Polling Consumer

Polling Consumer

Camel supports implementing the Polling Consumer from the EIP patterns using the PollingConsumer interface which can be created via the Endpoint.createPollingConsumer() method.

In Java:

javaEndpoint endpoint = context.getEndpoint("activemq:my.queue"); PollingConsumer consumer = endpoint.createPollingConsumer(); Exchange exchange = consumer.receive();

The **`ConsumerTemplate`** (discussed below) is also available.

There are three main polling methods on PollingConsumer

Method name	Description
receive()	Waits until a message is available and then returns it; potentially blocking forever
receive(long)	Attempts to receive a message exchange, waiting up to the given timeout and returning null if no message exchange could be received within the time available
receiveNoWait()	Attempts to receive a message exchange immediately without waiting and returning null if a message exchange is not available yet

EventDrivenPollingConsumer Options

The **`EventDrivePollingConsumer`** (the default implementation) supports the following options:

confluenceTableSmall

Option	Default	Description
`pollingConsumerQueueSize`	`1000`	**Camel 2.14/2.13.1/2.12.4:** The queue size for the internal hand-off queue between the polling consumer, and producers sending data into the queue.
`pollingConsumerBlockWhenFull`	`true`	**Camel 2.14/2.13.1/2.12/4:** Whether to block any producer if the internal queue is full.
`pollingConsumerBlockTimeout`	`0`	**Camel 2.16:** To use a timeout (in milliseconds) when the producer is blocked if the internal queue is full. If the value is **`0`** or negative then no timeout is in use. If a timeout is triggered then a **`ExchangeTimedOutException`** is thrown.

Notice that some Camel Components has their own implementation of **`PollingConsumer`** and therefore do not support the options above.

You can configure these options in endpoints URIs, such as shown below:

javaEndpoint endpoint = context.getEndpoint("file:inbox?pollingConsumerQueueSize=50"); PollingConsumer consumer = endpoint.createPollingConsumer(); Exchange exchange = consumer.receive(5000);

ConsumerTemplate

The **`ConsumerTemplate`** is a template much like Spring's **`JmsTemplate`** or **`JdbcTemplate`** supporting the Polling Consumer EIP. With the template you can consume Exchanges from an Endpoint. The template supports the three operations listed above. However, it also includes convenient methods for returning the body, etc **`consumeBody`**.

Example:

Exchange exchange = consumerTemplate.receive("activemq:my.queue");

Or to extract and get the body you can do:

Object body = consumerTemplate.receiveBody("activemq:my.queue");

And you can provide the body type as a parameter and have it returned as the type:

String body = consumerTemplate.receiveBody("activemq:my.queue", String.class);

You get hold of a **`ConsumerTemplate`** from the **`CamelContext`** with the **`createConsumerTemplate`** operation:

ConsumerTemplate consumer = context.createConsumerTemplate();

Using ConsumerTemplate with Spring DSL

With the Spring DSL we can declare the consumer in the **`CamelContext`** with the **`consumerTemplate`** tag, just like the **`ProducerTemplate`**. The example below illustrates this:{snippet:id=e1|lang=xml|url=camel/components/camel-spring/src/test/resources/org/apache/camel/spring/SpringConsumerTemplateTest-context.xml}Then we can get leverage Spring to inject the **`ConsumerTemplate`** in our java class. The code below is part of an unit test but it shows how the consumer and producer can work together.{snippet:id=e1|lang=java|url=camel/components/camel-spring/src/test/java/org/apache/camel/spring/SpringConsumerTemplateTest.java}

Timer Based Polling Consumer

In this sample we use a Timer to schedule a route to be started every 5th second and invoke our bean **`MyCoolBean`** where we implement the business logic for the Polling Consumer. Here we want to consume all messages from a JMS queue, process the message and send them to the next queue.

First we setup our route as:{snippet:id=e1|lang=java|url=camel/tags/camel-2.6.0/components/camel-jms/src/test/java/org/apache/camel/component/jms/JmsTimerBasedPollingConsumerTest.java}And then we have out logic in our bean:{snippet:id=e2|lang=java|url=camel/tags/camel-2.6.0/components/camel-jms/src/test/java/org/apache/camel/component/jms/JmsTimerBasedPollingConsumerTest.java}

Scheduled Poll Components

Quite a few inbound Camel endpoints use a scheduled poll pattern to receive messages and push them through the Camel processing routes. That is to say externally from the client the endpoint appears to use an Event Driven Consumer but internally a scheduled poll is used to monitor some kind of state or resource and then fire message exchanges.

