hbase

HBase Component

Available as of Camel 2.10

This component provides an idempotent repository, producers and consumers for Apache HBase.

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

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

Apache HBase Overview

HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable: A Distributed Storage System for Structured Data. You can use HBase when you need random, realtime read/write access to your Big Data. More information at Apache HBase.

Camel and HBase

When using a datastore inside a camel route, there is always the challenge of specifying how the camel message will stored to the datastore. In document based stores things are more easy as the message body can be directly mapped to a document. In relational databases an ORM solution can be used to map properties to columns etc. In column based stores things are more challenging as there is no standard way to perform that kind of mapping.

HBase adds two additional challenges:

- HBase groups columns into families, so just mapping a property to a column using a name convention is just not enough.
- HBase doesn't have the notion of type, which means that it stores everything as byte[] and doesn't know if the byte[] represents a String, a Number, a serialized Java object or just binary data.

To overcome these challenges, camel-hbase makes use of the message headers to specify the mapping of the message to HBase columns. It also provides the ability to use some camel-hbase provided classes that model HBase data and can be easily convert to and from xml/json etc.
Finally it provides the ability to the user to implement and use his own mapping strategy.

Regardless of the mapping strategy camel-hbase will convert a message into an org.apache.camel.component.hbase.model.HBaseData object and use that object for its internal operations.

Configuring the component

The HBase component can be provided a custom HBaseConfiguration object as a property or it can create an HBase configuration object on its own based on the HBase related resources that are found on classpath.

```
    <bean id="hbase" class="org.apache.camel.component.hbase.HBaseComponent">
        <property name="configuration" ref="config"/>
    </bean>
```

If no configuration object is provided to the component, the component will create one. The created configuration will search the class path for an hbase-site.xml file, from which it will draw the configuration. You can find more information about how to configure HBase clients at: HBase client configuration and dependencies

HBase Producer

As mentioned above camel provides producer endpoints for HBase. This allows you to store, delete, retrieve or query data from HBase using your camel routes.

```
hbase://table[?options]
```

where **table** is the table name.

The supported operations are:

- Put
- Get

- Delete
- Scan

Supported URI options on producer

Name	Default Value	Description
`operation`	`CamelHBasePut`	The HBase operation to perform. **Supported values**: `CamelHBasePut`, `CamelHBaseGet`, `CamelHBaseDelete`, and `CamelHBaseScan`.
`maxResults`	`100`	The maximum number of rows to scan.**Supported operations**: `CamelHBaseScan`.
`mappingStrategyName`	`header`	The strategy to use for mapping Camel messages to HBase columns. Supported values: `header`, or `body`.
`mappingStrategyClassName`	`null`	The class name of a custom mapping strategy implementation.
`filters`	`null`	A list of filters. **Supported operations**: `CamelHBaseScan`.
userGroupInformation	UserGroupInformation	**Camel 2.17:** Defines privileges to communicate with HBase such as using kerberos
row.xxx	null	**Camel 2.17:** To map the key/values to the HBaseRow model. From **Camel 2.17** onwards the mapping requires to use row. as prefix. The keys is listed below in the header mapping table. An example: `row.family=info&row.qualifier=firstName&row.family2=birthdate&row.qualifier2=year`

Header mapping options:

Name	Default Value	Description
`rowId`		The id of the row. This has limited use as the row usually changes per Exchange.
`rowType`	String	The type to covert row id to. **Supported operations**: `CamelHBaseScan`.
`family`		The column family. **Supports** a number suffix for referring to more than one columns
`qualifier`		The column qualifier. **Supports** a number suffix for referring to more than one columns
`value`		The value. **Supports** a number suffix for referring to more than one columns
`valueType`	String	The value type. Supports a number suffix for referring to more than one columns. **Supported operations**: `CamelHBaseGet`, and `CamelHBaseScan`.

Put Operations.

HBase is a column based store, which allows you to store data into a specific column of a specific row. Columns are grouped into families, so in order to specify a column you need to specify the column family and the qualifier of that column. To store data into a specific column you need to specify both the column and the row.

The simplest scenario for storing data into HBase from a camel route, would be to store part of the message body to specified HBase column.

