Red Hat build of Apache Camel for Spring Boot 3.20

Camel Spring Boot Reference

Camel Spring Boot Reference

Last Updated: 2024-03-18

Red Hat build of Apache Camel for Spring Boot 3.20 Camel Spring Boot Reference

Camel Spring Boot Reference

Legal Notice

Abstract

This guide describes the settings for Camel Spring Boot components.

Table of Contents

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. AMQP

Since Camel 1.2

Both producer and consumer are supported

The AMQP component supports the AMQP 1.0 protocol using the JMS Client API of the Qpid project.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-amqp</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>
```

1.1. URI FORMAT

```
amqp:[queue:|topic:]destinationName[?options]
```

1.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

1.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

1.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

1.3. COMPONENT OPTIONS

The AMQP component supports 100 options, which are listed below.

Name	Description	Default	Type
clientId (common)	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String
connectionFactory (common)	The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory
disableReplyTo (common)	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	boolean
durableSubscriptionName (common)	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String
includeAmqpAnnotations (common)	Whether to include AMQP annotations when mapping from AMQP to Camel Message. Setting this to true maps AMQP message annotations that contain a JMS_AMQP_MA_ prefix to message headers. Due to limitations in Apache Qpid JMS API, currently delivery annotations are ignored.	false	boolean

Name	Description	Default	Type
jmsMessageType (common)	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it. Enum values: • Bytes • Map • Object • Stream • Text		JmsMessageType
replyTo (common)	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean
acknowledgementModeName (consumer)	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE. Enum values: • SESSION_TRANSACTED • CLIENT_ACKNOWLEDGE • AUTO_ACKNOWLEDGE • DUPS_OK_ACKNOWLEDGE	AUTO_ACKNOWLEDGE	String

Name	Description	Default	Type
artemisConsumerPriority (consumer)	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		int
asyncConsumer (consumer)	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean
cacheLevel (consumer)	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		int

Name	Description	Default	Type
cacheLevelName (consumer)	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION	CACHE_AUTO	String
concurrentConsumers (consumer)	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	int
maxConcurrentConsumers (consumer)	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		int
replyToDeliveryPersistent (consumer)	Specifies whether to use persistent delivery by default for replies.	true	boolean
selector (consumer)	Sets the JMS selector to use.		String

Name	Description	Default	Type
subscriptionDurable (consumer)	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	boolean
subscriptionName (consumer)	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String
subscriptionShared (consumer)	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	boolean
acceptMessagesWhileStopping (consumer (advanced))	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	boolean

Name	Description	Default	Type
allowReplyManagerQuickStop (consumer (advanced))	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	boolean
consumerType (consumer (advanced))	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType

Name	Description	Default	Type
defaultTaskExecutorType (consumer (advanced))	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers. Enum values: • ThreadPool • SimpleAsync		DefaultTaskExecutorType
eagerLoadingOfProperties (consumer (advanced))	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	boolean
eagerPoisonBody (consumer (advanced))	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
exposeListenerSession (consumer (advanced))	Specifies whether the listener session should be exposed when consuming messages.	false	boolean

Name	Description	Default	Type
replyToConsumerType (consumer (advanced))	The consumer type of the reply consumer (when doing request/reply), which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType
replyToSameDestinationAllowed (consumer (advanced))	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	boolean
taskExecutor (consumer (advanced))	Allows you to specify a custom task executor for consuming messages.		TaskExecutor
deliveryDelay (producer)	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	long
deliveryMode (producer)	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2. Enum values: • 1 • 2		Integer
deliveryPersistent (producer)	Specifies whether persistent delivery is used by default.	true	boolean

Name	Description	Default	Type
explicitQosEnabled (producer)	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
formatDateHeadersToIso8601 (producer)	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
preserveMessageQos (producer)	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	boolean

Name	Description	Default	Type
priority (producer)	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect. Enum values: • 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9	4	int
replyToConcurrentConsumers (producer)	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	int
replyToMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		int
replyToOnTimeoutMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	int
replyToOverride (producer)	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String

Name	Description	Default	Type
replyToType (producer)	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive. Enum values: • Temporary • Shared • Exclusive		ReplyToType
requestTimeout (producer)	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option.	20000	long
timeToLive (producer)	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	long
allowAdditionalHeaders (producer (advanced))	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
allowNullBody (producer (advanced))	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	boolean

Name	Description	Default	Type
alwaysCopyMessage (producer (advanced))	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	boolean
correlationProperty (producer (advanced))	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String
disableTimeToLive (producer (advanced))	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	boolean
forceSendOriginalMessage (producer (advanced))	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	boolean
includeSentJMSMessageID (producer (advanced))	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	boolean

Name	Description	Default	Type
replyToCacheLevelName (producer (advanced))	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION		String
replyToDestinationSelectorName (producer (advanced))	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String
streamMessageTypeEnabled (producer (advanced))	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	boolean
allowAutoWiredConnectionFactory (advanced)	Whether to auto-discover ConnectionFactory from the registry, if no connection factory has been configured. If only one instance of ConnectionFactory is found then it will be used. This is enabled by default.	true	boolean

Name	Description	Default	Type
allowAutoWiredDestinationResolver (advanced)	Whether to auto-discover DestinationResolver from the registry, if no destination resolver has been configured. If only one instance of DestinationResolver is found then it will be used. This is enabled by default.	true	boolean
allowSerializedHeaders (advanced)	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
artemisStreamingEnabled (advanced)	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	boolean
asyncStartListener (advanced)	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	boolean
asyncStopListener (advanced)	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
configuration (advanced)	To use a shared JMS configuration.		JmsConfiguration

Name	Description	Default	Type
destinationResolver (advanced)	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry).		DestinationResolver
errorHandler (advanced)	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler.		ErrorHandler
exceptionListener (advanced)	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions.		ExceptionListener
idleConsumerLimit (advanced)	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	int
idleTaskExecutionLimit (advanced)	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	int
includeAllJMSXProperties (advanced)	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	boolean

Name	Description	Default	Type
jmsKeyFormatStrategy (advanced)	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation. Enum values: • default • passthrough		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	boolean
maxMessagesPerTask (advanced)	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	int
messageConverter (advanced)	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message.		MessageConverter
messageCreatedStrategy (advanced)	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message.		MessageCreatedStrategy
messageIdEnabled (advanced)	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	boolean
messageListenerContainerFactory (advanced)	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom.		MessageListenerContainerFactory

Name	Description	Default	Type
messageTimestampEnabled (advanced)	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	boolean
pubSubNoLocal (advanced)	Specifies whether to inhibit the delivery of messages published by its own connection.	false	boolean
queueBrowseStrategy (advanced)	To use a custom QueueBrowseStrategy when browsing queues.		QueueBrowseStrategy
receiveTimeout (advanced)	The timeout for receiving messages (in milliseconds).	1000	long
recoveryInterval (advanced)	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds.	5000	long
requestTimeoutCheckerInterval (advanced)	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout.	1000	long
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
transferException (advanced)	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	boolean

Name	Description	Default	Type
transferExchange (advanced)	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	boolean
useMessageIDAsCorrelationID (advanced)	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	boolean
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleepingTime (advanced)	Interval in millis to sleep each time while waiting for provisional correlation id to be updated.	100	long
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy

Name	Description	Default	Type
errorHandlerLoggingLevel (logging)	Allows to configure the default errorHandler logging level for logging uncaught exceptions. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	boolean
password (security)	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
username (security)	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
transacted (transaction)	Specifies whether to use transacted mode.	false	boolean

Name	Description	Default	Type
transactedInOut (transaction)	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	boolean
lazyCreateTransactionManager (transaction (advanced))	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	boolean
transactionManager (transaction (advanced))	The Spring transaction manager to use.		PlatformTransactionManager
transactionName (transaction (advanced))	The name of the transaction to use.		String
transactionTimeout (transaction (advanced))	The timeout value of the transaction (in seconds), if using transacted mode.	-1	int

1.4. ENDPOINT OPTIONS

The AMQP endpoint is configured using URI syntax:

```
amqp:destinationType:destinationName
```

with the following path and query parameters:

1.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
destinationType (common)	The kind of destination to use. Enum values: • queue • topic • temp-queue • temp-topic	queue	String
destinationName (common)	**Required** Name of the queue or topic to use as destination.		String

1.4.2. Query Parameters (96 parameters)

Name	Description	Default	Type
clientId (common)	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String
connectionFactory (common)	The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory
disableReplyTo (common)	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	boolean
durableSubscriptionName (common)	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String

Name	Description	Default	Type
jmsMessageType (common)	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it. Enum values: • Bytes • Map • Object • Stream • Text		JmsMessageType
replyTo (common)	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean
acknowledgementModeName (consumer)	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE. Enum values: • SESSION_TRANSACTED • CLIENT_ACKNOWLEDGE • AUTO_ACKNOWLEDGE • DUPS_OK_ACKNOWLEDGE	AUTO_ACKNOWLEDGE	String

Name	Description	Default	Type
artemisConsumerPriority (consumer)	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		int
asyncConsumer (consumer)	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean
cacheLevel (consumer)	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		int

Name	Description	Default	Type
cacheLevelName (consumer)	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION	CACHE_AUTO	String
concurrentConsumers (consumer)	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	int
maxConcurrentConsumers (consumer)	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		int
replyToDeliveryPersistent (consumer)	Specifies whether to use persistent delivery by default for replies.	true	boolean
selector (consumer)	Sets the JMS selector to use.		String

Name	Description	Default	Type
subscriptionDurable (consumer)	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	boolean
subscriptionName (consumer)	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String
subscriptionShared (consumer)	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	boolean
acceptMessagesWhileStopping (consumer (advanced))	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	boolean

Name	Description	Default	Type
allowReplyManagerQuickStop (consumer (advanced))	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	boolean
consumerType (consumer (advanced))	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType

Name	Description	Default	Type
defaultTaskExecutorType (consumer (advanced))	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers. Enum values: • ThreadPool • SimpleAsync		DefaultTaskExecutorType
eagerLoadingOfProperties (consumer (advanced))	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	boolean
eagerPoisonBody (consumer (advanced))	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
exposeListenerSession (consumer (advanced))	Specifies whether the listener session should be exposed when consuming messages.	false	boolean
replyToConsumerType (consumer (advanced))	The consumer type of the reply consumer (when doing request/reply), which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType
replyToSameDestinationAllowed (consumer (advanced))	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	boolean
taskExecutor (consumer (advanced))	Allows you to specify a custom task executor for consuming messages.		TaskExecutor
deliveryDelay (producer)	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	long

Name	Description	Default	Type
deliveryMode (producer)	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2. Enum values: • 1 • 2		Integer
deliveryPersistent (producer)	Specifies whether persistent delivery is used by default.	true	boolean
explicitQosEnabled (producer)	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
formatDateHeadersToIso8601 (producer)	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	boolean
preserveMessageQos (producer)	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	boolean

Name	Description	Default	Type
priority (producer)	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect. Enum values: • 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9	4	int
replyToConcurrentConsumers (producer)	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	int
replyToMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		int
replyToOnTimeoutMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	int
replyToOverride (producer)	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String

Name	Description	Default	Type
replyToType (producer)	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive. Enum values: • Temporary • Shared • Exclusive		ReplyToType
requestTimeout (producer)	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option.	20000	long
timeToLive (producer)	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	long
allowAdditionalHeaders (producer (advanced))	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
allowNullBody (producer (advanced))	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	boolean

Name	Description	Default	Type
alwaysCopyMessage (producer (advanced))	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	boolean
correlationProperty (producer (advanced))	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String
disableTimeToLive (producer (advanced))	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	boolean
forceSendOriginalMessage (producer (advanced))	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	boolean
includeSentJMSMessageID (producer (advanced))	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
replyToCacheLevelName (producer (advanced))	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION		String
replyToDestinationSelectorName (producer (advanced))	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String

Name	Description	Default	Type
streamMessageTypeEnabled (producer (advanced))	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	boolean
allowSerializedHeaders (advanced)	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
artemisStreamingEnabled (advanced)	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	boolean
asyncStartListener (advanced)	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	boolean
asyncStopListener (advanced)	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	boolean
destinationResolver (advanced)	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry).		DestinationResolver

Name	Description	Default	Type
errorHandler (advanced)	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler.		ErrorHandler
exceptionListener (advanced)	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions.		ExceptionListener
headerFilterStrategy (advanced)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
idleConsumerLimit (advanced)	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	int
idleTaskExecutionLimit (advanced)	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	int
includeAllJMSXProperties (advanced)	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	boolean

Name	Description	Default	Type
jmsKeyFormatStrategy (advanced)	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation. Enum values: • default • passthrough		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	boolean
maxMessagesPerTask (advanced)	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	int
messageConverter (advanced)	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message.		MessageConverter
messageCreatedStrategy (advanced)	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message.		MessageCreatedStrategy
messageIdEnabled (advanced)	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	boolean

Name	Description	Default	Type
messageListenerContainerFactory (advanced)	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom.		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	boolean
pubSubNoLocal (advanced)	Specifies whether to inhibit the delivery of messages published by its own connection.	false	boolean
receiveTimeout (advanced)	The timeout for receiving messages (in milliseconds).	1000	long
recoveryInterval (advanced)	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds.	5000	long
requestTimeoutCheckerInterval (advanced)	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout.	1000	long
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean

Name	Description	Default	Type
transferException (advanced)	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	boolean
transferExchange (advanced)	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	boolean
useMessageIDAsCorrelationID (advanced)	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	boolean
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleepingTime (advanced)	Interval in millis to sleep each time while waiting for provisional correlation id to be updated.	100	long

Name	Description	Default	Type
errorHandlerLoggingLevel (logging)	Allows to configure the default errorHandler logging level for logging uncaught exceptions. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	boolean
password (security)	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
username (security)	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
transacted (transaction)	Specifies whether to use transacted mode.	false	boolean

Name	Description	Default	Type
transactedInOut (transaction)	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	boolean
lazyCreateTransactionManager (transaction (advanced))	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	boolean
transactionManager (transaction (advanced))	The Spring transaction manager to use.		PlatformTransactionManager
transactionName (transaction (advanced))	The name of the transaction to use.		String
transactionTimeout (transaction (advanced))	The timeout value of the transaction (in seconds), if using transacted mode.	-1	int

1.5. USAGE

As AMQP component is inherited from JMS component, the usage of the former is almost identical to the latter:

Using AMQP component

```
// Consuming from AMQP queue
from("amqp:queue:incoming").
  to(...);

// Sending message to the AMQP topic
from(...).
  to("amqp:topic:notify");
```

1.6. CONFIGURING AMQP COMPONENT

Creating AMQP 1.0 component

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672");

AMQPComponent authorizedAmqp = AMQPComponent.amqpComponent("amqp://localhost:5672",
"user", "password");
```

You can also add an instance of **org.apache.camel.component.amqp.AMQPConnectionDetails** to the registry in order to automatically configure the AMQP component. For example for Spring Boot you just have to define bean:

AMQP connection details auto-configuration

```
@Bean
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}

@Bean
AMQPConnectionDetails securedAmqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672", "username", "password");
}
```

Likewise, you can also use CDI producer methods when using Camel-CDI

AMQP connection details auto-configuration for CDI

```
@Produces
AMQPConnectionDetails amqpConnection() {
  return new AMQPConnectionDetails("amqp://localhost:5672");
}
```

You can also rely on the to read the AMQP connection details. Factory method **AMQPConnectionDetails.discoverAMQP()** attempts to read Camel properties in a Kubernetes-like convention, just as demonstrated on the snippet below:

AMQP connection details auto-configuration

```
export AMQP_SERVICE_HOST = "mybroker.com"
export AMQP_SERVICE_PORT = "6666"
export AMQP_SERVICE_USERNAME = "username"
export AMQP_SERVICE_PASSWORD = "password"
```

```
...

@Bean
AMQPConnectionDetails amqpConnection() {
  return AMQPConnectionDetails.discoverAMQP();
}
```

Enabling AMQP specific options

If you, for example, need to enable **amqp.traceFrames** you can do that by appending the option to your URI, like the following example:

```
AMQPComponent amqp = AMQPComponent.amqpComponent("amqp://localhost:5672?
amqp.traceFrames=true");
```

For reference refer QPID JMS client configuration .

