errors meaning the client cannot interact with the cluster.
- Read and write data in HDFS in a cluster behind a firewall. A proxy WebHDFS (for example: HttpFS) could be use, it acts as a gateway and is the only system that is allowed to send and receive data through the firewall.
  The only difference between using or not the proxy will be in the `host:port` pair where the HTTP requests are issued:
  - Default port for WebHDFS is 50075.
  - Default port for HttpFS is 14000.

3.5.2 Custom wrapper

The base views created from the `WebHDFSFileWrapper` need the following parameters:

- `Host IP`: IP or `<bucket>.s3.amazonaws.com` for Amazon S3.
- `Host port`: HTTP port. Default port for WebHDFS is 50075. For HttpFS is 14000. For Amazon S3 is 80.

- `User`: The name of the the authenticated user when security is off. If is not set, the server may either set the authenticated user to a default web user, if there is any, or return an error response.
  When using **Amazon S3** `<id>:<secret>` should be indicated.
- `Path`: input path for the delimited file.
- `Separator`: delimiter between values. Default is the `comma`.
- `Quote`: Character used to encapsulate values containing special characters. Default is
  `quote`.
- `Comment marker`: Character marking the start of a line comment. Comments are disable by default.
- `Escape`: Escape character. Escapes are disable by default.
- `Ignore spaces`: Whether spaces around values are ignored. `False` by default.
- `Header`: Whether the file has a header or not. `True` by default.
- `Delete after reading`: Requests that the file or directory denoted by the path be deleted when the wrapper terminates.



`WebHDFSFileWrapper` base view edition

View schema:

Field Name	Field Type
playerid	text
birthyear	text
birthmonth	text
birthday	text
birthcountry	text
birthstate	text
birthcity	text
deathyear	text
deathmonth	text
deathday	text
deathcountry	text
deathstate	text
deathcity	text
namefirst	text
namelast	text
namegiven	text
weight	text
height	text
bats	text
throws	text

View schema

The execution of the wrapper returns the values contained in the file.

Total rows received: 18589 (shown 150)

playerid	birthyear	birthmonth	birthday	birthcountry	birthstate	birthcity	deathyear	deathmonth	deathday	de
aardsda01	1981	12	27	USA	CO	Denver				
aaronha01	1934	2	5	USA	AL	Mobile				
aaronto01	1939	8	5	USA	AL	Mobile	1984	8	16	US
aasedo01	1954	9	8	USA	CA	Orange				
abadan01	1972	8	25	USA	FL	Palm Beach				
abadfe01	1985	12	17	D.R.	La Romana	La Romana				
abadijo01	1854	11	4	USA	PA	Philadelphia	1905	5	17	US
abbated01	1877	4	15	USA	PA	Latrobe	1957	1	6	US
abbeybe01	1869	11	11	USA	VT	Essex	1962	6	11	US
abbeych01	1866	10	14	USA	NE	Falls City	1926	4	27	US
abbotda01	1862	3	16	USA	OH	Portage	1930	2	12	US

