# Denodo HDFS Custom Wrapper

## Revision 20140925

NOTE

This document is confidential and proprietary of **Denodo Technologies**.
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2014
Denodo Technologies Proprietary and Confidential

# CONTENTS

# 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)
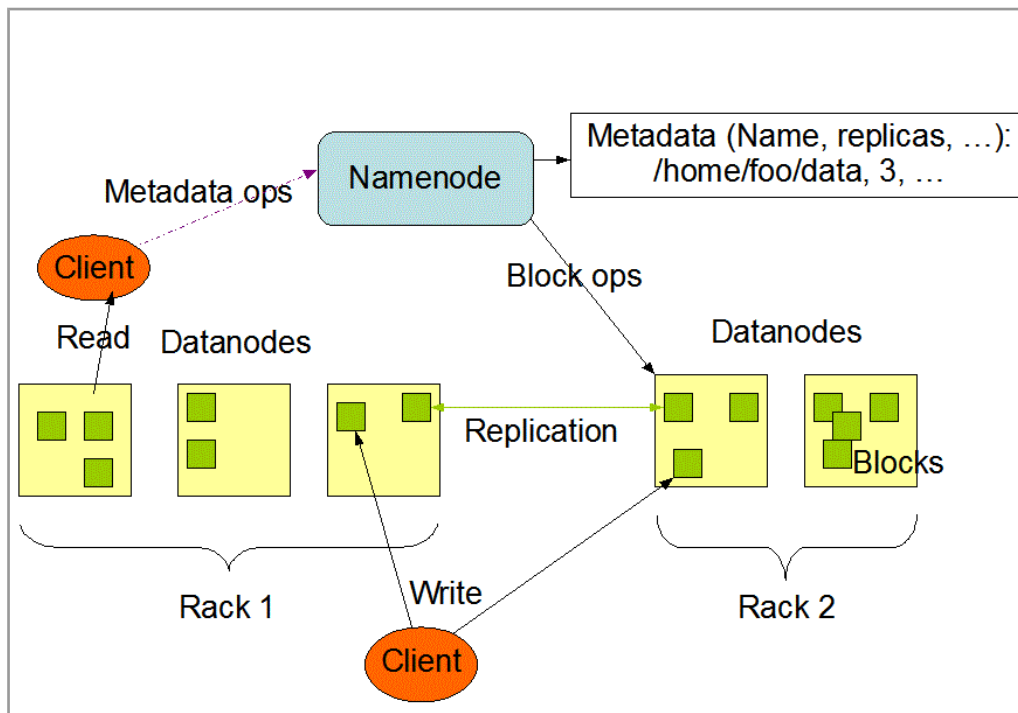- Sequence files
- Map files
- Avro files