1.7. USING TOPICS

To have using topics working with **camel-amqp** you need to configure the component to use **topic:**// as topic prefix, as shown below:

```
<bean id="amqp" class="org.apache.camel.component.amqp.AmqpComponent">
  <property name="connectionFactory">
    <bean class="org.apache.qpid.jms.JmsConnectionFactory" factory-method="createFromURL">
      <property name="remoteURI" value="amqp://localhost:5672" />
      <property name="topicPrefix" value="topic://" />  <!-- only necessary when connecting to
ActiveMQ over AMQP 1.0 -->
    </bean>
  </property>
</bean>
```

Keep in mind that both **AMQPComponent#amqpComponent()** methods and **AMQPConnectionDetails** pre-configure the component with the topic prefix, so you don't have to configure it explicitly.

1.8. SPRING BOOT AUTO-CONFIGURATION

When using amqp with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-amqp-starter</artifactId>
</dependency>
```

The component supports 101 options, which are listed below.

Name	Description	Default	Type
camel.component.amqp.accept-messages-while-stopping	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	Boolean
camel.component.amqp.acknowledgement-mode-name	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE.	AUTO_ACKNOWLEDGE	String
camel.component.amqp.allow-additional-headers	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
camel.component.amqp.allow-auto-wired-connection-factory	Whether to auto-discover ConnectionFactory from the registry, if no connection factory has been configured. If only one instance of ConnectionFactory is found then it will be used. This is enabled by default.	true	Boolean
camel.component.amqp.allow-auto-wired-destination-resolver	Whether to auto-discover DestinationResolver from the registry, if no destination resolver has been configured. If only one instance of DestinationResolver is found then it will be used. This is enabled by default.	true	Boolean
camel.component.amqp.allow-null-body	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	Boolean

Name	Description	Default	Type
camel.component.amqp.allow-reply-manager-quick-stop	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	Boolean
camel.component.amqp.allow-serialized-headers	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	Boolean
camel.component.amqp.always-copy-message	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	Boolean
camel.component.amqp.artemis-consumer-priority	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		Integer
camel.component.amqp.artemis-streaming-enabled	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.async-consumer	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	Boolean
camel.component.amqp.async-start-listener	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	Boolean
camel.component.amqp.async-stop-listener	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	Boolean
camel.component.amqp.auto-startup	Specifies whether the consumer container should auto-startup.	true	Boolean
camel.component.amqp.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.amqp.cache-level	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		Integer

Name	Description	Default	Type
camel.component.amqp.cache-level-name	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information.	CACHE_AUTO	String
camel.component.amqp.client-id	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String
camel.component.amqp.concurrent-consumers	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	Integer
camel.component.amqp.configuration	To use a shared JMS configuration. The option is a org.apache.camel.component.jms.JmsConfiguration type.		JmsConfiguration
camel.component.amqp.connection-factory	The connection factory to be use. A connection factory must be configured either on the component or endpoint. The option is a javax.jms.ConnectionFactory type.		ConnectionFactory
camel.component.amqp.consumer-type	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use.		ConsumerType

Name	Description	Default	Type
camel.component.amqp.correlation-property	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String
camel.component.amqp.default-task-executor-type	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers.		DefaultTaskExecutorType
camel.component.amqp.delivery-delay	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	Long
camel.component.amqp.delivery-mode	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2.		Integer
camel.component.amqp.delivery-persistent	Specifies whether persistent delivery is used by default.	true	Boolean
camel.component.amqp.destination-resolver	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry). The option is a org.springframework.jms.support.destination.DestinationResolver type.		DestinationResolver

Name	Description	Default	Type
camel.component.amqp.disable-reply-to	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	Boolean
camel.component.amqp.disable-time-to-live	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	Boolean
camel.component.amqp.durable-subscription-name	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String
camel.component.amqp.eager-loading-of-properties	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	Boolean
camel.component.amqp.eager-poison-body	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
camel.component.amqp.enabled	Whether to enable auto configuration of the amqp component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.amqp.error-handler	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler. The option is a org.springframework.util.ErrorHandler type.		ErrorHandler
camel.component.amqp.error-handler-log-stack-trace	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	Boolean
camel.component.amqp.error-handler-logging-level	Allows to configure the default errorHandler logging level for logging uncaught exceptions.		LoggingLevel
camel.component.amqp.exception-listener	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions. The option is a javax.jms.ExceptionListener type.		ExceptionListener
camel.component.amqp.explicit-qos-enabled	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
camel.component.amqp.expose-listener-session	Specifies whether the listener session should be exposed when consuming messages.	false	Boolean
camel.component.amqp.force-send-original-message	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.format-date-headers-to-iso8601	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	Boolean
camel.component.amqp.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.amqp.idle-consumer-limit	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	Integer
camel.component.amqp.idle-task-execution-limit	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	Integer
camel.component.amqp.include-all-jmsx-properties	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	Boolean
camel.component.amqp.include-amqp-annotations	Whether to include AMQP annotations when mapping from AMQP to Camel Message. Setting this to true maps AMQP message annotations that contain a JMS_AMQP_MA_ prefix to message headers. Due to limitations in Apache Qpid JMS API, currently delivery annotations are ignored.	false	Boolean
camel.component.amqp.include-sent-jms-message-id	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.jms-key-format-strategy	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation.		JmsKeyFormatStrategy
camel.component.amqp.jms-message-type	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it.		JmsMessageType
camel.component.amqp.lazy-create-transaction-manager	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	Boolean
camel.component.amqp.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.amqp.map-jms-message	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	Boolean
camel.component.amqp.max-concurrent-consumers	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		Integer

Name	Description	Default	Type
camel.component.amqp.max-messages-per-task	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	Integer
camel.component.amqp.message-converter	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message. The option is a org.springframework.jms.support.converter.MessageConverter type.		MessageConverter
camel.component.amqp.message-created-strategy	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message. The option is a org.apache.camel.component.jms.MessageCreatedStrategy type.		MessageCreatedStrategy
camel.component.amqp.message-id-enabled	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	Boolean
camel.component.amqp.message-listener-container-factory	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom. The option is a org.apache.camel.component.jms.MessageListenerContainerFactory type.		MessageListenerContainerFactory
camel.component.amqp.message-timestamp-enabled	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	Boolean
camel.component.amqp.password	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String

Name	Description	Default	Type
camel.component.amqp.preserve-message-qos	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	Boolean
camel.component.amqp.priority	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect.	4	Integer
camel.component.amqp.pub-sub-no-local	Specifies whether to inhibit the delivery of messages published by its own connection.	false	Boolean
camel.component.amqp.queue-browse-strategy	To use a custom QueueBrowseStrategy when browsing queues. The option is a org.apache.camel.component.jms.QueueBrowseStrategy type.		QueueBrowseStrategy
camel.component.amqp.receive-timeout	The timeout for receiving messages (in milliseconds). The option is a long type.	1000	Long
camel.component.amqp.recovery-interval	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds. The option is a long type.	5000	Long
camel.component.amqp.reply-to	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String

Name	Description	Default	Type
camel.component.amqp.reply-to-cache-level-name	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION.		String
camel.component.amqp.reply-to-concurrent-consumers	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	Integer
camel.component.amqp.reply-to-consumer-type	The consumer type of the reply consumer (when doing request/reply), which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use.		ConsumerType
camel.component.amqp.reply-to-delivery-persistent	Specifies whether to use persistent delivery by default for replies.	true	Boolean
camel.component.amqp.reply-to-destination-selector-name	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String
camel.component.amqp.reply-to-max-concurrent-consumers	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		Integer

Name	Description	Default	Type
camel.component.amqp.reply-to-on-timeout-max-concurrent-consumers	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	Integer
camel.component.amqp.reply-to-override	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String
camel.component.amqp.reply-to-same-destination-allowed	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	Boolean
camel.component.amqp.reply-to-type	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive.		ReplyToType
camel.component.amqp.request-timeout	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option. The option is a long type.	20000	Long
camel.component.amqp.request-timeout-checker-interval	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout. The option is a long type.	1000	Long

Name	Description	Default	Type
camel.component.amqp.selector	Sets the JMS selector to use.		String
camel.component.amqp.stream-message-type-enabled	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	Boolean
camel.component.amqp.subscription-durable	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	Boolean
camel.component.amqp.subscription-name	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String
camel.component.amqp.subscription-shared	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.synchronous	Sets whether synchronous processing should be strictly used.	false	Boolean
camel.component.amqp.task-executor	Allows you to specify a custom task executor for consuming messages. The option is a org.springframework.core.task.TaskExecutor type.		TaskExecutor
camel.component.amqp.test-connection-on-startup	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	Boolean
camel.component.amqp.time-to-live	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	Long
camel.component.amqp.transacted	Specifies whether to use transacted mode.	false	Boolean
camel.component.amqp.transacted-in-out	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.transaction-manager	The Spring transaction manager to use. The option is a org.springframework.transaction.PlatformTransactionManager type.		PlatformTransactionManager
camel.component.amqp.transaction-name	The name of the transaction to use.		String
camel.component.amqp.transaction-timeout	The timeout value of the transaction (in seconds), if using transacted mode.	-1	Integer
camel.component.amqp.transfer-exception	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	Boolean
camel.component.amqp.transfer-exchange	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	Boolean

Name	Description	Default	Type
camel.component.amqp.use-message-id-as-correlation-id	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	Boolean
camel.component.amqp.username	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
camel.component.amqp.wait-for-provision-correlation-to-be-updated-counter	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	Integer
camel.component.amqp.wait-for-provision-correlation-to-be-updated-thread-sleeping-time	Interval in millis to sleep each time while waiting for provisional correlation id to be updated. The option is a long type.	100	Long

CHAPTER 2. AWS CLOUDWATCH

Only producer is supported

The AWS2 Cloudwatch component allows messages to be sent to an Amazon CloudWatch metrics. The implementation of the Amazon API is provided by the AWS SDK.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon CloudWatch. More information is available at Amazon CloudWatch.

2.1. URI FORMAT

```
aws2-cw://namespace[?options]
```

The metrics will be created if they don't already exists. You can append query options to the URI in the following format, **?options=value&option2=value&...**

2.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

2.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

2.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

2.3. COMPONENT OPTIONS

The AWS CloudWatch component supports 18 options, which are listed below.

Name	Description	Default	Type
amazonCwClient (producer)	**Autowired** To use the AmazonCloudWatch as the client.		CloudWatchClient
configuration (producer)	The component configuration.		Cw2Configuration
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
name (producer)	The metric name.		String
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (producer)	To define a proxy host when instantiating the CW client.		String
proxyPort (producer)	To define a proxy port when instantiating the CW client.		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the CW client. Enum values: • HTTP • HTTPS	HTTPS	Protocol

Name	Description	Default	Type
region (producer)	The region in which CW client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
timestamp (producer)	The metric timestamp.		Instant
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
unit (producer)	The metric unit.		String
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
value (producer)	The metric value.		Double
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

2.4. ENDPOINT OPTIONS

The AWS CloudWatch endpoint is configured using URI syntax:

```
aws2-cw:namespace
```

with the following path and query parameters:

2.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
namespace (producer)	**Required** The metric namespace.		String

2.4.2. Query Parameters (16 parameters)

Name	Description	Default	Type
amazonCwClient (producer)	**Autowired** To use the AmazonCloudWatch as the client.		CloudWatchClient
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
name (producer)	The metric name.		String
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (producer)	To define a proxy host when instantiating the CW client.		String
proxyPort (producer)	To define a proxy port when instantiating the CW client.		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the CW client. Enum values: • HTTP • HTTPS	HTTPS	Protocol

Name	Description	Default	Type
region (producer)	The region in which CW client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
timestamp (producer)	The metric timestamp.		Instant
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
unit (producer)	The metric unit.		String
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
value (producer)	The metric value.		Double
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required CW component options

You have to provide the amazonCwClient in the Registry or your accessKey and secretKey to access the Amazon's CloudWatch.

2.5. USAGE

2.5.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey
- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Web Identity Token from AWS STS.
- The shared credentials and config files.

- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

2.5.2. Message headers evaluated by the CW producer

Header	Type	Description
CamelAwsCwMetricName	**String**	The Amazon CW metric name.
CamelAwsCwMetricValue	**Double**	The Amazon CW metric value.
CamelAwsCwMetricUnit	**String**	The Amazon CW metric unit.
CamelAwsCwMetricNamespace	**String**	The Amazon CW metric namespace.
CamelAwsCwMetricTimestamp	**Date**	The Amazon CW metric timestamp.
CamelAwsCwMetricDimensionName	**String**	The Amazon CW metric dimension name.
CamelAwsCwMetricDimensionValue	**String**	The Amazon CW metric dimension value.
CamelAwsCwMetricDimensions	**Map<String, String>**	A map of dimension names and dimension values.

2.5.3. Advanced CloudWatchClient configuration

If you need more control over the **CloudWatchClient** instance configuration you can create your own instance and refer to it from the URI:

```
from("direct:start")
.to("aws2-cw://namespace?amazonCwClient=#client");
```

The **#client** refers to a **CloudWatchClient** in the Registry.

2.6. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
  <groupId>org.apache.camel</groupId>
```

```
    <artifactId>camel-aws2-cw</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **{camel-version}** must be replaced by the actual version of Camel.

2.7. EXAMPLES

2.7.1. Producer Example

```
from("direct:start")
  .to("aws2-cw://http://camel.apache.org/aws-cw");
```

and sends something like

```
exchange.getIn().setHeader(Cw2Constants.METRIC_NAME, "ExchangesCompleted");
exchange.getIn().setHeader(Cw2Constants.METRIC_VALUE, "2.0");
exchange.getIn().setHeader(Cw2Constants.METRIC_UNIT, "Count");
```

2.8. SPRING BOOT AUTO-CONFIGURATION

When using aws2-cw with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-cw-starter</artifactId>
</dependency>
```

The component supports 19 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-cw.access-key	Amazon AWS Access Key.		String
camel.component.aws2-cw.amazon-cw-client	To use the AmazonCloudWatch as the client. The option is a software.amazon.awssdk.services.cloudwatch.CloudWatchClient type.		CloudWatchClient
camel.component.aws2-cw.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.aws2-cw.configuration	The component configuration. The option is a org.apache.camel.component.aws2.cw.Cw2Configuration type.		Cw2Configuration
camel.component.aws2-cw.enabled	Whether to enable auto configuration of the aws2-cw component. This is enabled by default.		Boolean
camel.component.aws2-cw.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-cw.name	The metric name.		String
camel.component.aws2-cw.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-cw.proxy-host	To define a proxy host when instantiating the CW client.		String
camel.component.aws2-cw.proxy-port	To define a proxy port when instantiating the CW client.		Integer
camel.component.aws2-cw.proxy-protocol	To define a proxy protocol when instantiating the CW client.		Protocol
camel.component.aws2-cw.region	The region in which CW client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-cw.secret-key	Amazon AWS Secret Key.		String

Name	Description	Default	Type
camel.component.aws2-cw.timestamp	The metric timestamp. The option is a java.time.Instant type.		Instant
camel.component.aws2-cw.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-cw.unit	The metric unit.		String
camel.component.aws2-cw.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-cw.use-default-credentials-provider	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean
camel.component.aws2-cw.value	The metric value.		Double

CHAPTER 3. AWS DYNAMODB

Only producer is supported

The AWS2 DynamoDB component supports storing and retrieving data from/to service.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon DynamoDB. More information is available at Amazon DynamoDB.