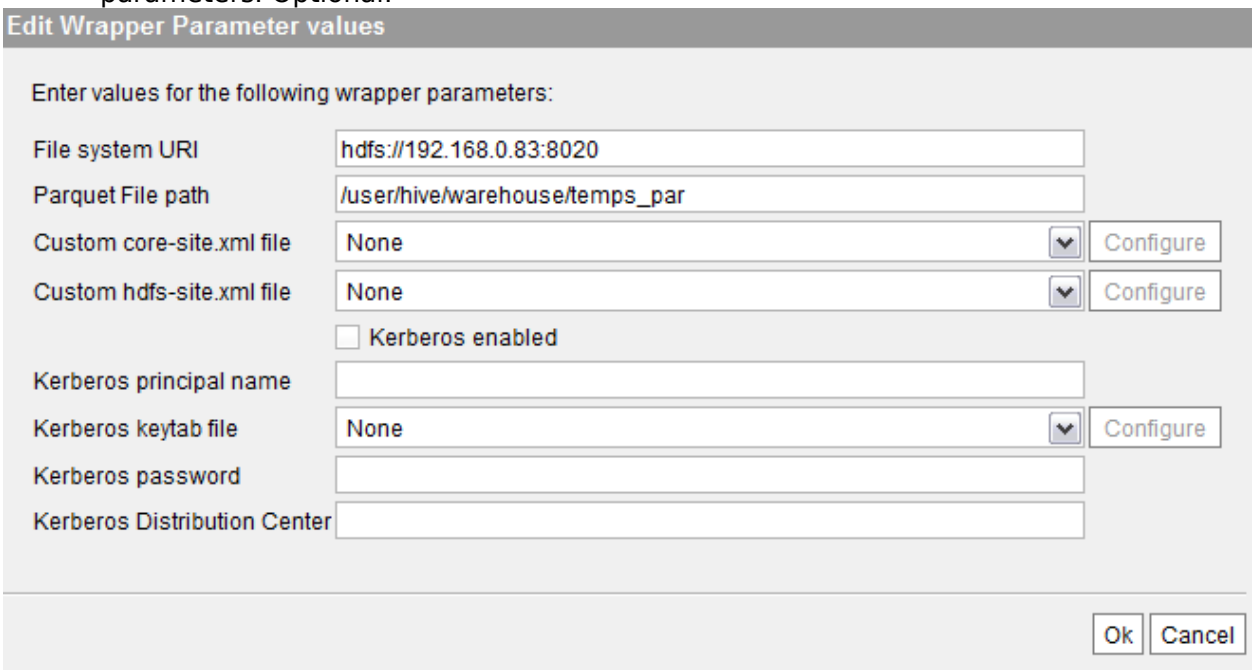
View results

3.6 HDFSPARQUETFILEWRAPPER

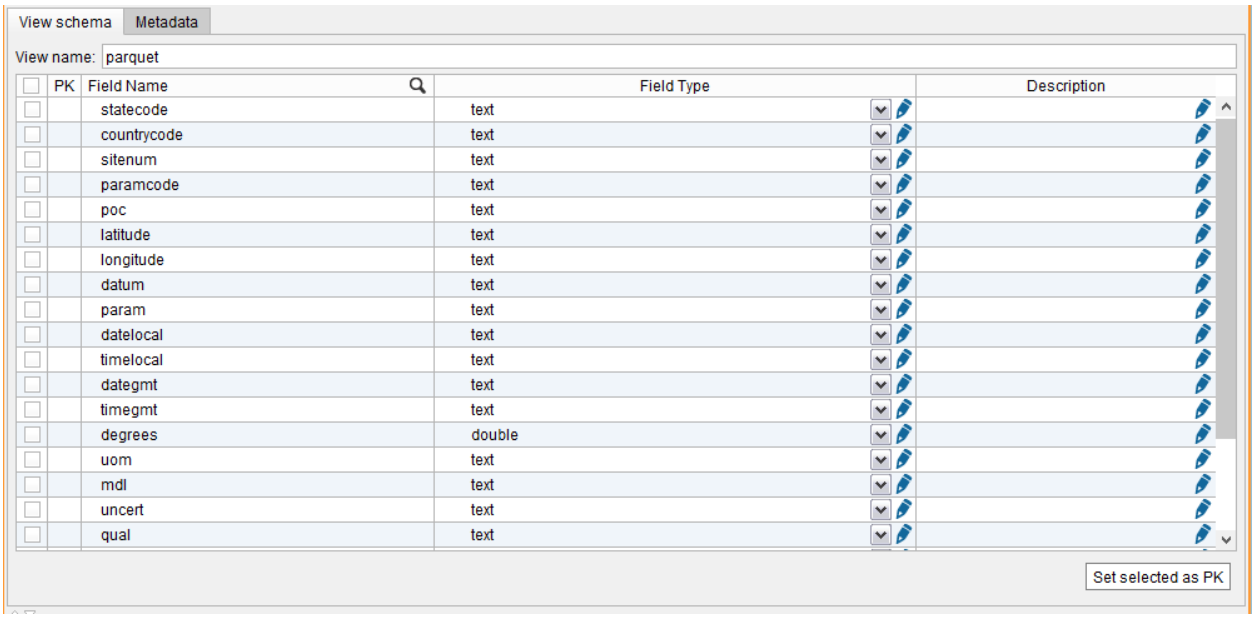
Custom wrapper for reading Parquet files stored in HDFS.