## 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 key/value pairs. The key/value pairs are delimited by a separator 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.
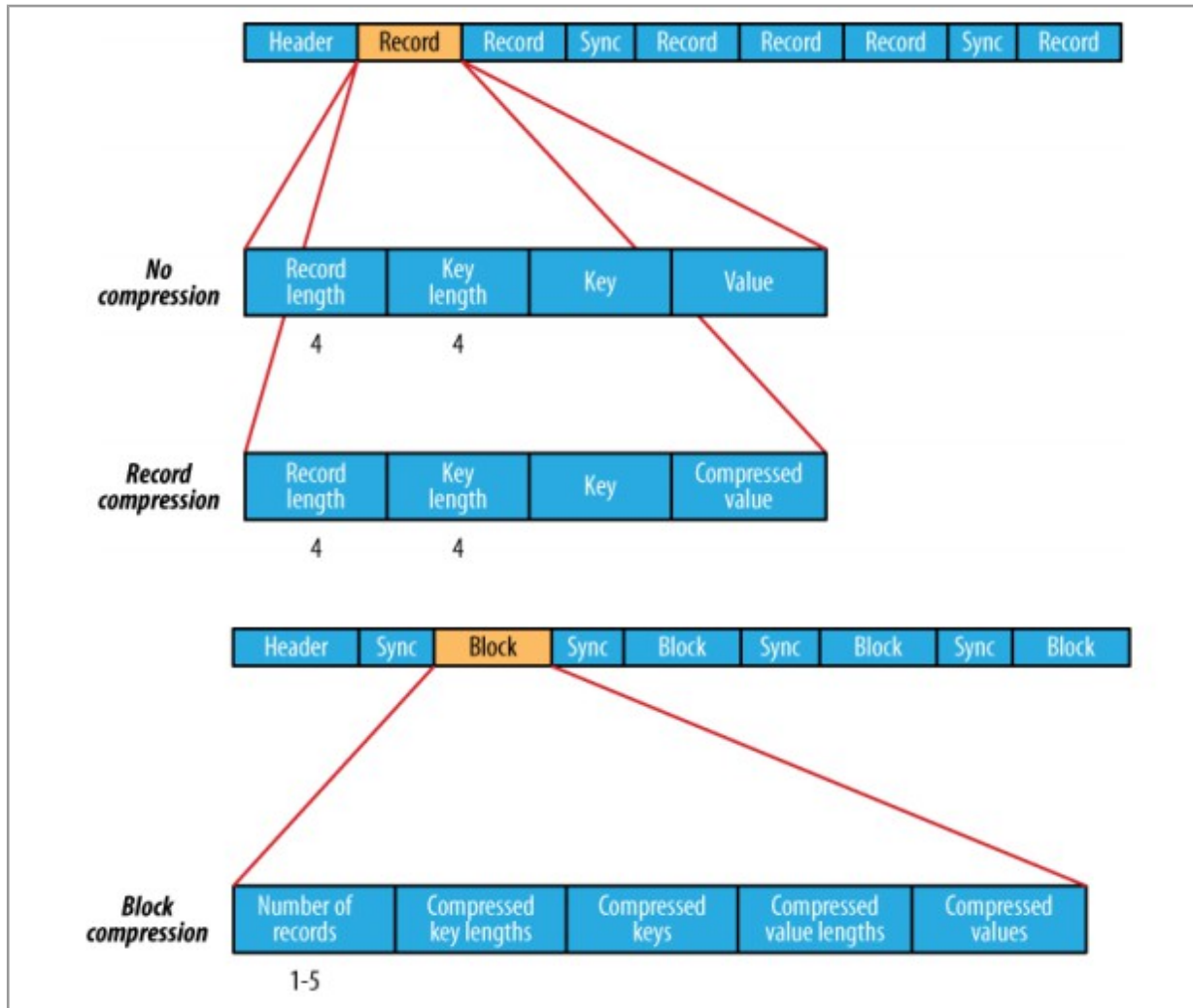
*Sequence file formats*

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.

Denodo Technologies North America & APAC: 530 Lytton Avenue, Suite 301. Palo Alto, CA 94301. USA
Denodo Technologies Iberia & LatinoAmérica:C/ Montalbán, 5. 28014 Madrid. SPAIN
Denodo Technologies EMEA: 17th Floor, Portland House, Bressenden Place, London SW1E 5RS. UK

www.denodo.com

- 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.
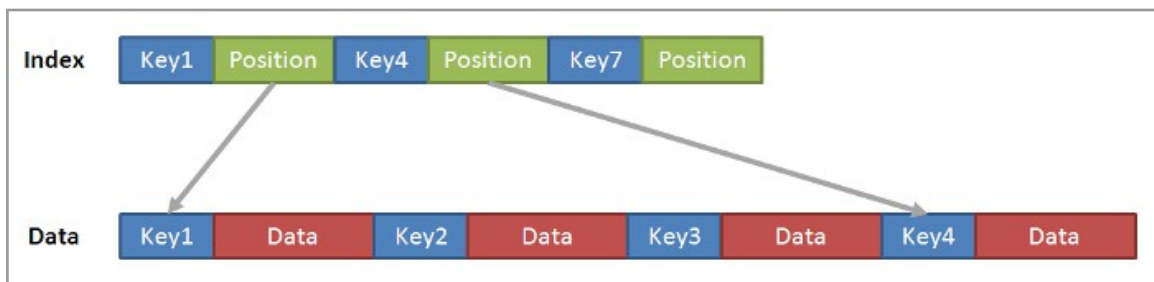


*Map file format*

## 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.

# 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`

## 3.1  HDFSDELIMITEDTEXTFILEWRAPPER

Custom wrapper for reading key/value delimited text files stored in HDFS.