Since this a such a common pattern, polling components can extend the ScheduledPollConsumer base class which makes it simpler to implement this pattern. There is also the Quartz Component which provides scheduled delivery of messages using the Quartz enterprise scheduler.

For more details see:

- PollingConsumer
- Scheduled Polling Components
  - ScheduledPollConsumer
  - Scheduler
  - Atom
  - Beanstalk
  - File
  - FTP
  - hbase
  - iBATIS
  - JPA
  - Mail
  - MyBatis
  - Quartz
  - SNMP
  - AWS-S3
  - AWS-SQS

ScheduledPollConsumer Options

The **`ScheduledPollConsumer`** supports the following options:

confluenceTableSmall

Option	Default	Description
`backoffErrorThreshold`	`0`	**Camel 2.12:** The number of subsequent error polls (failed due some error) that should happen before the **`backoffMultipler`** should kick-in.
`backoffIdleThreshold`	`0`	**Camel 2.12:** The number of subsequent idle polls that should happen before the **`backoffMultipler`** should kick-in.
`backoffMultiplier`	`0`	**Camel 2.12:** To let the scheduled polling consumer back-off if there has been a number of subsequent idles/errors in a row. The multiplier is then the number of polls that will be skipped before the next actual attempt is happening again. When this option is in use then **`backoffIdleThreshold`** and/or **`backoffErrorThreshold`** must also be configured.
`delay`	`500`	Milliseconds before the next poll.

greedy	false	**Camel 2.10.6/2.11.1:** If greedy is enabled, then the **ScheduledPollConsumer** will run immediately again, if the previous run polled 1 or more messages.
initialDelay	1000	Milliseconds before the first poll starts.
pollStrategy		A pluggable **org.apache.camel.PollingConsumerPollingStrategy** allowing you to provide your custom implementation to control error handling usually occurred during the **poll** operation ***before*** an Exchange has been created and routed in Camel. In other words the error occurred while the polling was gathering information, for instance access to a file network failed so Camel cannot access it to scan for files. The default implementation will log the caused exception at **WARN** level and ignore it.
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.
scheduledExecutorService	null	**Camel 2.10:** Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool. This option allows you to share a thread pool among multiple consumers.
scheduler	null	**Camel 2.12:** Allow to plugin a custom **org.apache.camel.spi.ScheduledPollConsumerScheduler** to use as the scheduler for firing when the polling consumer runs. The default implementation uses the **ScheduledExecutorService** and there is a Quartz2, and Spring based which supports CRON expressions. **Notice:** If using a custom scheduler then the options for **initialDelay**, **useFixedDelay**, **timeUnit** and **scheduledExecutorService** may not be in use. Use the text **quartz2** to refer to use the Quartz2 scheduler; and use the text spring to use the Spring based; and use the text **#myScheduler** to refer to a custom scheduler by its id in the Registry. See Quartz2 page for an example.
scheduler.xxx	null	**Camel 2.12:** To configure additional properties when using a custom **scheduler** or any of the Quartz2, Spring based scheduler.
sendEmptyMessageWhenIdle	false	**Camel 2.9:** If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.
startScheduler	true	Whether the scheduler should be auto started.
timeUnit	TimeUnit.MILLISECONDS	Time unit for **initialDelay** and **delay** options.
useFixedDelay		Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details. In **Camel 2.7.x** or older the default value is **false**. From **Camel 2.8**: the default value is **true**.

Using backoff to Let the Scheduler be Less Aggressive

Available as of Camel 2.12

The scheduled Polling Consumer is by default static by using the same poll frequency whether or not there is messages to pickup or not.

From **Camel 2.12**: you can configure the scheduled Polling Consumer to be more dynamic by using **backoff**. This allows the scheduler to skip N number of polls when it becomes idle, or there has been X number of errors in a row. See more details in the table above for the **backoffXXX** options.