The base views created from the `HDFSParquetFileWrapper` need the following parameters:

- `File system URI`: A URI whose scheme and authority identify the file system.The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
    - For HDFS the URI has the form `hdfs://<ip>:<port>`.
    - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash `/` should be replaced with the string `%2F`).
- `Parquet File Path`: path of the file that we want to read.
- `Custom core-site.xml file`: configuration file that overrides the default core parameters (common to HDFS and MapReduce). Optional.
- `Custom hdfs-site xml file`: configuration file that overrides the default hdfs parameters. Optional.

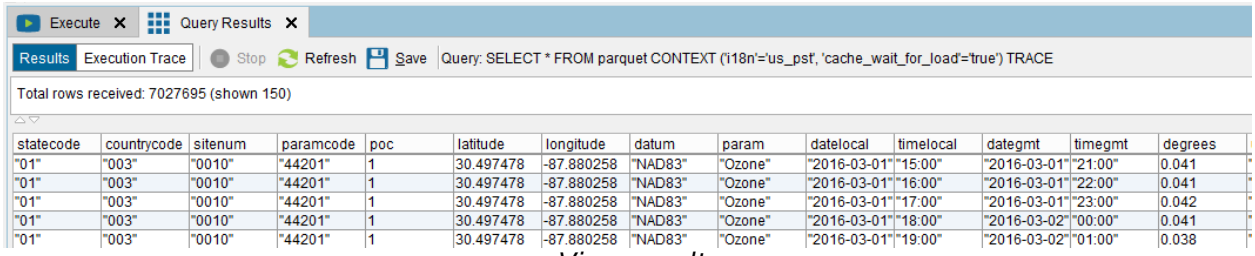


`HDFSParquetWrapper` base view edition



View schema

The execution of the wrapper returns the values contained in the file.



View results

3.6.1 Limitations

- The HDFSParquetFileWrapper **only reads and understands the primitive types of parquet**. Simple types can be read and also the logical types STRING, JSON and BSON. But all other logical types, as well as complex types, are **not** supported.

4 HDFS COMPRESSED FILES

Hadoop is intended for storing large data volumes, so compression becomes a mandatory requirement here. There are different compression formats available like DEFLATE, GZip, BZip2, Snappy and LZO.

Hadoop has native implementations of compression libraries for performance reasons and for non-availability of Java implementations:

Compression format	Java implementation	Native implementation
DEFLATE	Yes	Yes
gzip	Yes	Yes
bzip2	Yes	No
LZO	No	Yes
Snappy	No	Yes

Compression library implementations

For reading HDFS compressed files using the `hdfs-customwrapper` library there are two options:

- Use the Java implementation. `hdfs-customwrapper` handles compressed files transparently.
- Use the native implementation (for performance reasons or for non-availability of Java implementation). `hdfs-customwrapper` must have Hadoop native libraries in the `java.library.path`.

  Hadoop comes with prebuilt native compression libraries for 32- and 64-bit Linux, which could be found in the lib/native directory. For other platforms, the libraries must be compiled, following the instructions on the Hadoop wiki at http://wiki.apache.org/hadoop/NativeHadoop.

5 SECURE CLUSTER WITH KERBEROS

The configuration required for accessing a Hadoop cluster with Kerberos enabled is the same as the one needed to access HDFS and, additionally, the user must supplied the Kerberos credentials.

The Kerberos parameters are:

- `Kerberos enabled`: Check it when accessing a Hadoop cluster with Kerberos enabled (required).
- `Kerberos principal name`: Kerberos v5 Principal name to access HDFS, e.g. primary/instance@realm (optional).
- `Kerberos keytab file`: Keytab file containing the key of the Kerberos principal (optional).
- `Kerberos password`: Password associated with the principal (optional).
- `Kerberos Distribution Center`: Kerberos Key Distribution Center (optional).

`hdfs-customwrapper` provides **three ways** for accessing a kerberized Hadoop cluster:

1. The client has a valid Kerberos ticket in the **ticket cache** obtained, for example, using the `kinit` command in the Kerberos Client.
   In this case only the `Kerberos enabled` parameter should be checked. The HDFS wrapper would use the Kerberos ticket to authenticate itself against the Hadoop cluster.

