HDFS

HDFS Component

Available as of Camel 2.8

The **hdfs** component enables you to read and write messages from/to an HDFS file system. HDFS is the distributed file system at the heart of Hadoop.

Maven users will need to add the following dependency to their `pom.xml` for this component:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-hdfs</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

URI format

```
hdfs://hostname[:port][/path][?options]
```

You can append query options to the URI in the following format, `?option=value&option=value&...`
The path is treated in the following way:

1. as a consumer, if it's a file, it just reads the file, otherwise if it represents a directory it scans all the file under the path satisfying the configured pattern. All the files under that directory must be of the same type.
2. as a producer, if at least one split strategy is defined, the path is considered a directory and under that directory the producer creates a different file per split named using the configured UuidGenerator.

When consuming from hdfs then in normal mode, a file is split into chunks, producing a message per chunk. You can configure the size of the chunk using the chunkSize option. If you want to read from hdfs and write to a regular file using the file component, then you can use the fileMode=Append to append each of the chunks together.

Options

Name	Default Value	Description
`overwrite`	`true`	The file can be overwritten
`append`	`false`	Append to existing file. Notice that not all HDFS file systems support the append option.
`bufferSize`	`4096`	The buffer size used by HDFS
`replication`	`3`	The HDFS replication factor
`blockSize`	`67108864`	The size of the HDFS blocks
`fileType`	`NORMAL_FILE`	It can be SEQUENCE_FILE, MAP_FILE, ARRAY_FILE, or BLOOMMAP_FILE, see Hadoop
`fileSystemType`	`HDFS`	It can be LOCAL for local filesystem
`keyType`	`NULL`	The type for the key in case of sequence or map files. See below.
`valueType`	`TEXT`	The type for the key in case of sequence or map files. See below.
`splitStrategy`		A string describing the strategy on how to split the file based on different criteria. See below.

`openedSuffix`	`opened`	When a file is opened for reading/writing the file is renamed with this suffix to avoid to read it during the writing phase.
`readSuffix`	`read`	Once the file has been read is renamed with this suffix to avoid to read it again.
`initialDelay`	`0`	For the consumer, how much to wait (milliseconds) before to start scanning the directory.
`delay`	`0`	The interval (milliseconds) between the directory scans.
`pattern`	`*`	The pattern used for scanning the directory
`chunkSize`	`4096`	When reading a normal file, this is split into chunks producing a message per chunk.
`connectOnStartup`	`true`	**Camel 2.9.3/2.10.1:** Whether to connect to the HDFS file system on starting the producer/consumer. If `false` then the connection is created on-demand. Notice that HDFS may take up till 15 minutes to establish a connection, as it has hardcoded 45 x 20 sec redelivery. By setting this option to `false` allows your application to startup, and not block for up till 15 minutes.
`owner`		**Camel 2.13/2.12.4:** The file owner must match this owner for the consumer to pickup the file. Otherwise the file is skipped.

KeyType and ValueType

- NULL it means that the key or the value is absent
- BYTE for writing a byte, the java Byte class is mapped into a BYTE
- BYTES for writing a sequence of bytes. It maps the java ByteBuffer class
- INT for writing java integer
- FLOAT for writing java float
- LONG for writing java long
- DOUBLE for writing java double
- TEXT for writing java strings

BYTES is also used with everything else, for example, in Camel a file is sent around as an InputStream, int this case is written in a sequence file or a map file as a sequence of bytes.

Splitting Strategy

In the current version of Hadoop opening a file in append mode is disabled since it's not very reliable. So, for the moment, it's only possible to create new files. The Camel HDFS endpoint tries to solve this problem in this way:

- If the split strategy option has been defined, the hdfs path will be used as a directory and files will be created using the configured UuidGenerator
- Every time a splitting condition is met, a new file is created.
  The splitStrategy option is defined as a string with the following syntax:
  splitStrategy=<ST>:<value>,<ST>:<value>,*

where <ST> can be:

- BYTES a new file is created, and the old is closed when the number of written bytes is more than <value>
- MESSAGES a new file is created, and the old is closed when the number of written messages is more than <value>
- IDLE a new file is created, and the old is closed when no writing happened in the last <value> milliseconds

note that this strategy currently requires either setting an IDLE value or setting the HdfsConstants.HDFS_CLOSE header to false to use the BYTES/MESSAGES configuration...otherwise, the file will be closed with each message

for example:

```
hdfs://localhost/tmp/simple-file?splitStrategy=IDLE:1000,BYTES:5
```

it means: a new file is created either when it has been idle for more than 1 second or if more than 5 bytes have been written. So, running `hadoop fs -ls /tmp/simple-file` you'll see that multiple files have been created.

Message Headers

The following headers are supported by this component:

Producer only

Header	Description
`CamelFileName`	**Camel 2.13:** Specifies the name of the file to write (relative to the endpoint path). The name can be a `String` or an Expression object. Only relevant when not using a split strategy.

Controlling to close file stream

Available as of Camel 2.10.4

When using the HDFS producer **without** a split strategy, then the file output stream is by default closed after the write. However you may want to keep the stream open, and only explicitly close the stream later. For that you can use the header `HdfsConstants.HDFS_CLOSE` (value = `"CamelHdfsClose"`) to control this. Setting this value to a boolean allows you to explicit control whether the stream should be closed or not.

Notice this does not apply if you use a split strategy, as there are various strategies that can control when the stream is closed.

Using this component in OSGi

This component is fully functional in an OSGi environment, however, it requires some actions from the user. Hadoop uses the thread context class loader in order to load resources. Usually, the thread context classloader will be the bundle class loader of the bundle that contains the routes. So, the default configuration files need to be visible from the bundle class loader. A typical way to deal with it is to keep a copy of core-default.xml in your bundle root. That file can be found in the hadoop-common.jar.