The base views created from the `HDFSDelimitedTextFileWrapper` need the following **mandatory** 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.
- `Separator`: delimiter between the keys and values.

*HDFSDelimitedTextFileWrapper* 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.



*View results*

## 3.2   HDFSSEQUENCEFILEWRAPPER

Custom wrapper for reading sequence files stored in HDFS.

The base views created from the `HDFSSequenceFileWrapper` need the following **mandatory** 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.
- `Key class`: key class name implementing `org.apache.hadoop.io.Writable` interface.
- `Value class`: value class name implementing `org.apache.hadoop.io.Writable` interface.

| | |
|---|---|
| File system URI | s3n://AKIAJXVIEYD2Q74CGMTA:SALdz9RIETUxC( |
| Path | /sequencefile/sequence.seq |
| Delete after reading | ☐ |
| Key class | org.apache.hadoop.io.IntWritable |
| Value class | org.apache.hadoop.io.Text |
| Kerberos enabled | ☐ |
| Kerberos principal name | |
| Kerberos keytab file | None ▾    Configure |
| Kerberos password | |
| Kerberos Distribution Center | |

*HDFSSequenceFileWrapper* base view edition

| sequence_ds_file | |
|---|---|
| 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 |
| 88 | Five, six, pick up sticks |
| 87 | Seven, eight, lay them straight |
| 86 | Nine, ten, a big fat hen |
| 85 | One, two, buckle my shoe |
| 84 | Three, four, shut the door |
| 83 | Five, six, pick up sticks |
| 82 | Seven, eight, lay them straight |
| 81 | Nine, ten, a big fat hen |
| 80 | One, two, buckle my shoe |

*View results*

## 3.3  HDFSMAPFILEWRAPPER

Custom wrapper for reading map files stored in HDFS.

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

Denodo Technologies North America & APAC: 530 Lytton Avenue, Suite 301. Palo Alto, CA 94301. USA
Denodo Technologies Iberia & LatinoAmérica:C/ Montalbán, 5. 28014 Madrid. SPAIN
Denodo Technologies EMEA: 17th Floor, Portland House, Bressenden Place, London SW1E 5RS. UK

www.denodo.com

- `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.
- `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.

| | |
|---|---|
| File system URI | hdfs://192.168.132.130:8020 |
| Path | /user/guest/map |
| Delete after reading | ☐ |
| Key class | org.apache.hadoop.io.IntWritable |
| Value class | org.apache.hadoop.io.Text |
| Kerberos enabled | ☐ |
| Kerberos principal name | |
| Kerberos keytab file | None ▾   Configure |
| Kerberos password | |
| Kerberos Distribution Center | |

*HDFSMapFileWrapper base view edition*

| map_ds_file | |
|---|---|
| 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: 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 |
| 13 | Five, six, pick up sticks |
| 14 | Seven, eight, lay them straight |
| 15 | Nine, ten, a big fat hen |
| 16 | One, two, buckle my shoe |
| 17 | Three, four, shut the door |
| 18 | Five, six, pick up sticks |
| 19 | Seven, eight, lay them straight |
| 20 | Nine, ten, a big fat hen |
| 21 | One, two, buckle my shoe |
| 22 | Three, four, shut the door |

*View results*

## 3.4   HDFSAVROFILEWRAPPER

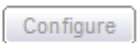Custom wrapper for reading Avro files stored in HDFS.

The base views created from the `HDFSAvroFileWrapper` need the following mandatory 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.

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.

| | |
|---|---|
| File system URI | hdfs://melkus.denodo.com:8020 |
| Avro schema path | /user/test/schema.avsc |
| Avro schema JSON | |
| Delete after reading | ☐ |
| Kerberos principal name | |
| Kerberos keytab file | None ⌄  Configure |
| Kerberos password | |
| Kerberos Distribution Center | |