2. The client does not have a valid Kerberos ticket in the ticket cache. In this case you should provide the `Kerberos principal name` parameter and
   2.1. `Kerberos keytab file` parameter or
   2.2. `Kerberos password` parameter.

In all these **three scenarios** the **`krb5.conf` file should be present in the file system**. Below there is an example of the Kerberos configuration file:

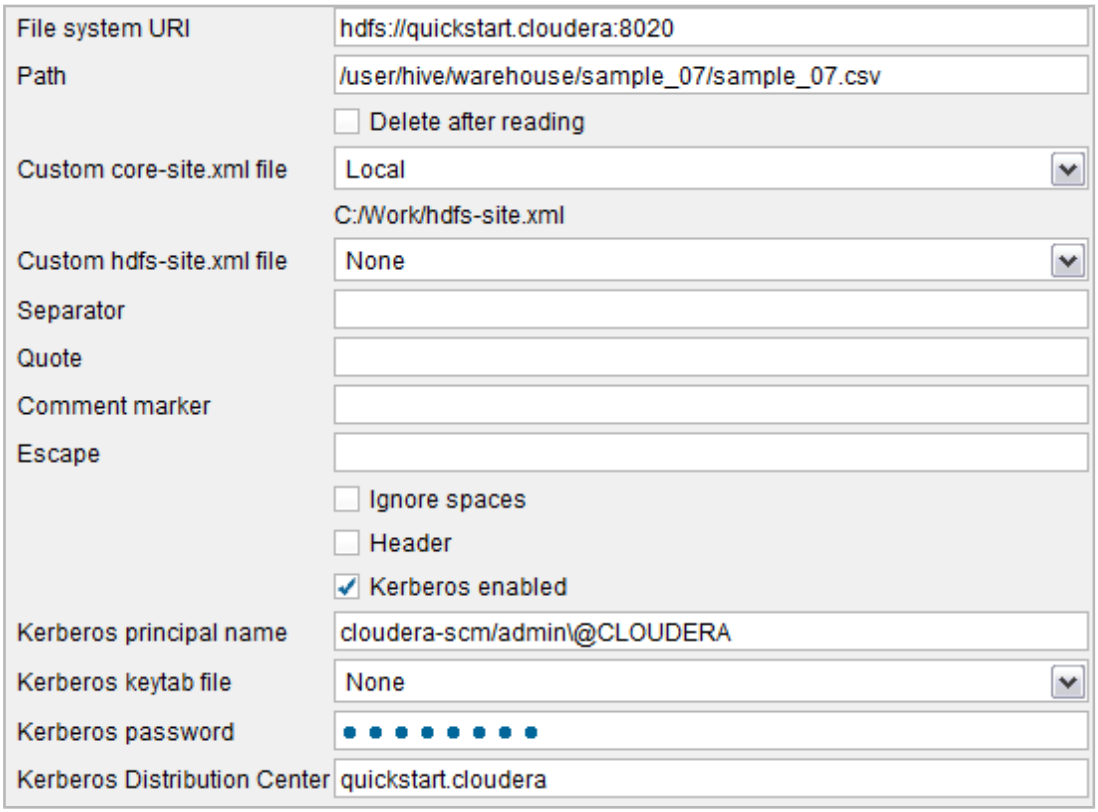
```
[libdefaults]
 renew_lifetime = 7d
 forwardable = true
 default_realm = EXAMPLE.COM
 ticket_lifetime = 24h
 dns_lookup_realm = false
 dns_lookup_kdc = false


[domain_realm]
 sandbox.hortonworks.com = EXAMPLE.COM
 cloudera = CLOUDERA
```

```
[realms]
 EXAMPLE.COM = {
  admin_server = sandbox.hortonworks.com
  kdc = sandbox.hortonworks.com
 }

CLOUDERA = {
 kdc = quickstart.cloudera
 admin_server = quickstart.cloudera
 max_renewable_life = 7d 0h 0m 0s
 default_principal_flags = +renewable
}

[logging]
 default = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log
 kdc = FILE:/var/log/krb5kdc.log
```

