Aggregate Example

Aggregate Example

Available as of Camel 2.3

About

This example demonstrates the new overhauled Aggregator EIP in Apache Camel 2.3. The example is focused on the persistence support, which means the aggregated messages is stored in a persistent file storage using the new HawtDB component.

How to run

You simply run it using the following maven goal: `mvn camel:run`

How it works

The example is an interactive example where it prompt on the console for you to enter a number and press ENTER. The numbers you enter will then be aggregated and persisted. That means you can at any time hit `ctrl + c` to shutdown Camel. Then you should be able to start the example again and resume where you left.

When you want to *complete* the aggregation you can enter `STOP` as input and Camel will show you the result, which is the sum of all the numbers entered.

The persistent datastore is located in the `data/hawtdb.dat` file. Its automatic created the first time.

Example

For example we start the example for the first time using `mvn camel:run` and then we enter the two numbers 5 and 7 before we shutdown using `ctrl + c`. The relevant console output:

```
[pache.camel.spring.Main.main()] DefaultCamelContext           INFO  Apache Camel 2.3-SNAPSHOT (CamelContext:
camel) started
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel): 5
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel): 7
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel): ^C[                     Thread-2]
Main$HangupInterceptor         INFO  Received hang up - stopping the main instance.
[                     Thread-2] MainSupport                   INFO  Apache Camel 2.3-SNAPSHOT stopping
```

Now we start the example again using `mvn camel:run` and enter the number 3 and then enter `STOP` to see the result. As expected the result is 5+7+3 = 15 as outputted on the console. As you can see the persistence of the aggregated messages ensures we could continue where we stopped.

```
[pache.camel.spring.Main.main()] DefaultCamelContext           INFO  Apache Camel 2.3-SNAPSHOT (CamelContext:
camel) started
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel): 3
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel): STOP
The result is: 15
Enter a number to be added (use STOP to end, and ctrl c to shutdown Camel):
```

Using Aggregator

The example is configured as follows in Spring XML.

Error rendering macro 'code': Invalid value specified for parameter 'java.lang.NullPointerException'

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

  <!-- This is our aggregation strategy for the numbers received as input -->
  <bean id="myStrategy" class="org.apache.camel.example.NumberAggregationStrategy"/>

  <!-- This is the repository to store aggregated messages -->
  <bean id="myRepo" class="org.apache.camel.processor.aggregate.MemoryAggregationRepository"/>

  <!-- This is the Camel route which asks for input and aggregates incoming numbers -->
  <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
      <!-- ask user to enter a number -->
      <from uri="stream:in?promptMessage=Enter a number to be added (enter STOP to end, and Ctrl-C to shutdown
Camel): &amp;promptDelay=1000"/>
      <!-- aggregate the input, use eagerCheckCompletion to let the completionPredicate easily detect the STOP
command -->
      <aggregate strategyRef="myStrategy" aggregationRepositoryRef="myRepo" eagerCheckCompletion="true">
        <!-- aggregate all messages into the same group -->
        <correlationExpression>
          <constant>true</constant>
        </correlationExpression>
        <!-- if end user enters STOP then complete the aggregation -->
        <completionPredicate>
          <simple>${body} contains 'STOP'</simple>
        </completionPredicate>
        <!-- and transform the completed message to a human readable -->
        <transform>
          <simple>The result is: ${body}</simple>
        </transform>
        <!-- which is printed on the console -->
        <to uri="stream:out"/>
      </aggregate>
    </route>

  </camelContext>

</beans>
```

And it has a `AggregationStrategy` to sum the numbers which is done in Java code as:

Error rendering macro 'code': Invalid value specified for parameter 'java.lang.NullPointerException'

```
public class NumberAggregationStrategy implements AggregationStrategy {

    public Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
        if (oldExchange == null) {
            return newExchange;
        }

        // check for stop command
        String input = newExchange.getIn().getBody(String.class);
        if ("STOP".equalsIgnoreCase(input)) {
            return oldExchange;
        }

        Integer num1 = oldExchange.getIn().getBody(Integer.class);
        Integer num2 = newExchange.getIn().getBody(Integer.class);

        // just avoid bad inputs by assuming its a 0 value
        Integer num3 = (num1 != null ? num1 : 0) + (num2 != null ? num2 : 0);
        oldExchange.getIn().setBody(num3);

        return oldExchange;
    }

}
```

See Also

- Examples
- Aggregator
- HawtDB
- LevelDB