Krati

Krati Component

Available as of Camel 2.9

This component allows the use krati datastores and datasets inside Camel. Krati is a simple persistent data store with very low latency and high throughput. It is designed for easy integration with read-write-intensive applications with little effort in tuning configuration, performance and JVM garbage collection.

Camel provides a producer and consumer for krati datastore_(key/value engine)_. It also provides an idempotent repository for filtering out duplicate messages.

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

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

URI format

```
krati:[the path of the datastore][?options]
```

The **path of the datastore** is the relative path of the folder that krati will use for its datastore.

You can append query options to the URI in the following format, `?option=value&option=value&...`

Krati URI Options

Name	Default Value	Description
`operation`	CamelKratiPut	**Producer Only**. Specifies the type of operation that will be performed to the datastore. Allowed values are CamelKratiPut, CamelKratiGet, CamelKratiDelete & CamelKratiDeleteAll.
`initialCapacity`	100	The inital capcity of the store.
`keySerializer`	KratiDefaultSerializer	The serializer serializer that will be used to serialize the key.
`valueSerializer`	KratiDefaultSerializer	The serializer serializer that will be used to serialize the value.
`segmentFactory`	ChannelSegmentFactory	The segment factory to use. Allowed instance classes: ChannelSegmentFactory,MemorySegmentFactory, MappedSegmentFactory & WriteBufferSegmentFactory.
`hashFunction`	FnvHashFunction	The hash function to use. Allowed instance classes: FnvHashFunction,Fnv1Hash32,FnvHash64,Fnv1aHash32,Fnv1aHash64, JenkisHashFunction, MurmurHashFunctiom
`maxMessagesPerPoll`		**Camel 2.10.5/2.11.1:** The maximum number of messages which can be received in one poll. This can be used to avoid reading in too much data and taking up too much memory.

You can have as many of these options as you like.

```
krati:/tmp/krati?operation=CamelKratiGet&initialCapacity=10000&keySerializer=#myCustomSerializer
```

For producer endpoint you can override all of the above URI options by passing the appropriate headers to the message.

Message Headers for datastore

Header	Description

`CamelKratiOperation`	The operation to be performed on the datastore. The valid options are • CamelKratiAdd • CamelKratiGet • CamelKratiDelete • CamelKratiDeleteAll
`CamelKratiKey`	The key.
`CamelKratiValue`	The value.

Usage Samples

Example 1: Putting to the datastore.

This example will show you how you can store any message inside a datastore.

```
from("direct:put").to("krati:target/test/producertest");
```

In the above example you can override any of the URI parameters with headers on the message.
Here is how the above example would look like using xml to define our route.

```
        <route>
            <from uri="direct:put"/>
            <to uri="krati:target/test/producerspringtest"/>
        </route>
```

Example 2: Getting/Reading from a datastore

This example will show you how you can read the contnet of a datastore.

```
from("direct:get")
    .setHeader(KratiConstants.KRATI_OPERATION, constant(KratiConstants.KRATI_OPERATION_GET))
    .to("krati:target/test/producertest");
```

In the above example you can override any of the URI parameters with headers on the message.
Here is how the above example would look like using xml to define our route.

```
<route>
     <from uri="direct:get"/>
     <to uri="krati:target/test/producerspringtest?operation=CamelKratiGet"/>
</route>
```

Example 3: Consuming from a datastore

This example will consume all items that are under the specified datastore.

```
    from("krati:target/test/consumertest")
        .to("direct:next");
```

You can achieve the same goal by using xml, as you can see below.

```
<route>
    <from uri="krati:target/test/consumerspringtest"/>
    <to uri="mock:results"/>
</route>
```

Idempotent Repository

As already mentioned this component also offers and idemptonet repository which can be used for filtering out duplicate messages.

```
from("direct://in").idempotentConsumer(header("messageId"), new KratiIdempotentRepositroy("/tmp/idempotent").to
("log://out");
```

See also

Krati Websitre