3.1. URI FORMAT

```
aws2-ddb://domainName[?options]
```

You can append query options to the URI in the following format, **?options=value&option2=value&…**

3.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

3.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

3.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

3.3. COMPONENT OPTIONS

The AWS DynamoDB component supports 22 options, which are listed below.

Name	Description	Default	Type
amazonDDBClient (producer)	**Autowired** To use the AmazonDynamoDB as the client.		DynamoDbClient
configuration (producer)	The component configuration.		Ddb2Configuration
consistentRead (producer)	Determines whether or not strong consistency should be enforced when data is read.	false	boolean
enabledInitialDescribeTable (producer)	Set whether the initial Describe table operation in the DDB Endpoint must be done, or not.	true	boolean
keyAttributeName (producer)	Attribute name when creating table.		String
keyAttributeType (producer)	Attribute type when creating table.		String
keyScalarType (producer)	The key scalar type, it can be S (String), N (Number) and B (Bytes).		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	What operation to perform. Enum values: • BatchGetItems • DeleteItem • DeleteTable • DescribeTable • GetItem • PutItem • Query • Scan • UpdateItem • UpdateTable	PutItem	Ddb2Operations
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (producer)	To define a proxy host when instantiating the DDB client.		String
proxyPort (producer)	The region in which DynamoDB client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the DDB client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
readCapacity (producer)	The provisioned throughput to reserve for reading resources from your table.		Long
region (producer)	The region in which DDB client needs to work.		String

Name	Description	Default	Type
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
writeCapacity (producer)	The provisioned throughput to reserved for writing resources to your table.		Long
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

3.4. ENDPOINT OPTIONS

The AWS DynamoDB endpoint is configured using URI syntax:

```
aws2-ddb:tableName
```

with the following path and query parameters:

3.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
tableName (producer)	**Required** The name of the table currently worked with.		String

3.4.2. Query Parameters (20 parameters)

Name	Description	Default	Type
amazonDDBClient (producer)	**Autowired** To use the AmazonDynamoDB as the client.		DynamoDbClient
consistentRead (producer)	Determines whether or not strong consistency should be enforced when data is read.	false	boolean
enabledInitialDescribeTable (producer)	Set whether the initial Describe table operation in the DDB Endpoint must be done, or not.	true	boolean
keyAttributeName (producer)	Attribute name when creating table.		String
keyAttributeType (producer)	Attribute type when creating table.		String
keyScalarType (producer)	The key scalar type, it can be S (String), N (Number) and B (Bytes).		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	What operation to perform. Enum values: • BatchGetItems • DeleteItem • DeleteTable • DescribeTable • GetItem • PutItem • Query • Scan • UpdateItem • UpdateTable	PutItem	Ddb2Operations
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (producer)	To define a proxy host when instantiating the DDB client.		String
proxyPort (producer)	The region in which DynamoDB client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the DDB client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
readCapacity (producer)	The provisioned throughput to reserve for reading resources from your table.		Long
region (producer)	The region in which DDB client needs to work.		String

Name	Description	Default	Type
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
writeCapacity (producer)	The provisioned throughput to reserved for writing resources to your table.		Long
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required DDB component options

You have to provide the amazonDDBClient in the Registry or your accessKey and secretKey to access the Amazon's DynamoDB.

3.5. USAGE

3.5.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey
- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Web Identity Token from AWS STS.
- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

3.5.2. Message headers evaluated by the DDB producer

Header	Type	Description
CamelAwsDdbBatchItems	**Map<String, KeysAndAttributes>**	A map of the table name and corresponding items to get by primary key.
CamelAwsDdbTableName	**String**	Table Name for this operation.
CamelAwsDdbKey	**Key**	The primary key that uniquely identifies each item in a table.
CamelAwsDdbReturnValues	**String**	Use this parameter if you want to get the attribute name-value pairs before or after they are modified(NONE, ALL_OLD, UPDATED_OLD, ALL_NEW, UPDATED_NEW).
CamelAwsDdbUpdateCondition	**Map<String, ExpectedAttributeValue>**	Designates an attribute for a conditional modification.
CamelAwsDdbAttributeNames	**Collection<String>**	If attribute names are not specified then all attributes will be returned.
CamelAwsDdbConsistentRead	**Boolean**	If set to true, then a consistent read is issued, otherwise eventually consistent is used.
CamelAwsDdbIndexName	**String**	If set will be used as Secondary Index for Query operation.
CamelAwsDdbItem	**Map<String, AttributeValue>**	A map of the attributes for the item, and must include the primary key values that define the item.
CamelAwsDdbExactCount	**Boolean**	If set to true, Amazon DynamoDB returns a total number of items that match the query parameters, instead of a list of the matching items and their attributes.
CamelAwsDdbKeyConditions	**Map<String, Condition>**	This header specify the selection criteria for the query, and merge together the two old headers **CamelAwsDdbHashKeyValue** and **CamelAwsDdbScanRangeKeyCondition**
CamelAwsDdbStartKey	**Key**	Primary key of the item from which to continue an earlier query.
CamelAwsDdbHashKeyValue	**AttributeValue**	Value of the hash component of the composite primary key.
CamelAwsDdbLimit	**Integer**	The maximum number of items to return.

Header	Type	Description
CamelAwsDdbScanRangeKeyCondition	**Condition**	A container for the attribute values and comparison operators to use for the query.
CamelAwsDdbScanIndexForward	**Boolean**	Specifies forward or backward traversal of the index.
CamelAwsDdbScanFilter	**Map<String, Condition>**	Evaluates the scan results and returns only the desired values.
CamelAwsDdbUpdateValues	**Map<String, AttributeValueUpdate>**	Map of attribute name to the new value and action for the update.

3.5.3. Message headers set during BatchGetItems operation

Header	Type	Description
CamelAwsDdbBatchResponse	**Map<String,BatchResponse>**	Table names and the respective item attributes from the tables.
CamelAwsDdbUnprocessedKeys	**Map<String,KeysAndAttributes>**	Contains a map of tables and their respective keys that were not processed with the current response.

3.5.4. Message headers set during DeleteItem operation

Header	Type	Description
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	The list of attributes returned by the operation.

3.5.5. Message headers set during DeleteTable operation

Header	Type	Description
CamelAwsDdbProvisionedThroughput		
ProvisionedThroughputDescription		The value of the ProvisionedThroughput property for this table
CamelAwsDdbCreationDate	**Date**	Creation DateTime of this table.

Header	Type	Description
CamelAwsDdbTableItemCount	**Long**	Item count for this table.
CamelAwsDdbKeySchema	**KeySchema**	The KeySchema that identifies the primary key for this table. **From Camel 2.16.0 the type of this header is List<KeySchemaElement> and not KeySchema**
CamelAwsDdbTableName	**String**	The table name.
CamelAwsDdbTableSize	**Long**	The table size in bytes.
CamelAwsDdbTableStatus	**String**	The status of the table: CREATING, UPDATING, DELETING, ACTIVE

3.5.6. Message headers set during DescribeTable operation

Header	Type	Description
CamelAwsDdbProvisionedThroughput	\ {{ProvisionedThroughputDescription}}	The value of the ProvisionedThroughput property for this table
CamelAwsDdbCreationDate	**Date**	Creation DateTime of this table.
CamelAwsDdbTableItemCount	**Long**	Item count for this table.
CamelAwsDdbKeySchema	\{{KeySchema}}	The KeySchema that identifies the primary key for this table.
CamelAwsDdbTableName	**String**	The table name.
CamelAwsDdbTableSize	**Long**	The table size in bytes.
CamelAwsDdbTableStatus	**String**	The status of the table: CREATING, UPDATING, DELETING, ACTIVE
CamelAwsDdbReadCapacity	**Long**	ReadCapacityUnits property of this table.
CamelAwsDdbWriteCapacity	**Long**	WriteCapacityUnits property of this table.

3.5.7. Message headers set during GetItem operation

Header	Type	Description
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	The list of attributes returned by the operation.

3.5.8. Message headers set during PutItem operation

Header	Type	Description
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	The list of attributes returned by the operation.

3.5.9. Message headers set during Query operation

Header	Type	Description
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	The list of attributes returned by the operation.
CamelAwsDdbLastEvaluatedKey	**Key**	Primary key of the item where the query operation stopped, inclusive of the previous result set.
CamelAwsDdbConsumedCapacity	**Double**	The number of Capacity Units of the provisioned throughput of the table consumed during the operation.
CamelAwsDdbCount	**Integer**	Number of items in the response.

3.5.10. Message headers set during Scan operation

Header	Type	Description
CamelAwsDdbItems	**List<java.util.Map<String,AttributeValue>>**	The list of attributes returned by the operation.
CamelAwsDdbLastEvaluatedKey	**Key**	Primary key of the item where the query operation stopped, inclusive of the previous result set.
CamelAwsDdbConsumedCapacity	**Double**	The number of Capacity Units of the provisioned throughput of the table consumed during the operation.
CamelAwsDdbCount	**Integer**	Number of items in the response.

Header	Type	Description
CamelAwsDdbScannedCount	**Integer**	Number of items in the complete scan before any filters are applied.

3.5.11. Message headers set during UpdateItem operation

Header	Type	Description
CamelAwsDdbAttributes	**Map<String, AttributeValue>**	The list of attributes returned by the operation.

3.5.12. Advanced AmazonDynamoDB configuration

If you need more control over the **AmazonDynamoDB** instance configuration you can create your own instance and refer to it from the URI:

```
from("direct:start")
.to("aws2-ddb://domainName?amazonDDBClient=#client");
```

The **#client** refers to a **DynamoDbClient** in the Registry.

3.6. SUPPORTED PRODUCER OPERATIONS

- BatchGetItems
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem
- UpdateTable

3.7. EXAMPLES

3.7.1. Producer Examples

- PutItem: this operation will create an entry into DynamoDB

-

```
from("direct:start")
  .setHeader(Ddb2Constants.OPERATION, Ddb2Operations.PutItem)
  .setHeader(Ddb2Constants.CONSISTENT_READ, "true")
  .setHeader(Ddb2Constants.RETURN_VALUES, "ALL_OLD")
  .setHeader(Ddb2Constants.ITEM, attributeMap)
  .setHeader(Ddb2Constants.ATTRIBUTE_NAMES, attributeMap.keySet());
  .to("aws2-ddb://" + tableName + "?keyAttributeName=" + attributeName + "&keyAttributeType=" +
KeyType.HASH
  + "&keyScalarType=" + ScalarAttributeType.S
  + "&readCapacity=1&writeCapacity=1");
```

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-ddb</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

3.8. SPRING BOOT AUTO-CONFIGURATION

When using aws2-ddb with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-ddb-starter</artifactId>
</dependency>
```

The component supports 40 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-ddb.access-key	Amazon AWS Access Key.		String
camel.component.aws2-ddb.amazon-d-d-b-client	To use the AmazonDynamoDB as the client. The option is a software.amazon.awssdk.services.dynamodb.DynamoDbClient type.		DynamoDbClient

Name	Description	Default	Type
camel.component.aws2-ddb.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-ddb.configuration	The component configuration. The option is a org.apache.camel.component.aws2.ddb.Ddb2Configuration type.		Ddb2Configuration
camel.component.aws2-ddb.consistent-read	Determines whether or not strong consistency should be enforced when data is read.	false	Boolean
camel.component.aws2-ddb.enabled	Whether to enable auto configuration of the aws2-ddb component. This is enabled by default.		Boolean
camel.component.aws2-ddb.enabled-initial-describe-table	Set whether the initial Describe table operation in the DDB Endpoint must be done, or not.	true	Boolean
camel.component.aws2-ddb.key-attribute-name	Attribute name when creating table.		String
camel.component.aws2-ddb.key-attribute-type	Attribute type when creating table.		String
camel.component.aws2-ddb.key-scalar-type	The key scalar type, it can be S (String), N (Number) and B (Bytes).		String

Name	Description	Default	Type
camel.component.aws2-ddb.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-ddb.operation	What operation to perform.		Ddb2Operations
camel.component.aws2-ddb.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-ddb.proxy-host	To define a proxy host when instantiating the DDB client.		String
camel.component.aws2-ddb.proxy-port	The region in which DynamoDB client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		Integer
camel.component.aws2-ddb.proxy-protocol	To define a proxy protocol when instantiating the DDB client.		Protocol
camel.component.aws2-ddb.read-capacity	The provisioned throughput to reserve for reading resources from your table.		Long
camel.component.aws2-ddb.region	The region in which DDB client needs to work.		String
camel.component.aws2-ddb.secret-key	Amazon AWS Secret Key.		String

Name	Description	Default	Type
camel.component.aws2-ddb.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-ddb.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-ddb.use-default-credentials-provider	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean
camel.component.aws2-ddb.write-capacity	The provisioned throughput to reserved for writing resources to your table.		Long
camel.component.aws2-ddbstream.access-key	Amazon AWS Access Key.		String
camel.component.aws2-ddbstream.amazon-dynamo-db-streams-client	Amazon DynamoDB client to use for all requests for this endpoint. The option is a software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient type.		DynamoDbStreamsClient
camel.component.aws2-ddbstream.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-ddbstream.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.aws2-ddbstream.configuration	The component configuration. The option is a org.apache.camel.component.aws2.ddbstream.Ddb2StreamConfiguration type.		Ddb2StreamConfiguration
camel.component.aws2-ddbstream.enabled	Whether to enable auto configuration of the aws2-ddbstream component. This is enabled by default.		Boolean
camel.component.aws2-ddbstream.max-results-per-request	Maximum number of records that will be fetched in each poll.		Integer
camel.component.aws2-ddbstream.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-ddbstream.proxy-host	To define a proxy host when instantiating the DDBStreams client.		String
camel.component.aws2-ddbstream.proxy-port	To define a proxy port when instantiating the DDBStreams client.		Integer
camel.component.aws2-ddbstream.proxy-protocol	To define a proxy protocol when instantiating the DDBStreams client.		Protocol
camel.component.aws2-ddbstream.region	The region in which DDBStreams client needs to work.		String
camel.component.aws2-ddbstream.secret-key	Amazon AWS Secret Key.		String

Name	Description	Default	Type
camel.component.aws2-ddbstream.stream-iterator-type	Defines where in the DynamoDB stream to start getting records. Note that using FROM_START can cause a significant delay before the stream has caught up to real-time.		Ddb2StreamConfiguration$StreamIteratorType
camel.component.aws2-ddbstream.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-ddbstream.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-ddbstream.use-default-credentials-provider	Set whether the DynamoDB Streams client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean

CHAPTER 4. AWS KINESIS

Both producer and consumer are supported

The AWS2 Kinesis component supports receiving messages from and sending messages to Amazon Kinesis (no Batch supported) service.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon Kinesis. More information are available at AWS Kinesis.

4.1. URI FORMAT