The algorithm to locate the `krb5.conf` file is the following:

- If the system property `java.security.krb5.conf` is set, its value is assumed to specify the path and file name.
- If that system property value is not set, then the configuration file is looked for in the directory
  - `<java-home>\lib\security` (Windows)
  - `<java-home>/lib/security` (Solaris and Linux)
- If the file is still not found, then an attempt is made to locate it as follows:
  - `/etc/krb5/krb5.conf` (Solaris)
  - `c:\winnt\krb5.ini` (Windows)
  - `/etc/krb5.conf` (Linux)

There is an **exception**. If you are planning to create HDFS views that use the **same Key Distribution Center and the same realm** the `Kerberos Distribution Center` parameter can be provided instead of having the krb5.conf file in the file system.



View edition

6 SOFTWARE REQUIREMENTS

`hdfs-customwrapper` has been tested in **Cloudera** QuickStart VM 5.8 and in **Hortonworks** Sandbox using Hadoop v2.7.3 and Avro v1.8.1.

`hdfs-customwrapper` has been tested in **Amazon S3** too, for `HDFSDelimitedTextFileWrapper`, `HDFSSequenceFileWrapper`, `HDFSMapFileWrapper`, `HDFSAvroFileWrapper`, `WebHDFSFileWrapper`, using Hadoop v2.7.3 and Avro v1.8.1. For more information see http://aws.amazon.com/es/s3/

7 TROUBLESHOOTING

Symptom

Error message: “Host Details : local host is: "<your domain/your IP>"; destination host is: "<hadoop IP":hadoop port>”.

Resolution

It is a version mismatch problem. Hadoop server version is **older** than the version distributed in the custom wrapper artifact `denodo-hdfs-customwrapper-5.5-xxx-jar-with-dependencies`, which is Hadoop v2.6.0.

To solve the problem you should use the custom wrapper artifact `denodo-hdfs-customwrapper-5.5-xxx` and copy the Hadoop server libraries to the `$DENODO_PLATFORM_HOME/extensions/thirdparty/lib` directory.

Symptom

Error message: “Server IPC version X cannot communicate with client version Y”.

Resolution

It is a version mismatch problem. Hadoop server version is **newer** than the version distributed in the custom wrapper artifact `denodo-hdfs-customwrapper-5.5-xxx-jar-with-dependencies`, which is Hadoop v2.6.0.

To solve the problem you should use the custom wrapper artifact `denodo-hdfs-customwrapper-5.5-xxx` and copy the Hadoop server libraries to the `$DENODO_PLATFORM_HOME/extensions/thirdparty/lib` directory.

Symptom

Error message: “SIMPLE authentication is not enabled. Available:[TOKEN, KERBEROS]”.

Resolution

You are trying to connect to a Kerberos-enabled Hadoop cluster. You should configure the custom wrapper accordingly. See Secure cluster with Kerberos section for **configuring Kerberos** on this custom wrapper.

Symptom

Error message: “Cannot get Kerberos service ticket: KrbException: Server not found in Kerberos database (7) ”.

Resolution

Check that `nslookup` is returning the fully qualified hostname of the KDC. If not, modify the `/etc/hosts` of the client machine for the KDC entry to be of the form "`IP address fully.qualified.hostname alias`".

Symptom

Error message: “Invalid hostname in URI `s3n://<id>:<secret>@<bucket>`”.

Resolution

Check your bucket name: underscores are not permitted. Also check your secret key, if it contains forward slashes request a new id and secret from the AWS site until you get one
that doesn't have a forward slash.

Symptom

Error message: "Error accessing Parquet file: Could not read footer: java.io.IOException: Could not read footer for file FileStatus{path= hdfs://serverhdfs/apps/hive/warehouse/parquet/.hive-staging_hive_2017-03-06_08-/-ext-10000; isDirectory=true; modification_time=1488790684826; access_time=0; owner=hive; group=hdfs; permission=rwxr-xr-x; isSymlink=false}"

Resolution

Hive could be store metadata into parquet file folder. You can check in the error message, if the custom wrapper is trying to access to any metadata. In the error of the example of the symptom you can see that it is accessing a folder called .hive-staging*. The solution is configure Hive to store these metadata in other location.