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier"/>
        </route>
```

The route above assumes that the message body contains an object that has an id and value property and will store the content of value in the HBase column myfamily:myqualifier in the row specified by id. If we needed to specify more than one column/value pairs we could just specify additional column mappings. Notice that you must use numbers from the 2nd header onwards, eg RowId2, RowId3, RowId4, etc. Only the 1st header does not have the number 1.

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row 1st column -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Row 2nd column -->
            <setHeader headerName="CamelHBaseRowId2">
                <el>${in.body.id}</el>
            </setHeader>
            <!-- Set the HBase Value for 1st column -->
            <setHeader headerName="CamelHBaseValue">
                <el>${in.body.value}</el>
            </setHeader>
            <!-- Set the HBase Value for 2nd column -->
            <setHeader headerName="CamelHBaseValue2">
                <el>${in.body.othervalue}</el>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBasePut&amp;family=myfamily&amp;qualifier=myqualifier&amp;
family2=myfamily&amp;qualifier2=myqualifier2"/>
        </route>
```

It is important to remember that you can use uri options, message headers or a combination of both. It is recommended to specify constants as part of the uri and dynamic values as headers. If something is defined both as header and as part of the uri, the header will be used.

Get Operations.

A Get Operation is an operation that is used to retrieve one or more values from a specified HBase row. To specify what are the values that you want to retrieve you can just specify them as part of the uri or as message headers.

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row of the Get -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseGet&amp;family=myfamily&amp;qualifier=myqualifier&amp;
valueType=java.lang.Long"/>
            <to uri="log:out"/>
        </route>
```

In the example above the result of the get operation will be stored as a header with name CamelHBaseValue.

Delete Operations.

You can also you camel-hbase to perform HBase delete operation. The delete operation will remove an entire row. All that needs to be specified is one or more rows as part of the message headers.

```
        <route>
            <from uri="direct:in"/>
            <!-- Set the HBase Row of the Get -->
            <setHeader headerName="CamelHBaseRowId">
                <el>${in.body.id}</el>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseDelete"/>
        </route>
```

Scan Operations.

A scan operation is the equivalent of a query in HBase. You can use the scan operation to retrieve multiple rows. To specify what columns should be part of the result and also specify how the values will be converted to objects you can use either uri options or headers.

```
        <route>
            <from uri="direct:in"/>
            <to uri="hbase:mytable?operation=CamelHBaseScan&amp;family=myfamily&amp;qualifier=myqualifier&amp;
valueType=java.lang.Long&amp;rowType=java.lang.String"/>
            <to uri="log:out"/>
        </route>
```

In this case its probable that you also also need to specify a list of filters for limiting the results. You can specify a list of filters as part of the uri and camel will return only the rows that satisfy **ALL** the filters.
To have a filter that will be aware of the information that is part of the message, camel defines the ModelAwareFilter. This will allow your filter to take into consideration the model that is defined by the message and the mapping strategy.
When using a ModelAwareFilter camel-hbase will apply the selected mapping strategy to the in message, will create an object that models the mapping and will pass that object to the Filter.

For example to perform scan using as criteria the message headers, you can make use of the ModelAwareColumnMatchingFilter as shown below.

```
        <route>
            <from uri="direct:scan"/>
            <!-- Set the Criteria -->
            <setHeader headerName="CamelHBaseFamily">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier">
                <constant>first</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue">
                <el>in.body.firstName</el>
            </setHeader>
            <setHeader headerName="CamelHBaseFamily2">
                <constant>name</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier2">
                <constant>last</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseValue2">
                <el>in.body.lastName</el>
            </setHeader>
            <!-- Set additional fields that you want to be return by skipping value -->
            <setHeader headerName="CamelHBaseFamily3">
                <constant>address</constant>
            </setHeader>
            <setHeader headerName="CamelHBaseQualifier3">
                <constant>country</constant>
            </setHeader>
            <to uri="hbase:mytable?operation=CamelHBaseScan&amp;filters=#myFilterList"/>
        </route>

        <bean id="myFilters" class="java.util.ArrayList">
            <constructor-arg>
                <list>
                    <bean class="org.apache.camel.component.hbase.filters.ModelAwareColumnMatchingFilter"/>
                </list>
            </constructor-arg>
        </bean>
```

The route above assumes that a pojo is with properties firstName and lastName is passed as the message body, it takes those properties and adds them as part of the message headers. The default mapping strategy will create a model object that will map the headers to HBase columns and will pass that model the the ModelAwareColumnMatchingFilter. The filter will filter out any rows, that do not contain columns that match the model. It is like query by example.

HBase Consumer

The Camel HBase Consumer, will perform repeated scan on the specified HBase table and will return the scan results as part of the message. You can either specify header mapping (default) or body mapping. The latter will just add the org.apache.camel.component.hbase.model.HBaseData as part of the message body.