```
aws2-kinesis://stream-name[?options]
```

The stream needs to be created prior to it being used. You can append query options to the URI in the following format, **?options=value&option2=value&...**

4.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

4.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

4.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

4.3. COMPONENT OPTIONS

The AWS Kinesis component supports 22 options, which are listed below.

Name	Description	Default	Type
amazonKinesisClient (common)	**Autowired** Amazon Kinesis client to use for all requests for this endpoint.		KinesisClient
cborEnabled (common)	This option will set the CBOR_ENABLED property during the execution.	true	boolean
configuration (common)	Component configuration.		Kinesis2Configuration
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (common)	To define a proxy host when instantiating the Kinesis client.		String
proxyPort (common)	To define a proxy port when instantiating the Kinesis client.		Integer
proxyProtocol (common)	To define a proxy protocol when instantiating the Kinesis client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
region (common)	The region in which Kinesis Firehose client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the Kinesis client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
iteratorType (consumer)	Defines where in the Kinesis stream to start getting records. Enum values: • AT_SEQUENCE_NUMBER • AFTER_SEQUENCE_NUMBER • TRIM_HORIZON • LATEST • AT_TIMESTAMP • null	TRIM_HORIZON	ShardIteratorType
maxResultsPerRequest (consumer)	Maximum number of records that will be fetched in each poll.	1	int
resumeStrategy (consumer)	Defines a resume strategy for AWS Kinesis. The default strategy reads the sequenceNumber if provided.	KinesisUserConfigurationResumeStrategy	KinesisResumeStrategy
sequenceNumber (consumer)	The sequence number to start polling from. Required if iteratorType is set to AFTER_SEQUENCE_NUMBER or AT_SEQUENCE_NUMBER.		String

Name	Description	Default	Type
shardClosed (consumer)	Define what will be the behavior in case of shard closed. Possible value are ignore, silent and fail. In case of ignore a message will be logged and the consumer will restart from the beginning,in case of silent there will be no logging and the consumer will start from the beginning,in case of fail a ReachedClosedStateException will be raised. Enum values: • ignore • fail • silent	ignore	Kinesis2ShardClosedStrategyEnum
shardId (consumer)	Defines which shardId in the Kinesis stream to get records from.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

4.4. ENDPOINT OPTIONS

The AWS Kinesis endpoint is configured using URI syntax:

```
aws2-kinesis:streamName
```

with the following path and query parameters:

4.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
streamName (common)	**Required** Name of the stream.		String

4.4.2. Query Parameters (38 parameters)

Name	Description	Default	Type
amazonKinesisClient (common)	**Autowired** Amazon Kinesis client to use for all requests for this endpoint.		KinesisClient
cborEnabled (common)	This option will set the CBOR_ENABLED property during the execution.	true	boolean
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
proxyHost (common)	To define a proxy host when instantiating the Kinesis client.		String
proxyPort (common)	To define a proxy port when instantiating the Kinesis client.		Integer
proxyProtocol (common)	To define a proxy protocol when instantiating the Kinesis client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
region (common)	The region in which Kinesis Firehose client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String

Name	Description	Default	Type
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the Kinesis client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
iteratorType (consumer)	Defines where in the Kinesis stream to start getting records. Enum values: • AT_SEQUENCE_NUMBER • AFTER_SEQUENCE_NUMBER • TRIM_HORIZON • LATEST • AT_TIMESTAMP • null	TRIM_HORIZON	ShardIteratorType
maxResultsPerRequest (consumer)	Maximum number of records that will be fetched in each poll.	1	int
resumeStrategy (consumer)	Defines a resume strategy for AWS Kinesis. The default strategy reads the sequenceNumber if provided.	KinesisUserConfigurationResumeStrategy	KinesisResumeStrategy

Name	Description	Default	Type
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
sequenceNumber (consumer)	The sequence number to start polling from. Required if iteratorType is set to AFTER_SEQUENCE_NUMBER or AT_SEQUENCE_NUMBER.		String
shardClosed (consumer)	Define what will be the behavior in case of shard closed. Possible value are ignore, silent and fail. In case of ignore a message will be logged and the consumer will restart from the beginning,in case of silent there will be no logging and the consumer will start from the beginning,in case of fail a ReachedClosedStateException will be raised. Enum values: • ignore • fail • silent	ignore	Kinesis2ShardClosedStrategyEnum
shardId (consumer)	Defines which shardId in the Kinesis stream to get records from.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required Kinesis component options

You have to provide the KinesisClient in the Registry with proxies and relevant credentials configured.

4.5. BATCH CONSUMER

This component implements the Batch Consumer.

This allows you for instance to know how many messages exists in this batch and for instance let the Aggregator aggregate this number of messages.

4.6. USAGE

4.6.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey
- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Web Identity Token from AWS STS.
- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

4.6.2. Message headers set by the Kinesis consumer

Header	Type	Description
CamelAwsKinesisSequenceNumber	**String**	The sequence number of the record. This is represented as a String as it size is not defined by the API. If it is to be used as a numerical type then use
CamelAwsKinesisApproximateArrivalTimestamp	**String**	The time AWS assigned as the arrival time of the record.
CamelAwsKinesisPartitionKey	**String**	Identifies which shard in the stream the data record is assigned to.

4.6.3. AmazonKinesis configuration

You then have to reference the KinesisClient in the **amazonKinesisClient** URI option.

```
from("aws2-kinesis://mykinesisstream?amazonKinesisClient=#kinesisClient")
  .to("log:out?showAll=true");
```

4.6.4. Providing AWS Credentials

It is recommended that the credentials are obtained by using the DefaultAWSCredentialsProviderChain that is the default when creating a new ClientConfiguration instance, however, a different AWSCredentialsProvider can be specified when calling createClient(...).

4.6.5. Message headers used by the Kinesis producer to write to Kinesis. The producer expects that the message body is a byte[].

Header	Type	Description
CamelAwsKinesisPartitionKey	**String**	The PartitionKey to pass to Kinesis to store this record.
CamelAwsKinesisSequenceNumber	**String**	Optional paramter to indicate the sequence number of this record.

4.6.6. Message headers set by the Kinesis producer on successful storage of a Record

Header	Type	Description
CamelAwsKinesisSequenceNumber	**String**	The sequence number of the record, as defined in Response Syntax
CamelAwsKinesisShardId	**String**	The shard ID of where the Record was stored

4.7. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-kinesis</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

4.8. SPRING BOOT AUTO-CONFIGURATION

When using aws2-kinesis with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-kinesis-starter</artifactId>
</dependency>
```

The component supports 40 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-kinesis-firehose.access-key	Amazon AWS Access Key.		String
camel.component.aws2-kinesis-firehose.amazon-kinesis-firehose-client	Amazon Kinesis Firehose client to use for all requests for this endpoint. The option is a software.amazon.awssdk.services.firehose.FirehoseClient type.		FirehoseClient
camel.component.aws2-kinesis-firehose.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-kinesis-firehose.cbor-enabled	This option will set the CBOR_ENABLED property during the execution.	true	Boolean

Name	Description	Default	Type
camel.component.aws2-kinesis-firehose.configuration	Component configuration. The option is a org.apache.camel.component.aws2.firehose.KinesisFirehose2Configuration type.		KinesisFirehose2Configuration
camel.component.aws2-kinesis-firehose.enabled	Whether to enable auto configuration of the aws2-kinesis-firehose component. This is enabled by default.		Boolean
camel.component.aws2-kinesis-firehose.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-kinesis-firehose.operation	The operation to do in case the user don't want to send only a record.		KinesisFirehose2Operations
camel.component.aws2-kinesis-firehose.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-kinesis-firehose.proxy-host	To define a proxy host when instantiating the Kinesis Firehose client.		String
camel.component.aws2-kinesis-firehose.proxy-port	To define a proxy port when instantiating the Kinesis Firehose client.		Integer
camel.component.aws2-kinesis-firehose.proxy-protocol	To define a proxy protocol when instantiating the Kinesis Firehose client.		Protocol

Name	Description	Default	Type
camel.component.aws2-kinesis-firehose.region	The region in which Kinesis Firehose client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-kinesis-firehose.secret-key	Amazon AWS Secret Key.		String
camel.component.aws2-kinesis-firehose.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-kinesis-firehose.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-kinesis-firehose.use-default-credentials-provider	Set whether the Kinesis Firehose client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean
camel.component.aws2-kinesis.access-key	Amazon AWS Access Key.		String
camel.component.aws2-kinesis.amazon-kinesis-client	Amazon Kinesis client to use for all requests for this endpoint. The option is a software.amazon.awssdk.services.kinesis.KinesisClient type.		KinesisClient
camel.component.aws2-kinesis.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.aws2-kinesis.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.aws2-kinesis.cbor-enabled	This option will set the CBOR_ENABLED property during the execution.	true	Boolean
camel.component.aws2-kinesis.configuration	Component configuration. The option is a org.apache.camel.component.aws2.kinesis.Kinesis2Configuration type.		Kinesis2Configuration
camel.component.aws2-kinesis.enabled	Whether to enable auto configuration of the aws2-kinesis component. This is enabled by default.		Boolean
camel.component.aws2-kinesis.iterator-type	Defines where in the Kinesis stream to start getting records.		ShardIteratorType
camel.component.aws2-kinesis.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-kinesis.max-results-per-request	Maximum number of records that will be fetched in each poll.	1	Integer

Name	Description	Default	Type
camel.component.aws2-kinesis.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-kinesis.proxy-host	To define a proxy host when instantiating the Kinesis client.		String
camel.component.aws2-kinesis.proxy-port	To define a proxy port when instantiating the Kinesis client.		Integer
camel.component.aws2-kinesis.proxy-protocol	To define a proxy protocol when instantiating the Kinesis client.		Protocol
camel.component.aws2-kinesis.region	The region in which Kinesis Firehose client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-kinesis.resume-strategy	Defines a resume strategy for AWS Kinesis. The default strategy reads the sequenceNumber if provided. The option is a org.apache.camel.component.aws2.kinesis.consumer.KinesisResumeStrategy type.		KinesisResumeStrategy
camel.component.aws2-kinesis.secret-key	Amazon AWS Secret Key.		String
camel.component.aws2-kinesis.sequence-number	The sequence number to start polling from. Required if iteratorType is set to AFTER_SEQUENCE_NUMBER or AT_SEQUENCE_NUMBER.		String
camel.component.aws2-kinesis.shard-closed	Define what will be the behavior in case of shard closed. Possible value are ignore, silent and fail. In case of ignore a message will be logged and the consumer will restart from the beginning,in case of silent there will be no logging and the consumer will start from the beginning,in case of fail a ReachedClosedStateException will be raised.		Kinesis2ShardClosedStrategyEnum

Name	Description	Default	Type
camel.component.aws2-kinesis.shard-id	Defines which shardId in the Kinesis stream to get records from.		String
camel.component.aws2-kinesis.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-kinesis.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-kinesis.use-default-credentials-provider	Set whether the Kinesis client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean

CHAPTER 5. AWS 2 LAMBDA

Only producer is supported

The AWS2 Lambda component supports create, get, list, delete and invoke AWS Lambda functions.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon Lambda. More information is available at AWS Lambda.

When creating a Lambda function, you need to specify a IAM role which has at least the AWSLambdaBasicExecuteRole policy attached.

5.1. URI FORMAT

```
aws2-lambda://functionName[?options]
```

You can append query options to the URI in the following format, **options=value&option2=value&…**

5.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

5.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

5.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

5.3. COMPONENT OPTIONS

The AWS Lambda component supports 16 options, which are listed below.

Name	Description	Default	Type
configuration (producer)	Component configuration.		Lambda2Configuration
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	The operation to perform. It can be listFunctions, getFunction, createFunction, deleteFunction or invokeFunction. Enum values: • listFunctions • getFunction • createAlias • deleteAlias • getAlias • listAliases • createFunction • deleteFunction • invokeFunction • updateFunction • createEventSourceMapping • deleteEventSourceMapping • listEventSourceMapping • listTags • tagResource • untagResource • publishVersion • listVersions	invokeFunction	Lambda2Operations
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
pojoRequest (producer)	If we want to use a POJO request as body or not.	false	boolean
region (producer)	The region in which Lambda client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String

Name	Description	Default	Type
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the Lambda client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
awsLambdaClient (advanced)	**Autowired** To use a existing configured AwsLambdaClient as client.		LambdaClient
proxyHost (proxy)	To define a proxy host when instantiating the Lambda client.		String
proxyPort (proxy)	To define a proxy port when instantiating the Lambda client.		Integer
proxyProtocol (proxy)	To define a proxy protocol when instantiating the Lambda client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

5.4. ENDPOINT OPTIONS

The AWS Lambda endpoint is configured using URI syntax:

```
aws2-lambda:function
```

with the following path and query parameters:

5.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
function (producer)	**Required** Name of the Lambda function.		String

5.4.2. Query Parameters (14 parameters)

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	The operation to perform. It can be listFunctions, getFunction, createFunction, deleteFunction or invokeFunction. Enum values: • listFunctions • getFunction • createAlias • deleteAlias • getAlias • listAliases • createFunction • deleteFunction • invokeFunction • updateFunction • createEventSourceMapping • deleteEventSourceMapping • listEventSourceMapping • listTags • tagResource • untagResource • publishVersion • listVersions	invokeFunction	Lambda2Operations
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
pojoRequest (producer)	If we want to use a POJO request as body or not.	false	boolean
region (producer)	The region in which Lambda client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String

Name	Description	Default	Type
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the Lambda client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
awsLambdaClient (advanced)	**Autowired** To use a existing configured AwsLambdaClient as client.		LambdaClient
proxyHost (proxy)	To define a proxy host when instantiating the Lambda client.		String
proxyPort (proxy)	To define a proxy port when instantiating the Lambda client.		Integer
proxyProtocol (proxy)	To define a proxy protocol when instantiating the Lambda client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required Lambda component options

You have to provide the awsLambdaClient in the Registry or your accessKey and secretKey to access the Amazon Lambda service..