*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*

Denodo Technologies North America & APAC: 530 Lytton Avenue, Suite 301. Palo Alto, CA 94301. USA
Denodo Technologies Iberia & LatinoAmérica:C/ Montalbán, 5. 28014 Madrid. SPAIN
Denodo Technologies EMEA: 17th Floor, Portland House, Bressenden Place, London SW1E 5RS. UK

www.denodo.com

| avro_ds_file | |
|---|---|
| avrofilepath | text |
| ⊟ doc | ⊞ avro_ds_file_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/test/file.avro'
```

| avrofilepath | doc |
|---|---|
| /user/test/file.avro | [Register]... |
| /user/test/file.avro | [Register]... |

RESULT -> doc

| id | user_frien... | user_locati... | user_desc... | user_statu... | user_follo... | user_name | user_scre... | created_at | text | retweet_c... | retweeted | in_reply_to... | sou |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1234567890 | 111 | Palo Alto | desc1 | 11111 | 111 | name1 | fake_user1 | 1985-09-0... | sample tw... | 0 | false | 0 | href |

*View results*

After applying a flattening operation results are as follows.

| avrofilepath | id | user_f... | user_locati... | user_d... | user_statu... | user_follo... | user_name | user_scre... | created_at | text | ret... | retwe... | in_r... | source | in_r... | me... | expar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| /user/test/fil... | 1234567890 | 111 | Palo Alto | desc1 | 11111 | 111 | name1 | fake_user1 | 1985-09-0... | sample tw... | 0 | false | 0 | href="http://... | 0 | <nul... | <null> |
| /user/test/fil... | 2345678901 | 222 | San Franci... | desc2 | 222222 | 222 | name2 | fake_user2 | 1985-09-0... | sample tw... | 0 | false | 0 | href="http://... | 0 | <nul... | <null> |

*Flattened results*

## 3.5   WEBHDFSFILEWRAPPER

Custom wrapper for reading key/value 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 mandatory 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.
- `Path`: input path for the delimited file.
- `Separator`: delimiter between the keys and values.
- `Delete after reading`: Requests that the file or directory denoted by the path be deleted when the wrapper terminates.

Optional parameter:

- `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.

| | |
|---|---|
| Host IP | melkus.denodo.com |
| Host port | 50070 |
| User | |
| Path | /user/test/text |
| Separator | &#124; |
| Delete after reading | ☐ |

*WebHDFSFileWrapper base view edition*

| webhdfs_ds_file | |
|---|---|
| key | text |
| value | text |

*View schema*

The execution of the wrapper returns the key/value pairs contained in the file.

Total rows received: 5 (shown 5)

| 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 |

*View results*

# 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.

  In the Cloudera QuickStart VM (see Software requirements section) native libraries could be found at `/opt/cloudera/parcels/CDH-5.0.0-1.cdh5.0.0.p0.47/lib/hadoop/lib/native/`.

# 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.
- `Kerberos principal name`: Kerberos v5 Principal name to access HDFS, e.g. primary/instance@realm
- `Kerberos keytab file`: Keytab file containing the key of the Kerberos principal.
- `Kerberos password`: Password associated with the principal.
- `Kerberos Distribution Center`: Kerberos Key Distribution Center.

`hdfs-customwrapper` provides two 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 this scenario all the Kerberos configuration parameters should be provided. `Kerberos keytab file` and `Kerberos password` are **mutually exclusive**.

Denodo Technologies North America & APAC: 530 Lytton Avenue, Suite 301. Palo Alto, CA 94301. USA
Denodo Technologies Iberia & LatinoAmérica:C/ Montalbán, 5. 28014 Madrid. SPAIN
Denodo Technologies EMEA: 17th Floor, Portland House, Bressenden Place, London SW1E 5RS. UK

www.denodo.com

*View edition*

# 6  SOFTWARE REQUIREMENTS

`hdfs-customwrapper` has been tested in Cloudera QuickStart VM 5.0.0 using Hadoop v2.4.0 and Avro v1.7.4.

**Cloudera** provides a virtual machine that gives a working Apache Hadoop environment out-of-the-box. It can be downloaded from:
http://www.cloudera.com/content/support/en/downloads.html

`hdfs-customwrapper` has been tested in Amazon S3 too, using Hadoop v2.4.0 and Avro v1.7.4. 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-5.0-xxx-jar-with-dependencies`, which is Hadoop v2.4.0.

To solve the problem you should use the custom wrapper artifact `denodo-hdfs-customwrapper-5.0-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-5.0-xxx-jar-with-dependencies`, which is Hadoop v2.4.0.

To solve the problem you should use the custom wrapper artifact `denodo-hdfs-customwrapper-5.0-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.