```
hbase://table[?options]
```

You can specify the columns that you want to be return and their types as part of the uri options:

```
hbase:mutable?family=name&qualifer=first&valueType=java.lang.
String&family=address&qualifer=number&valueType2=java.lang.Integer&rowType=java.lang.Long
```

The example above will create a model object that is consisted of the specified fields and the scan results will populate the model object with values. Finally the mapping strategy will be used to map this model to the camel message.

Supported URI options on consumer

Name	Default Value	Description
`initialDelay`	`1000`	Milliseconds before the first polling starts.
`delay`	`500`	Milliseconds before the next poll.
`useFixedDelay`	`true`	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.
timeUnit	`TimeUnit.MILLISECONDS`	time unit for `initialDelay` and `delay` options.
`runLoggingLevel`	`TRACE`	**Camel 2.8:** The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that.
`operation`	`CamelHBasePut`	The HBase operation to perform. **Supported values**: `CamelHBasePut`, `CamelHBaseGet`, `CamelHBaseDelete`, and `CamelHBaseScan`.
`maxResults`	`100`	The maximum number of rows to scan. **Supported operations:** `CamelHBaseScan`.
`mappingStrategyName`	`header`	The strategy to use for mapping Camel messages to HBase columns. Supported values: `header`, or `body`.
`mappingStrategyClassName`	`null`	The class name of a custom mapping strategy implementation.
`filters`	`null`	A list of filters. **Supported operations**: `CamelHBaseScan`
`remove`	`true`	If the option is true, Camel HBase Consumer will remove the rows which it processes.
userGroupInformation	UserGroupInformation	**Camel 2.17:** Defines privileges to communicate with HBase such as using kerberos

Header mapping options:

Name	Default Value	Description
`rowId`		The id of the row. This has limited use as the row usually changes per Exchange.
`rowType`	String	The type to covert row id to. **Supported operations**: CamelHBaseScan
`family`		The column family. *upports a number suffix for referring to more than one columns
`qualifier`		The column qualifier. *Supports a number suffix for referring to more than one columns
`value`		The value. Supports a number suffix for referring to more than one columns
`rowModel`	String	An instance of org.apache.camel.component.hbase.model.HBaseRow which describes how each row should be modeled

If the role of the rowModel is not clear, it allows you to construct the HBaseRow modle programmatically instead of "describing" it with uri options (such as family, qualifier, type etc).

HBase Idempotent repository

The camel-hbase component also provides an idempotent repository which can be used when you want to make sure that each message is processed only once. The HBase idempotent repository is configured with a table, a column family and a column qualifier and will create to that table a row per message.

```
HBaseConfiguration configuration = HBaseConfiguration.create();
HBaseIdempotentRepository repository = new HBaseIdempotentRepository(configuration, tableName, family,
qualifier);

from("direct:in")
  .idempotentConsumer(header("messageId"), repository)
  .to("log:out);
```

HBase Mapping

It was mentioned above that you the default mapping strategies are **header** and **body** mapping.
Below you can find some detailed examples of how each mapping strategy works.

HBase Header mapping Examples

The header mapping is the default mapping.
To put the value "myvalue" into HBase row "myrow" and column "myfamily:mycolum" the message should contain the following headers:

Header	Value
CamelHBaseRowId	myrow
CamelHBaseFamily	myfamily
CamelHBaseQualifier	myqualifier
CamelHBaseValue	myvalue

To put more values for different columns and / or different rows you can specify additional headers suffixed with the index of the headers, e.g:

Header	Value
CamelHBaseRowId	myrow
CamelHBaseFamily	myfamily
CamelHBaseQualifier	myqualifier
CamelHBaseValue	myvalue
CamelHBaseRowId2	myrow2
CamelHBaseFamily2	myfamily
CamelHBaseQualifier2	myqualifier
CamelHBaseValue2	myvalue2

In the case of retrieval operations such as get or scan you can also specify for each column the type that you want the data to be converted to. For exampe:

Header	Value
CamelHBaseFamily	myfamily
CamelHBaseQualifier	myqualifier
CamelHBaseValueType	Long

Please note that in order to avoid boilerplate headers that are considered constant for all messages, you can also specify them as part of the endpoint uri, as you will see below.

Body mapping Examples

In order to use the body mapping strategy you will have to specify the option mappingStrategy as part of the uri, for example:

```
hbase:mytable?mappingStrategy=body
```

To use the body mapping strategy the body needs to contain an instance of org.apache.camel.component.hbase.model.HBaseData. You can construct t

```
HBaseData data = new HBaseData();
HBaseRow row = new HBaseRow();
row.setId("myRowId");
HBaseCell cell = new HBaseCell();
cell.setFamily("myfamily");
cell.setQualifier("myqualifier");
cell.setValue("myValue");
row.getCells().add(cell);
data.addRows().add(row);
```

The object above can be used for example in a put operation and will result in creating or updating the row with id myRowId and add the value myvalue to the column myfamily:myqualifier.
The body mapping strategy might not seem very appealing at first. The advantage it has over the header mapping strategy is that the HBaseData object can be easily converted to or from xml/json.

See also

- Polling Consumer
- Apache HBase