5.5. USAGE

5.5.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey

- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Web Identity Token from AWS STS.
- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

5.5.2. Message headers evaluated by the Lambda producer

Operation	Header	Type	Description	Required
All	**CamelAwsLambdaOperation**	**String**	The operation we want to perform. Override operation passed as query parameter	Yes
createFunction	**CamelAwsLambdaS3Bucket**	**String**	Amazon S3 bucket name where the .zip file containing your deployment package is stored. This bucket must reside in the same AWS region where you are creating the Lambda function.	No
createFunction	**CamelAwsLambdaS3Key**	**String**	The Amazon S3 object (the deployment package) key name you want to upload.	No
createFunction	**CamelAwsLambdaS3ObjectVersion**	String	The Amazon S3 object (the deployment package) version you want to upload.	No
createFunction	**CamelAwsLambdaZipFile**	**String**	The local path of the zip file (the deployment package). Content of zip file can also be put in Message body.	No

Operation	Header	Type	Description	Required
createFunction	**CamelAwsLambdaRole**	**String**	The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.	Yes
createFunction	**CamelAwsLambdaRuntime**	String	The runtime environment for the Lambda function you are uploading. (nodejs, nodejs4.3, nodejs6.10, java8, python2.7, python3.6, dotnetcore1.0, odejs4.3-edge)	Yes
createFunction	**CamelAwsLambdaHandler**	**String**	The function within your code that Lambda calls to begin execution. For Node.js, it is the module-name.export value in your function. For Java, it can be package.class-name::handler or package.class-name.	Yes
createFunction	**CamelAwsLambdaDescription**	**String**	The user-provided description.	No
createFunction	**CamelAwsLambdaTargetArn**	**String**	The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.	No
createFunction	**CamelAwsLambdaMemorySize**	**Integer**	The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.	No

Operation	Header	Type	Description	Required
createFunction	**CamelAwsLambdaKMSKeyArn**	**String**	The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If not provided, AWS Lambda will use a default service key.	No
createFunction	**CamelAwsLambdaPublish**	**Boolean**	This boolean parameter can be used to request AWS Lambda to create the Lambda function and publish a version as an atomic operation.	No
createFunction	**CamelAwsLambdaTimeout**	**Integer**	The function execution time at which Lambda should terminate the function. The default is 3 seconds.	No
createFunction	**CamelAwsLambdaTracingConfig**	**String**	Your function's tracing settings (Active or PassThrough).	No
createFunction	**CamelAwsLambdaEnvironmentVariables**	**Map<String, String>**	The key-value pairs that represent your environment's configuration settings.	No
createFunction	**CamelAwsLambdaEnvironmentTags**	**Map<String, String>**	The list of tags (key-value pairs) assigned to the new function.	No
createFunction	**CamelAwsLambdaSecurityGroupIds**	**List<String>**	If your Lambda function accesses resources in a VPC, a list of one or more security groups IDs in your VPC.	No
createFunction	**CamelAwsLambdaSubnetIds**	**List<String>**	If your Lambda function accesses resources in a VPC, a list of one or more subnet IDs in your VPC.	No
createAlias	**CamelAwsLambdaFunctionVersion**	**String**	The function version to set in the alias	Yes

Operation	Header	Type	Description	Required
createAlias	**CamelAwsLambdaAliasFunctionName**	**String**	The function name to set in the alias	Yes
createAlias	**CamelAwsLambdaAliasFunctionDescription**	**String**	The function description to set in the alias	No
deleteAlias	**CamelAwsLambdaAliasFunctionName**	**String**	The function name of the alias	Yes
getAlias	**CamelAwsLambdaAliasFunctionName**	**String**	The function name of the alias	Yes
listAliases	**CamelAwsLambdaFunctionVersion**	**String**	The function version to set in the alias	No

5.6. LIST OF AVALAIBLE OPERATIONS

- listFunctions
- getFunction
- createFunction
- deleteFunction
- invokeFunction
- updateFunction
- createEventSourceMapping
- deleteEventSourceMapping
- listEventSourceMapping
- listTags
- tagResource
- untagResource
- publishVersion
- listVersions
- createAlias
- deleteAlias
- getAlias

- listAliases

5.7. EXAMPLES

5.7.1. Producer Example

To have a full understanding of how the component works, you may have a look at these integration tests.

5.7.2. Producer Examples

- CreateFunction: this operation will create a function for you in AWS Lambda

```
  from("direct:createFunction").to("aws2-lambda://GetHelloWithName?
operation=createFunction").to("mock:result");
```

and by sending

```
template.send("direct:createFunction", ExchangePattern.InOut, new Processor() {
    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(Lambda2Constants.RUNTIME, "nodejs6.10");
        exchange.getIn().setHeader(Lambda2Constants.HANDLER, "GetHelloWithName.handler");
        exchange.getIn().setHeader(Lambda2Constants.DESCRIPTION, "Hello with node.js on
Lambda");
        exchange.getIn().setHeader(Lambda2Constants.ROLE,
            "arn:aws:iam::643534317684:role/lambda-execution-role");
        ClassLoader classLoader = getClass().getClassLoader();
        File file = new File(
            classLoader

.getResource("org/apache/camel/component/aws2/lambda/function/node/GetHelloWithName.zip")
                .getFile());
        FileInputStream inputStream = new FileInputStream(file);
        exchange.getIn().setBody(inputStream);
    }
});
```

5.8. USING A POJO AS BODY

Sometimes build an AWS Request can be complex, because of multiple options. We introduce the possibility to use a POJO as body. In AWS Lambda there are multiple operations you can submit, as an example for Get Function request, you can do something like:

```
from("direct:getFunction")
    .setBody(GetFunctionRequest.builder().functionName("test").build())
    .to("aws2-lambda://GetHelloWithName?
awsLambdaClient=#awsLambdaClient&operation=getFunction&pojoRequest=true")
```

In this way you'll pass the request directly without the need of passing headers and options specifically related to this operation.

5.9. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-lambda</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

5.10. SPRING BOOT AUTO-CONFIGURATION

When using aws2-lambda with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-lambda-starter</artifactId>
</dependency>
```

The component supports 17 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-lambda.access-key	Amazon AWS Access Key.		String
camel.component.aws2-lambda.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-lambda.aws-lambda-client	To use a existing configured AwsLambdaClient as client. The option is a software.amazon.awssdk.services.lambda.LambdaClient type.		LambdaClient
camel.component.aws2-lambda.configuration	Component configuration. The option is a org.apache.camel.component.aws2.lambda.Lambda2Configuration type.		Lambda2Configuration

Name	Description	Default	Type
camel.component.aws2-lambda.enabled	Whether to enable auto configuration of the aws2-lambda component. This is enabled by default.		Boolean
camel.component.aws2-lambda.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-lambda.operation	The operation to perform. It can be listFunctions, getFunction, createFunction, deleteFunction or invokeFunction.		Lambda2Operations
camel.component.aws2-lambda.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-lambda.pojo-request	If we want to use a POJO request as body or not.	false	Boolean
camel.component.aws2-lambda.proxy-host	To define a proxy host when instantiating the Lambda client.		String
camel.component.aws2-lambda.proxy-port	To define a proxy port when instantiating the Lambda client.		Integer
camel.component.aws2-lambda.proxy-protocol	To define a proxy protocol when instantiating the Lambda client.		Protocol

Name	Description	Default	Type
camel.component.aws2-lambda.region	The region in which Lambda client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-lambda.secret-key	Amazon AWS Secret Key.		String
camel.component.aws2-lambda.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-lambda.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-lambda.use-default-credentials-provider	Set whether the Lambda client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean

CHAPTER 6. AWS S3 STORAGE SERVICE

Both producer and consumer are supported

The AWS2 S3 component supports storing and retrieving objects from/to Amazon's S3 service.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon S3. More information is available at link:https://aws.amazon.com/s3 [Amazon S3].

6.1. URI FORMAT

```
aws2-s3://bucketNameOrArn[?options]
```

The bucket will be created if it don't already exists. You can append query options to the URI in the following format,

options=value&option2=value&…

6.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

6.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

6.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

6.3. COMPONENT OPTIONS

The AWS S3 Storage Service component supports 50 options, which are listed below.

Name	Description	Default	Type
amazonS3Client (common)	**Autowired** Reference to a com.amazonaws.services.s3.AmazonS3 in the registry.		S3Client
amazonS3Presigner (common)	**Autowired** An S3 Presigner for Request, used mainly in createDownloadLink operation.		S3Presigner
autoCreateBucket (common)	Setting the autocreation of the S3 bucket bucketName. This will apply also in case of moveAfterRead option enabled and it will create the destinationBucket if it doesn't exist already.	false	boolean
configuration (common)	The component configuration.		AWS2S3Configuration
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
pojoRequest (common)	If we want to use a POJO request as body or not.	false	boolean
policy (common)	The policy for this queue to set in the com.amazonaws.services.s3.AmazonS3#setBucketPolicy() method.		String
proxyHost (common)	To define a proxy host when instantiating the SQS client.		String
proxyPort (common)	Specify a proxy port to be used inside the client definition.		Integer
proxyProtocol (common)	To define a proxy protocol when instantiating the S3 client. Enum values: • HTTP • HTTPS	HTTPS	Protocol

Name	Description	Default	Type
region (common)	The region in which S3 client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
customerAlgorithm (common (advanced))	Define the customer algorithm to use in case CustomerKey is enabled.		String
customerKeyId (common (advanced))	Define the id of Customer key to use in case CustomerKey is enabled.		String
customerKeyMD5 (common (advanced))	Define the MD5 of Customer key to use in case CustomerKey is enabled.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
deleteAfterRead (consumer)	Delete objects from S3 after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the AWS2S3Constants#BUCKET_NAME and AWS2S3Constants#KEY headers, or only the AWS2S3Constants#KEY header.	true	boolean
delimiter (consumer)	The delimiter which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
destinationBucket (consumer)	Define the destination bucket where an object must be moved when moveAfterRead is set to true.		String
destinationBucketPrefix (consumer)	Define the destination bucket prefix to use when an object must be moved and moveAfterRead is set to true.		String
destinationBucketSuffix (consumer)	Define the destination bucket suffix to use when an object must be moved and moveAfterRead is set to true.		String
doneFileName (consumer)	If provided, Camel will only consume files if a done file exists.		String
fileName (consumer)	To get the object from the bucket with the given file name.		String
ignoreBody (consumer)	If it is true, the S3 Object Body will be ignored completely, if it is set to false the S3 Object will be put in the body. Setting this to true, will override any behavior defined by includeBody option.	false	boolean

Name	Description	Default	Type
includeBody (consumer)	If it is true, the S3Object exchange will be consumed and put into the body and closed. If false the S3Object stream will be put raw into the body and the headers will be set with the S3 object metadata. This option is strongly related to autocloseBody option. In case of setting includeBody to true because the S3Object stream will be consumed then it will also be closed, while in case of includeBody false then it will be up to the caller to close the S3Object stream. However setting autocloseBody to true when includeBody is false it will schedule to close the S3Object stream automatically on exchange completion.	true	boolean
includeFolders (consumer)	If it is true, the folders/directories will be consumed. If it is false, they will be ignored, and Exchanges will not be created for those.	true	boolean
moveAfterRead (consumer)	Move objects from S3 bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	boolean
prefix (consumer)	The prefix which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
autocloseBody (consumer (advanced))	If this option is true and includeBody is false, then the S3Object.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to false and autocloseBody to false, it will be up to the caller to close the S3Object stream. Setting autocloseBody to true, will close the S3Object stream automatically.	true	boolean
batchMessageNumber (producer)	The number of messages composing a batch in streaming upload mode.	10	int
batchSize (producer)	The batch size (in bytes) in streaming upload mode.	1000000	int
deleteAfterWrite (producer)	Delete file object after the S3 file has been uploaded.	false	boolean
keyName (producer)	Setting the key name for an element in the bucket through endpoint parameter.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
multiPartUpload (producer)	If it is true, camel will upload the file with multi part format, the part size is decided by the option of partSize.	false	boolean
namingStrategy (producer)	The naming strategy to use in streaming upload mode. Enum values: • progressive • random	progressive	AWSS3NamingStrategyEnum
operation (producer)	The operation to do in case the user don't want to do only an upload. Enum values: • copyObject • listObjects • deleteObject • deleteBucket • listBuckets • getObject • getObjectRange • createDownloadLink		AWS2S3Operations
partSize (producer)	Setup the partSize which is used in multi part upload, the default size is 25M.	26214400	long

Name	Description	Default	Type
restartingPolicy (producer)	The restarting policy to use in streaming upload mode. Enum values: • override • lastPart	override	AWSS3RestartingPolicyEnum
storageClass (producer)	The storage class to set in the com.amazonaws.services.s3.model.PutObjectRequest request.		String
streamingUploadMode (producer)	When stream mode is true the upload to bucket will be done in streaming.	false	boolean
streamingUploadTimeout (producer)	While streaming upload mode is true, this option set the timeout to complete upload.		long
awsKMSKeyId (producer (advanced))	Define the id of KMS key to use in case KMS is enabled.		String
useAwsKMS (producer (advanced))	Define if KMS must be used or not.	false	boolean
useCustomerKey (producer (advanced))	Define if Customer Key must be used or not.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

6.4. ENDPOINT OPTIONS

The AWS S3 Storage Service endpoint is configured using URI syntax:

```
aws2-s3://bucketNameOrArn
```

with the following path and query parameters:

6.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
bucketNameOrArn (common)	**Required** Bucket name or ARN.		String

6.4.2. Query Parameters (68 parameters)

Name	Description	Default	Type
amazonS3Client (common)	**Autowired** Reference to a com.amazonaws.services.s3.AmazonS3 in the registry.		S3Client
amazonS3Presigner (common)	**Autowired** An S3 Presigner for Request, used mainly in createDownloadLink operation.		S3Presigner
autoCreateBucket (common)	Setting the autocreation of the S3 bucket bucketName. This will apply also in case of moveAfterRead option enabled and it will create the destinationBucket if it doesn't exist already.	false	boolean
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
pojoRequest (common)	If we want to use a POJO request as body or not.	false	boolean
policy (common)	The policy for this queue to set in the com.amazonaws.services.s3.AmazonS3#setBucketPolicy() method.		String
proxyHost (common)	To define a proxy host when instantiating the SQS client.		String
proxyPort (common)	Specify a proxy port to be used inside the client definition.		Integer

Name	Description	Default	Type
proxyProtocol (common)	To define a proxy protocol when instantiating the S3 client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
region (common)	The region in which S3 client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	boolean
customerAlgorithm (common (advanced))	Define the customer algorithm to use in case CustomerKey is enabled.		String
customerKeyId (common (advanced))	Define the id of Customer key to use in case CustomerKey is enabled.		String
customerKeyMD5 (common (advanced))	Define the MD5 of Customer key to use in case CustomerKey is enabled.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
deleteAfterRead (consumer)	Delete objects from S3 after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the AWS2S3Constants#BUCKET_NAME and AWS2S3Constants#KEY headers, or only the AWS2S3Constants#KEY header.	true	boolean
delimiter (consumer)	The delimiter which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
destinationBucket (consumer)	Define the destination bucket where an object must be moved when moveAfterRead is set to true.		String
destinationBucketPrefix (consumer)	Define the destination bucket prefix to use when an object must be moved and moveAfterRead is set to true.		String
destinationBucketSuffix (consumer)	Define the destination bucket suffix to use when an object must be moved and moveAfterRead is set to true.		String
doneFileName (consumer)	If provided, Camel will only consume files if a done file exists.		String
fileName (consumer)	To get the object from the bucket with the given file name.		String
ignoreBody (consumer)	If it is true, the S3 Object Body will be ignored completely, if it is set to false the S3 Object will be put in the body. Setting this to true, will override any behavior defined by includeBody option.	false	boolean

Name	Description	Default	Type
includeBody (consumer)	If it is true, the S3Object exchange will be consumed and put into the body and closed. If false the S3Object stream will be put raw into the body and the headers will be set with the S3 object metadata. This option is strongly related to autocloseBody option. In case of setting includeBody to true because the S3Object stream will be consumed then it will also be closed, while in case of includeBody false then it will be up to the caller to close the S3Object stream. However setting autocloseBody to true when includeBody is false it will schedule to close the S3Object stream automatically on exchange completion.	true	boolean
includeFolders (consumer)	If it is true, the folders/directories will be consumed. If it is false, they will be ignored, and Exchanges will not be created for those.	true	boolean
maxConnections (consumer)	Set the maxConnections parameter in the S3 client configuration.	60	int
maxMessagesPerPoll (consumer)	Gets the maximum number of messages as a limit to poll at each polling. Gets the maximum number of messages as a limit to poll at each polling. The default value is 10. Use 0 or a negative number to set it as unlimited.	10	int
moveAfterRead (consumer)	Move objects from S3 bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	boolean
prefix (consumer)	The prefix which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean

Name	Description	Default	Type
autocloseBody (consumer (advanced))	If this option is true and includeBody is false, then the S3Object.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to false and autocloseBody to false, it will be up to the caller to close the S3Object stream. Setting autocloseBody to true, will close the S3Object stream automatically.	true	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
batchMessageNumber (producer)	The number of messages composing a batch in streaming upload mode.	10	int
batchSize (producer)	The batch size (in bytes) in streaming upload mode.	1000000	int
deleteAfterWrite (producer)	Delete file object after the S3 file has been uploaded.	false	boolean
keyName (producer)	Setting the key name for an element in the bucket through endpoint parameter.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
multiPartUpload (producer)	If it is true, camel will upload the file with multi part format, the part size is decided by the option of partSize.	false	boolean
namingStrategy (producer)	The naming strategy to use in streaming upload mode. Enum values: • progressive • random	progressive	AWSS3NamingStrategyEnum
operation (producer)	The operation to do in case the user don't want to do only an upload. Enum values: • copyObject • listObjects • deleteObject • deleteBucket • listBuckets • getObject • getObjectRange • createDownloadLink		AWS2S3Operations
partSize (producer)	Setup the partSize which is used in multi part upload, the default size is 25M.	26214400	long

Name	Description	Default	Type
restartingPolicy (producer)	The restarting policy to use in streaming upload mode. Enum values: • override • lastPart	override	AWSS3RestartingPolicyEnum
storageClass (producer)	The storage class to set in the com.amazonaws.services.s3.model.PutObjectRequest request.		String
streamingUploadMode (producer)	When stream mode is true the upload to bucket will be done in streaming.	false	boolean
streamingUploadTimeout (producer)	While streaming upload mode is true, this option set the timeout to complete upload.		long
awsKMSKeyId (producer (advanced))	Define the id of KMS key to use in case KMS is enabled.		String
useAwsKMS (producer (advanced))	Define if KMS must be used or not.	false	boolean
useCustomerKey (producer (advanced))	Define if Customer Key must be used or not.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long

Name	Description	Default	Type
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean

Name	Description	Default	Type
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required S3 component options

You have to provide the amazonS3Client in the Registry or your accessKey and secretKey to access the Amazon's S3.

6.5. BATCH CONSUMER

This component implements the Batch Consumer.

This allows you for instance to know how many messages exists in this batch and for instance let the Aggregator aggregate this number of messages.

6.6. USAGE

For example in order to read file **hello.txt** from bucket **helloBucket**, use the following snippet:

```
from("aws2-s3://helloBucket?
accessKey=yourAccessKey&secretKey=yourSecretKey&prefix=hello.txt")
  .to("file:/var/downloaded");
```

6.6.1. Message headers evaluated by the S3 producer

Header	Type	Description
CamelAwsS3BucketName	**String**	The bucket Name which this object will be stored or which will be used for the current operation
CamelAwsS3BucketDestinationName	**String**	The bucket Destination Name which will be used for the current operation
CamelAwsS3ContentLength	**Long**	The content length of this object.
CamelAwsS3ContentType	**String**	The content type of this object.
CamelAwsS3ContentControl	**String**	The content control of this object.
CamelAwsS3ContentDisposition	**String**	The content disposition of this object.
CamelAwsS3ContentEncoding	**String**	The content encoding of this object.
CamelAwsS3ContentMD5	**String**	The md5 checksum of this object.
CamelAwsS3DestinationKey	**String**	The Destination key which will be used for the current operation
CamelAwsS3Key	**String**	The key under which this object will be stored or which will be used for the current operation
CamelAwsS3LastModified	**java.util.Date**	The last modified timestamp of this object.
CamelAwsS3Operation	**String**	The operation to perform. Permitted values are copyObject, deleteObject, listBuckets, deleteBucket, listObjects
CamelAwsS3StorageClass	**String**	The storage class of this object.
CamelAwsS3CannedAcl	**String**	The canned acl that will be applied to the object. see **software.amazon.awssdk.services.s3.model.ObjectCannedACL** for allowed values.
CamelAwsS3Acl	**software.amazon.awssdk.services.s3.model.BucketCannedACL**	A well constructed Amazon S3 Access Control List object. see **software.amazon.awssdk.services.s3.model.BucketCannedACL** for more details

Header	Type	Description
CamelAwsS3ServerSideEncryption	String	Sets the server-side encryption algorithm when encrypting the object using AWS-managed keys. For example use AES256.
CamelAwsS3VersionId	**String**	The version Id of the object to be stored or returned from the current operation
CamelAwsS3Metadata	**Map<String, String>**	A map of metadata to be stored with the object in S3. More details about metadata .

6.6.2. Message headers set by the S3 producer

Header	Type	Description
CamelAwsS3ETag	**String**	The ETag value for the newly uploaded object.
CamelAwsS3VersionId	**String**	The **optional** version ID of the newly uploaded object.

6.6.3. Message headers set by the S3 consumer

Header	Type	Description
CamelAwsS3Key	**String**	The key under which this object is stored.
CamelAwsS3BucketName	**String**	The name of the bucket in which this object is contained.
CamelAwsS3ETag	**String**	The hex encoded 128-bit MD5 digest of the associated object according to RFC 1864. This data is used as an integrity check to verify that the data received by the caller is the same data that was sent by Amazon S3.
CamelAwsS3LastModified	**Date**	The value of the Last-Modified header, indicating the date and time at which Amazon S3 last recorded a modification to the associated object.
CamelAwsS3VersionId	**String**	The version ID of the associated Amazon S3 object if available. Version IDs are only assigned to objects when an object is uploaded to an Amazon S3 bucket that has object versioning enabled.

Header	Type	Description
CamelAwsS3ContentType	**String**	The Content-Type HTTP header, which indicates the type of content stored in the associated object. The value of this header is a standard MIME type.
CamelAwsS3ContentMD5	**String**	The base64 encoded 128-bit MD5 digest of the associated object (content - not including headers) according to RFC 1864. This data is used as a message integrity check to verify that the data received by Amazon S3 is the same data that the caller sent.
CamelAwsS3ContentLength	**Long**	The Content-Length HTTP header indicating the size of the associated object in bytes.
CamelAwsS3ContentEncoding	**String**	The **optional** Content-Encoding HTTP header specifying what content encodings have been applied to the object and what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type field.
CamelAwsS3ContentDisposition	**String**	The **optional** Content-Disposition HTTP header, which specifies presentational information such as the recommended filename for the object to be saved as.
CamelAwsS3ContentControl	**String**	The **optional** Cache-Control HTTP header which allows the user to specify caching behavior along the HTTP request/reply chain.
CamelAwsS3ServerSideEncryption	String	The server-side encryption algorithm when encrypting the object using AWS-managed keys.
CamelAwsS3Metadata	**Map<String, String>**	A map of metadata stored with the object in S3. More details about metadata .

6.6.4. S3 Producer operations

Camel-AWS2-S3 component provides the following operation on the producer side:

- copyObject
- deleteObject
- listBuckets
- deleteBucket

- listObjects
- getObject (this will return an S3Object instance)
- getObjectRange (this will return an S3Object instance)
- createDownloadLink

If you don't specify an operation explicitly the producer will do: - a single file upload - a multipart upload if multiPartUpload option is enabled.

6.6.5. Advanced AmazonS3 configuration

If your Camel Application is running behind a firewall or if you need to have more control over the **S3Client** instance configuration, you can create your own instance and refer to it in your Camel aws2-s3 component configuration:

```
from("aws2-s3://MyBucket?amazonS3Client=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

6.6.6. Use KMS with the S3 component

To use AWS KMS to encrypt/decrypt data by using AWS infrastructure you can use the options introduced in 2.21.x like in the following example

```
from("file:tmp/test?fileName=test.txt")
    .setHeader(S3Constants.KEY, constant("testFile"))
    .to("aws2-s3://mybucket?amazonS3Client=#client&useAwsKMS=true&awsKMSKeyId=3f0637ad-296a-3dfe-a796-e60654fb128c");
```

In this way you'll ask to S3, to use the KMS key 3f0637ad-296a-3dfe-a796-e60654fb128c, to encrypt the file test.txt. When you'll ask to download this file, the decryption will be done directly before the download.

6.6.7. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey
- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- Web Identity Token from AWS STS.
- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

6.6.8. S3 Producer Operation examples

- Single Upload: This operation will upload a file to S3 based on the body content

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.KEY, "camel.txt");
        exchange.getIn().setBody("Camel rocks!");
    }
})
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client")
.to("mock:result");
```

This operation will upload the file camel.txt with the content "Camel rocks!" in the mycamelbucket bucket

- Multipart Upload: This operation will perform a multipart upload of a file to S3 based on the body content

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(AWS2S3Constants.KEY, "empty.txt");
        exchange.getIn().setBody(new File("src/empty.txt"));
    }
})
.to("aws2-s3://mycamelbucket?
amazonS3Client=#amazonS3Client&multiPartUpload=true&autoCreateBucket=true&partSize=1048576"
)
.to("mock:result");
```

This operation will perform a multipart upload of the file empty.txt with based on the content the file src/empty.txt in the mycamelbucket bucket

- CopyObject: this operation copy an object from one bucket to a different one

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.BUCKET_DESTINATION_NAME,
"camelDestinationBucket");
        exchange.getIn().setHeader(S3Constants.KEY, "camelKey");
        exchange.getIn().setHeader(S3Constants.DESTINATION_KEY, "camelDestinationKey");
    }
})
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=copyObject")
.to("mock:result");
```

This operation will copy the object with the name expressed in the header camelDestinationKey to the camelDestinationBucket bucket, from the bucket mycamelbucket.

- DeleteObject: this operation deletes an object from a bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.KEY, "camelKey");
    }
})
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=deleteObject")
.to("mock:result");
```

This operation will delete the object camelKey from the bucket mycamelbucket.

- ListBuckets: this operation list the buckets for this account in this region

```
from("direct:start")
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=listBuckets")
.to("mock:result");
```

This operation will list the buckets for this account

- DeleteBucket: this operation delete the bucket specified as URI parameter or header

```
from("direct:start")
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=deleteBucket")
.to("mock:result");
```

This operation will delete the bucket mycamelbucket

- ListObjects: this operation list object in a specific bucket

```
from("direct:start")
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=listObjects")
.to("mock:result");
```

This operation will list the objects in the mycamelbucket bucket

- GetObject: this operation get a single object in a specific bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.KEY, "camelKey");
    }
})
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=getObject")
.to("mock:result");
```

This operation will return an S3Object instance related to the camelKey object in mycamelbucket bucket.

- GetObjectRange: this operation get a single object range in a specific bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.KEY, "camelKey");
        exchange.getIn().setHeader(S3Constants.RANGE_START, "0");
        exchange.getIn().setHeader(S3Constants.RANGE_END, "9");
    }
})
.to("aws2-s3://mycamelbucket?amazonS3Client=#amazonS3Client&operation=getObjectRange")
.to("mock:result");
```

This operation will return an S3Object instance related to the camelKey object in mycamelbucket bucket, containing a the bytes from 0 to 9.

- CreateDownloadLink: this operation will return a download link through S3 Presigner

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(S3Constants.KEY, "camelKey");
    }
})
.to("aws2-s3://mycamelbucket?
accessKey=xxx&secretKey=yyy&region=region&operation=createDownloadLink")
.to("mock:result");
```

This operation will return a download link url for the file camel-key in the bucket mycamelbucket and region region

6.7. STREAMING UPLOAD MODE

With the stream mode enabled users will be able to upload data to S3 without knowing ahead of time the dimension of the data, by leveraging multipart upload. The upload will be completed when: the batchSize has been completed or the batchMessageNumber has been reached. There are two possible naming strategy:

- progressive
  With the progressive strategy each file will have the name composed by keyName option and a progressive counter, and eventually the file extension (if any)

- random.
  With the random strategy a UUID will be added after keyName and eventually the file extension will appended.

As an example:

```
from(kafka("topic1").brokers("localhost:9092"))
        .log("Kafka Message is: ${body}")
        .to(aws2S3("camel-
bucket").streamingUploadMode(true).batchMessageNumber(25).namingStrategy(AWS2S3EndpointBu
ilderFactory.AWSS3NamingStrategyEnum.progressive).keyName("
{{kafkaTopic1}}/{{kafkaTopic1}}.txt"));
```

```
from(kafka("topic2").brokers("localhost:9092"))
        .log("Kafka Message is: ${body}")
        .to(aws2S3("camel-
bucket").streamingUploadMode(true).batchMessageNumber(25).namingStrategy(AWS2S3EndpointBu
ilderFactory.AWSS3NamingStrategyEnum.progressive).keyName("
{{kafkaTopic2}}/{{kafkaTopic2}}.txt"));
```

The default size for a batch is 1 Mb, but you can adjust it according to your requirements.

When you'll stop your producer route, the producer will take care of flushing the remaining buffered messaged and complete the upload.

In Streaming upload you'll be able restart the producer from the point where it left. It's important to note that this feature is critical only when using the progressive naming strategy.

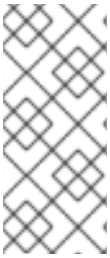
By setting the restartingPolicy to lastPart, you will restart uploading files and contents from the last part number the producer left.

Example

1. Start the route with progressive naming strategy and keyname equals to camel.txt, with batchMessageNumber equals to 20, and restartingPolicy equals to lastPart - Send 70 messages.
2. Stop the route
3. On your S3 bucket you should now see 4 files: * camel.txt
   - camel-1.txt
   - camel-2.txt
   - camel-3.txt
     The first three will have 20 messages, while the last one only 10.
4. Restart the route.
5. Send 25 messages.
6. Stop the route.
7. You'll now have 2 other files in your bucket: camel-5.txt and camel-6.txt, the first with 20 messages and second with 5 messages.
8. Go ahead

This won't be needed when using the random naming strategy.

On the opposite you can specify the override restartingPolicy. In that case you'll be able to override whatever you written before (for that particular keyName) on your bucket.



NOTE

In Streaming upload mode the only keyName option that will be taken into account is the endpoint option. Using the header will throw an NPE and this is done by design. Setting the header means potentially change the file name on each exchange and this is against the aim of the streaming upload producer. The keyName needs to be fixed and static. The selected naming strategy will do the rest of the of the work.

Another possibility is specifying a streamingUploadTimeout with batchMessageNumber and batchSize options. With this option the user will be able to complete the upload of a file after a certain time passed. In this way the upload completion will be passed on three tiers: the timeout, the number of messages and the batch size.

As an example:

```
from(kafka("topic1").brokers("localhost:9092"))
        .log("Kafka Message is: ${body}")
        .to(aws2S3("camel-
bucket").streamingUploadMode(true).batchMessageNumber(25).streamingUploadTimeout(10000).na
mingStrategy(AWS2S3EndpointBuilderFactory.AWSS3NamingStrategyEnum.progressive).keyName(
"{{kafkaTopic1}}/{{kafkaTopic1}}.txt"));
```

In this case the upload will be completed after 10 seconds.

6.8. BUCKET AUTOCREATION

With the option **autoCreateBucket** users are able to avoid the autocreation of an S3 Bucket in case it doesn't exist. The default for this option is **true**. If set to false any operation on a not-existent bucket in AWS won't be successful and an error will be returned.

6.9. MOVING STUFF BETWEEN A BUCKET AND ANOTHER BUCKET

Some users like to consume stuff from a bucket and move the content in a different one without using the copyObject feature of this component. If this is case for you, don't forget to remove the bucketName header from the incoming exchange of the consumer, otherwise the file will be always overwritten on the same original bucket.

6.10. MOVEAFTERREAD CONSUMER OPTION

In addition to deleteAfterRead it has been added another option, moveAfterRead. With this option enabled the consumed object will be moved to a target destinationBucket instead of being only deleted. This will require specifying the destinationBucket option. As example:

```
  from("aws2-s3://mycamelbucket?
amazonS3Client=#amazonS3Client&moveAfterRead=true&destinationBucket=myothercamelbucket")
   .to("mock:result");
```

In this case the objects consumed will be moved to myothercamelbucket bucket and deleted from the original one (because of deleteAfterRead set to true as default).

You have also the possibility of using a key prefix/suffix while moving the file to a different bucket. The options are destinationBucketPrefix and destinationBucketSuffix.

Taking the above example, you could do something like:

```
  from("aws2-s3://mycamelbucket?
amazonS3Client=#amazonS3Client&moveAfterRead=true&destinationBucket=myothercamelbucket&de
tinationBucketPrefix=RAW(pre-)&destinationBucketSuffix=RAW(-suff)")
   .to("mock:result");
```

In this case the objects consumed will be moved to myothercamelbucket bucket and deleted from the original one (because of deleteAfterRead set to true as default).

So if the file name is test, in the myothercamelbucket you should see a file called pre-test-suff.

6.11. USING CUSTOMER KEY AS ENCRYPTION

We introduced also the customer key support (an alternative of using KMS). The following code shows an example.

```
String key = UUID.randomUUID().toString();
byte[] secretKey = generateSecretKey();
String b64Key = Base64.getEncoder().encodeToString(secretKey);
String b64KeyMd5 = Md5Utils.md5AsBase64(secretKey);

String awsEndpoint = "aws2-s3://mycamel?
autoCreateBucket=false&useCustomerKey=true&customerKeyId=RAW(" + b64Key +
")&customerKeyMD5=RAW(" + b64KeyMd5 + ")&customerAlgorithm=" + AES256.name();

from("direct:putObject")
    .setHeader(AWS2S3Constants.KEY, constant("test.txt"))
    .setBody(constant("Test"))
    .to(awsEndpoint);
```

6.12. USING A POJO AS BODY

Sometimes build an AWS Request can be complex, because of multiple options. We introduce the possibility to use a POJO as body. In AWS S3 there are multiple operations you can submit, as an example for List brokers request, you can do something like:

```
from("direct:aws2-s3")
     .setBody(ListObjectsRequest.builder().bucket(bucketName).build())
     .to("aws2-s3://test?
amazonS3Client=#amazonS3Client&operation=listObjects&pojoRequest=true")
```

In this way you'll pass the request directly without the need of passing headers and options specifically related to this operation.

6.13. CREATE S3 CLIENT AND ADD COMPONENT TO REGISTRY

Sometimes you would want to perform some advanced configuration using AWS2S3Configuration which also allows to set the S3 client. You can create and set the S3 client in the component configuration as shown in the following example

```
String awsBucketAccessKey = "your_access_key";
String awsBucketSecretKey = "your_secret_key";
```

```
S3Client s3Client =
S3Client.builder().credentialsProvider(StaticCredentialsProvider.create(AwsBasicCredentials.create(aws
BucketAccessKey, awsBucketSecretKey)))
            .region(Region.US_EAST_1).build();

AWS2S3Configuration configuration = new AWS2S3Configuration();
configuration.setAmazonS3Client(s3Client);
configuration.setAutoDiscoverClient(true);
configuration.setBucketName("s3bucket2020");
configuration.setRegion("us-east-1");
```

Now you can configure the S3 component (using the configuration object created above) and add it to the registry in the configure method before initialization of routes.

```
AWS2S3Component s3Component = new AWS2S3Component(getContext());
s3Component.setConfiguration(configuration);
s3Component.setLazyStartProducer(true);
camelContext.addComponent("aws2-s3", s3Component);
```

Now your component will be used for all the operations implemented in camel routes.

6.14. DEPENDENCIES

Maven users will need to add the following dependency to their **pom.xml**.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-s3</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

6.15. SPRING BOOT AUTO-CONFIGURATION

When using aws2-s3 with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-s3-starter</artifactId>
</dependency>
```

The component supports 51 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-s3.access-key	Amazon AWS Access Key.		String
camel.component.aws2-s3.amazon-s3-client	Reference to a com.amazonaws.services.s3.AmazonS3 in the registry. The option is a software.amazon.awssdk.services.s3.S3Client type.		S3Client
camel.component.aws2-s3.amazon-s3-presigner	An S3 Presigner for Request, used mainly in createDownloadLink operation. The option is a software.amazon.awssdk.services.s3.presigner.S3Presigner type.		S3Presigner
camel.component.aws2-s3.auto-create-bucket	Setting the autocreation of the S3 bucket bucketName. This will apply also in case of moveAfterRead option enabled and it will create the destinationBucket if it doesn't exist already.	false	Boolean
camel.component.aws2-s3.autoclose-body	If this option is true and includeBody is false, then the S3Object.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to false and autocloseBody to false, it will be up to the caller to close the S3Object stream. Setting autocloseBody to true, will close the S3Object stream automatically.	true	Boolean
camel.component.aws2-s3.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-s3.aws-k-m-s-key-id	Define the id of KMS key to use in case KMS is enabled.		String
camel.component.aws2-s3.batch-message-number	The number of messages composing a batch in streaming upload mode.	10	Integer

Name	Description	Default	Type
camel.component.aws2-s3.batch-size	The batch size (in bytes) in streaming upload mode.	1000000	Integer
camel.component.aws2-s3.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.aws2-s3.configuration	The component configuration. The option is a org.apache.camel.component.aws2.s3.AWS2S3Configuration type.		AWS2S3Configuration
camel.component.aws2-s3.customer-algorithm	Define the customer algorithm to use in case CustomerKey is enabled.		String
camel.component.aws2-s3.customer-key-id	Define the id of Customer key to use in case CustomerKey is enabled.		String
camel.component.aws2-s3.customer-key-m-d5	Define the MD5 of Customer key to use in case CustomerKey is enabled.		String
camel.component.aws2-s3.delete-after-read	Delete objects from S3 after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the AWS2S3Constants#BUCKET_NAME and AWS2S3Constants#KEY headers, or only the AWS2S3Constants#KEY header.	true	Boolean

Name	Description	Default	Type
camel.component.aws2-s3.delete-after-write	Delete file object after the S3 file has been uploaded.	false	Boolean
camel.component.aws2-s3.delimiter	The delimiter which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
camel.component.aws2-s3.destination-bucket	Define the destination bucket where an object must be moved when moveAfterRead is set to true.		String
camel.component.aws2-s3.destination-bucket-prefix	Define the destination bucket prefix to use when an object must be moved and moveAfterRead is set to true.		String
camel.component.aws2-s3.destination-bucket-suffix	Define the destination bucket suffix to use when an object must be moved and moveAfterRead is set to true.		String
camel.component.aws2-s3.done-file-name	If provided, Camel will only consume files if a done file exists.		String
camel.component.aws2-s3.enabled	Whether to enable auto configuration of the aws2-s3 component. This is enabled by default.		Boolean
camel.component.aws2-s3.file-name	To get the object from the bucket with the given file name.		String
camel.component.aws2-s3.ignore-body	If it is true, the S3 Object Body will be ignored completely, if it is set to false the S3 Object will be put in the body. Setting this to true, will override any behavior defined by includeBody option.	false	Boolean

Name	Description	Default	Type
camel.component.aws2-s3.include-body	If it is true, the S3Object exchange will be consumed and put into the body and closed. If false the S3Object stream will be put raw into the body and the headers will be set with the S3 object metadata. This option is strongly related to autocloseBody option. In case of setting includeBody to true because the S3Object stream will be consumed then it will also be closed, while in case of includeBody false then it will be up to the caller to close the S3Object stream. However setting autocloseBody to true when includeBody is false it will schedule to close the S3Object stream automatically on exchange completion.	true	Boolean
camel.component.aws2-s3.include-folders	If it is true, the folders/directories will be consumed. If it is false, they will be ignored, and Exchanges will not be created for those.	true	Boolean
camel.component.aws2-s3.key-name	Setting the key name for an element in the bucket through endpoint parameter.		String
camel.component.aws2-s3.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-s3.move-after-read	Move objects from S3 bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	Boolean
camel.component.aws2-s3.multi-part-upload	If it is true, camel will upload the file with multi part format, the part size is decided by the option of partSize.	false	Boolean
camel.component.aws2-s3.naming-strategy	The naming strategy to use in streaming upload mode.		AWSS3NamingStrategyEnum

Name	Description	Default	Type
camel.component.aws2-s3.operation	The operation to do in case the user don't want to do only an upload.		AWS2S3Operations
camel.component.aws2-s3.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-s3.part-size	Setup the partSize which is used in multi part upload, the default size is 25M.	26214400	Long
camel.component.aws2-s3.pojo-request	If we want to use a POJO request as body or not.	false	Boolean
camel.component.aws2-s3.policy	The policy for this queue to set in the com.amazonaws.services.s3.AmazonS3#setBucketPolicy() method.		String
camel.component.aws2-s3.prefix	The prefix which is used in the com.amazonaws.services.s3.model.ListObjectsRequest to only consume objects we are interested in.		String
camel.component.aws2-s3.proxy-host	To define a proxy host when instantiating the SQS client.		String
camel.component.aws2-s3.proxy-port	Specify a proxy port to be used inside the client definition.		Integer
camel.component.aws2-s3.proxy-protocol	To define a proxy protocol when instantiating the S3 client.		Protocol
camel.component.aws2-s3.region	The region in which S3 client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-s3.restarting-policy	The restarting policy to use in streaming upload mode.		AWSS3RestartingPolicyEnum

Name	Description	Default	Type
camel.component.aws2-s3.secret-key	Amazon AWS Secret Key.		String
camel.component.aws2-s3.storage-class	The storage class to set in the com.amazonaws.services.s3.model.PutObjectRequest request.		String
camel.component.aws2-s3.streaming-upload-mode	When stream mode is true the upload to bucket will be done in streaming.	false	Boolean
camel.component.aws2-s3.streaming-upload-timeout	While streaming upload mode is true, this option set the timeout to complete upload.		Long
camel.component.aws2-s3.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-s3.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-s3.use-aws-k-m-s	Define if KMS must be used or not.	false	Boolean
camel.component.aws2-s3.use-customer-key	Define if Customer Key must be used or not.	false	Boolean
camel.component.aws2-s3.use-default-credentials-provider	Set whether the S3 client should expect to load credentials through a default credentials provider or to expect static credentials to be passed in.	false	Boolean

CHAPTER 7. AWS SIMPLE NOTIFICATION SYSTEM (SNS)

Only producer is supported

The AWS2 SNS component allows messages to be sent to an Amazon Simple Notification Topic. The implementation of the Amazon API is provided by the AWS SDK.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon SNS. More information is available at Amazon SNS.

7.1. URI FORMAT

```
aws2-sns://topicNameOrArn[?options]
```

The topic will be created if they don't already exists. You can append query options to the URI in the following format, **?options=value&option2=value&...**

7.2. URI OPTIONS

7.2.1. Configuring Options

Camel components are configured on two separate levels:

- component level
- endpoint level

7.2.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

7.2.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

7.3. COMPONENT OPTIONS

The AWS Simple Notification System (SNS) component supports 24 options, which are listed below.

Name	Description	Default	Type
amazonSNSClient (producer)	**Autowired** To use the AmazonSNS as the client.		SnsClient
autoCreateTopic (producer)	Setting the autocreation of the topic.	false	boolean
configuration (producer)	Component configuration.		Sns2Configuration
kmsMasterKeyId (producer)	The ID of an AWS-managed customer master key (CMK) for Amazon SNS or a custom CMK.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
messageDeduplicationIdStrategy (producer)	Only for FIFO Topic. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message. Enum values: • useExchangeId • useContentBasedDeduplication	useExchangeId	String

Name	Description	Default	Type
messageGroupIdStrategy (producer)	Only for FIFO Topic. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used. Enum values: • useConstant • useExchangeId • usePropertyValue		String
messageStructure (producer)	The message structure to use such as json.		String
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
policy (producer)	The policy for this topic. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
proxyHost (producer)	To define a proxy host when instantiating the SNS client.		String
proxyPort (producer)	To define a proxy port when instantiating the SNS client.		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the SNS client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
queueUrl (producer)	The queueUrl to subscribe to.		String

Name	Description	Default	Type
region (producer)	The region in which SNS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
serverSideEncryptionEnabled (producer)	Define if Server Side Encryption is enabled or not on the topic.	false	boolean
subject (producer)	The subject which is used if the message header 'CamelAwsSnsSubject' is not present.		String
subscribeSNStoSQS (producer)	Define if the subscription between SNS Topic and SQS must be done or not.	false	boolean
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the SNS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

7.4. ENDPOINT OPTIONS

The AWS Simple Notification System (SNS) endpoint is configured using URI syntax:

```
aws2-sns:topicNameOrArn
```

with the following path and query parameters:

7.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
topicNameOrArn (producer)	**Required** Topic name or ARN.		String

7.4.2. Query Parameters (23 parameters)

Name	Description	Default	Type
amazonSNSClient (producer)	**Autowired** To use the AmazonSNS as the client.		SnsClient
autoCreateTopic (producer)	Setting the autocreation of the topic.	false	boolean
headerFilterStrategy (producer)	To use a custom HeaderFilterStrategy to map headers to/from Camel.		HeaderFilterStrategy
kmsMasterKeyId (producer)	The ID of an AWS-managed customer master key (CMK) for Amazon SNS or a custom CMK.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
messageDeduplicationIdStrategy (producer)	Only for FIFO Topic. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message. Enum values: • useExchangeId • useContentBasedDeduplication	useExchangeId	String

Name	Description	Default	Type
messageGroupIdStrategy (producer)	Only for FIFO Topic. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used. Enum values: • useConstant • useExchangeId • usePropertyValue		String
messageStructure (producer)	The message structure to use such as json.		String
overrideEndpoint (producer)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
policy (producer)	The policy for this topic. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
proxyHost (producer)	To define a proxy host when instantiating the SNS client.		String
proxyPort (producer)	To define a proxy port when instantiating the SNS client.		Integer
proxyProtocol (producer)	To define a proxy protocol when instantiating the SNS client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
queueUrl (producer)	The queueUrl to subscribe to.		String

Name	Description	Default	Type
region (producer)	The region in which SNS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
serverSideEncryptionEnabled (producer)	Define if Server Side Encryption is enabled or not on the topic.	false	boolean
subject (producer)	The subject which is used if the message header 'CamelAwsSnsSubject' is not present.		String
subscribeSNStoSQS (producer)	Define if the subscription between SNS Topic and SQS must be done or not.	false	boolean
trustAllCertificates (producer)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (producer)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (producer)	Set whether the SNS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required SNS component options

You have to provide the amazonSNSClient in the Registry or your accessKey and secretKey to access the Amazon's SNS.