For example to let a FTP consumer back-off if its becoming idle for a while you can do:

javafrom("ftp://myserver?username=foo&passowrd=secret?delete=true&delay=5s&backoffMultiplier=6&backoffIdleThreshold=5") .to("bean:processFile");

In this example, the FTP consumer will poll for new FTP files every 5th second. But if it has been idle for 5 attempts in a row, then it will back-off using a multiplier of 6, which means it will now poll every 5 x 6 = 30th second instead. When the consumer eventually pickup a file, then the back-off will reset, and the consumer will go back and poll every 5th second again.

Camel will log at **DEBUG** level using **org.apache.camel.impl.ScheduledPollConsumer** when back-off is kicking-in.

About Error Handling and Scheduled Polling Consumers

ScheduledPollConsumer is scheduled based and its **run** method is invoked periodically based on schedule settings. But errors can also occur when a poll is being executed. For instance if Camel should poll a file network, and this network resource is not available then a **java.io.IOException** could occur. As this error happens **before** any Exchange has been created and prepared for routing, then the regular Error handling in Camel does not apply. So what does the consumer do then? Well the exception is propagated back to the **run** method where its handled. Camel will by default log the exception at **WARN** level and then ignore it. At next schedule the error could have been resolved and thus being able to poll the endpoint successfully.

Using a Custom Scheduler

Available as of Camel 2.12:

The SPI interface **`org.apache.camel.spi.ScheduledPollConsumerScheduler`** allows to implement a custom scheduler to control when the Polling Consumer runs. The default implementation is based on the JDKs **`ScheduledExecutorService`** with a single thread in the thread pool. There is a CRON based implementation in the Quartz2, and Spring components.

For an example of developing and using a custom scheduler, see the unit test **`org.apache.camel.component.file.FileConsumerCustomSchedulerTest`** from the source code in **`camel-core`**.

Error Handling When Using `PollingConsumerPollStrategy`

`org.apache.camel.PollingConsumerPollStrategy` is a pluggable strategy that you can configure on the **`ScheduledPollConsumer`**. The default implementation **`org.apache.camel.impl.DefaultPollingConsumerPollStrategy`** will log the caused exception at **`WARN`** level and then ignore this issue.

The strategy interface provides the following three methods:

- **`begin`**
  - `void begin(Consumer consumer, Endpoint endpoint)`
- **`begin`** (**Camel 2.3**)
  - `boolean begin(Consumer consumer, Endpoint endpoint)`
- **`commit`**
  - `void commit(Consumer consumer, Endpoint endpoint)`
- **`commit`** (**Camel 2.6**)
  - `void commit(Consumer consumer, Endpoint endpoint, int polledMessages)`
- **`rollback`**
  - `boolean rollback(Consumer consumer, Endpoint endpoint, int retryCounter, Exception e) throws Exception`

In **Camel 2.3**: the begin method returns a **`boolean`** which indicates whether or not to skipping polling. So you can implement your custom logic and return **`false`** if you do not want to poll this time.

In **Camel 2.6**: the commit method has an additional parameter containing the number of message that was actually polled. For example if there was no messages polled, the value would be zero, and you can react accordingly.

The most interesting is the **`rollback`** as it allows you do handle the caused exception and decide what to do.

For instance if we want to provide a retry feature to a scheduled consumer we can implement the **`PollingConsumerPollStrategy`** method and put the retry logic in the **`rollback`** method. Lets just retry up till three times:

javapublic boolean rollback(Consumer consumer, Endpoint endpoint, int retryCounter, Exception e) throws Exception { if (retryCounter < 3) { // return true to tell Camel that it should retry the poll immediately return true; } // okay we give up do not retry anymore return false; }

Notice that we are given the **`Consumer`** as a parameter. We could use this to *restart* the consumer as we can invoke stop and start:

java// error occurred lets restart the consumer, that could maybe resolve the issue consumer.stop(); consumer.start();

Note: if you implement the **`begin`** operation make sure to avoid throwing exceptions as in such a case the **`poll`** operation is not invoked and Camel will invoke the **`rollback`** directly.

Configuring an Endpoint to Use `PollingConsumerPollStrategy`

To configure an Endpoint to use a custom **`PollingConsumerPollStrategy`** you use the option **`pollStrategy`**. For example in the file consumer below we want to use our custom strategy defined in the Registry with the bean id **`myPoll`**:

from("file://inbox/?pollStrategy=#myPoll") .to("activemq:queue:inbox")

Using This Pattern

See Also

- POJO Consuming
- Batch Consumer