7.5. USAGE

7.5.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the useDefaultCredentialsProvider option and set it to true.

- Java system properties - aws.accessKeyId and aws.secretKey
- Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.

- Web Identity Token from AWS STS.
- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation.

7.5.2. Message headers evaluated by the SNS producer

Header	Type	Description
CamelAwsSnsSubject	**String**	The Amazon SNS message subject. If not set, the subject from the **SnsConfiguration** is used.

7.5.3. Message headers set by the SNS producer

Header	Type	Description
CamelAwsSnsMessageId	**String**	The Amazon SNS message ID.

7.5.4. Advanced AmazonSNS configuration

If you need more control over the **SnsClient** instance configuration you can create your own instance and refer to it from the URI:

```
from("direct:start")
.to("aws2-sns://MyTopic?amazonSNSClient=#client");
```

The **#client** refers to a **AmazonSNS** in the Registry.

7.5.5. Create a subscription between an AWS SNS Topic and an AWS SQS Queue

You can create a subscription of an SQS Queue to an SNS Topic in this way:

```
from("direct:start")
.to("aws2-sns://test-camel-sns1?
amazonSNSClient=#amazonSNSClient&subscribeSNStoSQS=true&queueUrl=https://sqs.eu-central-
1.amazonaws.com/780410022472/test-camel");
```

The **#amazonSNSClient** refers to a **SnsClient** in the Registry. By specifying **subscribeSNStoSQS** to true and a **queueUrl** of an existing SQS Queue, you'll be able to subscribe your SQS Queue to your SNS Topic.

At this point you can consume messages coming from SNS Topic through your SQS Queue

```
from("aws2-sqs://test-camel?
amazonSQSClient=#amazonSQSClient&delay=50&maxMessagesPerPoll=5")
    .to(...);
```

7.6. TOPIC AUTOCREATION

With the option **autoCreateTopic** users are able to avoid the autocreation of an SNS Topic in case it doesn't exist. The default for this option is **true**. If set to false any operation on a not-existent topic in AWS won't be successful and an error will be returned.

7.7. SNS FIFO

SNS FIFO are supported. While creating the SQS queue you will subscribe to the SNS topic there is an important point to remember, you'll need to make possible for the SNS Topic to send message to the SQS Queue.

Example

Suppose you created an SNS FIFO Topic called **Order.fifo** and an SQS Queue called **QueueSub.fifo**.

In the access Policy of the **QueueSub.fifo** you should submit something like this:

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::780560123482:root"
      },
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:eu-west-1:780560123482:QueueSub.fifo"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:eu-west-1:780560123482:QueueSub.fifo",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:eu-west-1:780410022472:Order.fifo"
        }
      }
    }
  ]
}
```

This is a critical step to make the subscription work correctly.

7.7.1. SNS Fifo Topic Message group Id Strategy and message Deduplication Id Strategy

When sending something to the FIFO topic you'll need to always set up a message group Id strategy.

If the content-based message deduplication has been enabled on the SNS Fifo topic, where won't be the need of setting a message deduplication id strategy, otherwise you'll have to set it.

7.8. EXAMPLES

7.8.1. Producer Examples

Sending to a topic

```
from("direct:start")
  .to("aws2-sns://camel-topic?subject=The+subject+message&autoCreateTopic=true");
```

7.9. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-sns</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

7.10. SPRING BOOT AUTO-CONFIGURATION

When using aws2-sns with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-sns-starter</artifactId>
</dependency>
```

The component supports 25 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-sns.access-key	Amazon AWS Access Key.		String

Name	Description	Default	Type
camel.component.aws2-sns.amazon-s-n-s-client	To use the AmazonSNS as the client. The option is a software.amazon.awssdk.services.sns.SnsClient type.		SnsClient
camel.component.aws2-sns.auto-create-topic	Setting the autocreation of the topic.	false	Boolean
camel.component.aws2-sns.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-sns.configuration	Component configuration. The option is a org.apache.camel.component.aws2.sns.Sns2Configuration type.		Sns2Configuration
camel.component.aws2-sns.enabled	Whether to enable auto configuration of the aws2-sns component. This is enabled by default.		Boolean
camel.component.aws2-sns.kms-master-key-id	The ID of an AWS-managed customer master key (CMK) for Amazon SNS or a custom CMK.		String
camel.component.aws2-sns.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-sns.message-deduplication-id-strategy	Only for FIFO Topic. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message.	useExchangeId	String

Name	Description	Default	Type
camel.component.aws2-sns.message-group-id-strategy	Only for FIFO Topic. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used.		String
camel.component.aws2-sns.message-structure	The message structure to use such as json.		String
camel.component.aws2-sns.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-sns.policy	The policy for this topic. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
camel.component.aws2-sns.proxy-host	To define a proxy host when instantiating the SNS client.		String
camel.component.aws2-sns.proxy-port	To define a proxy port when instantiating the SNS client.		Integer
camel.component.aws2-sns.proxy-protocol	To define a proxy protocol when instantiating the SNS client.		Protocol
camel.component.aws2-sns.queue-url	The queueUrl to subscribe to.		String
camel.component.aws2-sns.region	The region in which SNS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-sns.secret-key	Amazon AWS Secret Key.		String

Name	Description	Default	Type
camel.component.aws2-sns.server-side-encryption-enabled	Define if Server Side Encryption is enabled or not on the topic.	false	Boolean
camel.component.aws2-sns.subject	The subject which is used if the message header 'CamelAwsSnsSubject' is not present.		String
camel.component.aws2-sns.subscribe-s-n-sto-s-q-s	Define if the subscription between SNS Topic and SQS must be done or not.	false	Boolean
camel.component.aws2-sns.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-sns.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-sns.use-default-credentials-provider	Set whether the SNS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	Boolean

CHAPTER 8. AWS SIMPLE QUEUE SERVICE (SQS)

Both producer and consumer are supported

The AWS2 SQS component supports sending and receiving messages to Amazon's SQS service.

Prerequisites

You must have a valid Amazon Web Services developer account, and be signed up to use Amazon SQS. More information is available at Amazon SQS.

8.1. URI FORMAT

```
aws2-sqs://queueNameOrArn[?options]
```

The queue will be created if they don't already exists. You can append query options to the URI in the following format,

?options=value&option2=value&...

8.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

8.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

8.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

8.3. COMPONENT OPTIONS

The AWS Simple Queue Service (SQS) component supports 43 options, which are listed below.

Name	Description	Default	Type
amazonAWSHost (common)	The hostname of the Amazon AWS cloud.	amazonaws.com	String
amazonSQSClient (common)	**Autowired** To use the AmazonSQS as client.		SqsClient
autoCreateQueue (common)	Setting the autocreation of the queue.	false	boolean
configuration (common)	The AWS SQS default configuration.		Sqs2Configuration
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
protocol (common)	The underlying protocol used to communicate with SQS.	https	String
proxyProtocol (common)	To define a proxy protocol when instantiating the SQS client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
queueOwnerAWSAccountId (common)	Specify the queue owner aws account id when you need to connect the queue with different account owner.		String
region (common)	The region in which SQS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean

Name	Description	Default	Type
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the SQS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	boolean
attributeNames (consumer)	A list of attribute names to receive when consuming. Multiple names can be separated by comma.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Allows you to use multiple threads to poll the sqs queue to increase throughput.	1	int
defaultVisibilityTimeout (consumer)	The default visibility timeout (in seconds).		Integer
deleteAfterRead (consumer)	Delete message from SQS after it has been read.	true	boolean
deleteIfFiltered (consumer)	Whether or not to send the DeleteMessage to the SQS queue if the exchange has property with key Sqs2Constants#SQS_DELETE_FILTERED (CamelAwsSqsDeleteFiltered) set to true.	true	boolean
extendMessageVisibility (consumer)	If enabled then a scheduled background task will keep extending the message visibility on SQS. This is needed if it takes a long time to process the message. If set to true defaultVisibilityTimeout must be set.	false	boolean
kmsDataKeyReusePeriodSeconds (consumer)	The length of time, in seconds, for which Amazon SQS can reuse a data key to encrypt or decrypt messages before calling AWS KMS again. An integer representing seconds, between 60 seconds (1 minute) and 86,400 seconds (24 hours). Default: 300 (5 minutes).		Integer

Name	Description	Default	Type
kmsMasterKeyId (consumer)	The ID of an AWS-managed customer master key (CMK) for Amazon SQS or a custom CMK.		String
messageAttributeNames (consumer)	A list of message attribute names to receive when consuming. Multiple names can be separated by comma.		String
serverSideEncryptionEnabled (consumer)	Define if Server Side Encryption is enabled or not on the queue.	false	boolean
visibilityTimeout (consumer)	The duration (in seconds) that the received messages are hidden from subsequent retrieve requests after being retrieved by a ReceiveMessage request to set in the com.amazonaws.services.sqs.model.SetQueueAttributesRequest. This only make sense if its different from defaultVisibilityTimeout. It changes the queue visibility timeout attribute permanently.		Integer
waitTimeSeconds (consumer)	Duration in seconds (0 to 20) that the ReceiveMessage action call will wait until a message is in the queue to include in the response.		Integer
batchSeparator (producer)	Set the separator when passing a String to send batch message operation.	,	String
delaySeconds (producer)	Delay sending messages for a number of seconds.		Integer
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
messageDeduplicationIdStrategy (producer)	Only for FIFO queues. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message. Enum values: • useExchangeId • useContentBasedDeduplication	useExchangeId	String
messageGroupIdStrategy (producer)	Only for FIFO queues. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used. Enum values: • useConstant • useExchangeId • usePropertyValue		String
operation (producer)	The operation to do in case the user don't want to send only a message. Enum values: • sendBatchMessage • deleteMessage • listQueues • purgeQueue • deleteQueue		Sqs2Operations
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
delayQueue (advanced)	Define if you want to apply delaySeconds option to the queue or on single messages.	false	boolean
queueUrl (advanced)	To define the queueUrl explicitly. All other parameters, which would influence the queueUrl, are ignored. This parameter is intended to be used, to connect to a mock implementation of SQS, for testing purposes.		String
proxyHost (proxy)	To define a proxy host when instantiating the SQS client.		String
proxyPort (proxy)	To define a proxy port when instantiating the SQS client.		Integer
maximumMessageSize (queue)	The maximumMessageSize (in bytes) an SQS message can contain for this queue.		Integer
messageRetentionPeriod (queue)	The messageRetentionPeriod (in seconds) a message will be retained by SQS for this queue.		Integer
policy (queue)	The policy for this queue. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
receiveMessageWaitTimeSeconds (queue)	If you do not specify WaitTimeSeconds in the request, the queue attribute ReceiveMessageWaitTimeSeconds is used to determine how long to wait.		Integer
redrivePolicy (queue)	Specify the policy that send message to DeadLetter queue. See detail at Amazon docs.		String
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

8.4. ENDPOINT OPTIONS

The AWS Simple Queue Service (SQS) endpoint is configured using URI syntax:

```
aws2-sqs:queueNameOrArn
```

with the following path and query parameters:

8.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
queueNameOrArn (common)	**Required** Queue name or ARN.		String

8.4.2. Query Parameters (61 parameters)

Name	Description	Default	Type
amazonAWSHost (common)	The hostname of the Amazon AWS cloud.	amazonaws.com	String
amazonSQSClient (common)	**Autowired** To use the AmazonSQS as client.		SqsClient
autoCreateQueue (common)	Setting the autocreation of the queue.	false	boolean
headerFilterStrategy (common)	To use a custom HeaderFilterStrategy to map headers to/from Camel.		HeaderFilterStrategy
overrideEndpoint (common)	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	boolean
protocol (common)	The underlying protocol used to communicate with SQS.	https	String
proxyProtocol (common)	To define a proxy protocol when instantiating the SQS client. Enum values: • HTTP • HTTPS	HTTPS	Protocol
queueOwnerAWSAccountId (common)	Specify the queue owner aws account id when you need to connect the queue with different account owner.		String

Name	Description	Default	Type
region (common)	The region in which SQS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
trustAllCertificates (common)	If we want to trust all certificates in case of overriding the endpoint.	false	boolean
uriEndpointOverride (common)	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
useDefaultCredentialsProvider (common)	Set whether the SQS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	boolean
attributeNames (consumer)	A list of attribute names to receive when consuming. Multiple names can be separated by comma.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Allows you to use multiple threads to poll the sqs queue to increase throughput.	1	int
defaultVisibilityTimeout (consumer)	The default visibility timeout (in seconds).		Integer
deleteAfterRead (consumer)	Delete message from SQS after it has been read.	true	boolean
deleteIfFiltered (consumer)	Whether or not to send the DeleteMessage to the SQS queue if the exchange has property with key Sqs2Constants#SQS_DELETE_FILTERED (CamelAwsSqsDeleteFiltered) set to true.	true	boolean

Name	Description	Default	Type
extendMessageVisibility (consumer)	If enabled then a scheduled background task will keep extending the message visibility on SQS. This is needed if it takes a long time to process the message. If set to true defaultVisibilityTimeout must be set. See details at Amazon docs.	false	boolean
kmsDataKeyReusePeriodSeconds (consumer)	The length of time, in seconds, for which Amazon SQS can reuse a data key to encrypt or decrypt messages before calling AWS KMS again. An integer representing seconds, between 60 seconds (1 minute) and 86,400 seconds (24 hours). Default: 300 (5 minutes).		Integer
kmsMasterKeyId (consumer)	The ID of an AWS-managed customer master key (CMK) for Amazon SQS or a custom CMK.		String
maxMessagesPerPoll (consumer)	Gets the maximum number of messages as a limit to poll at each polling. Is default unlimited, but use 0 or negative number to disable it as unlimited.		int
messageAttributeNames (consumer)	A list of message attribute names to receive when consuming. Multiple names can be separated by comma.		String
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
serverSideEncryptionEnabled (consumer)	Define if Server Side Encryption is enabled or not on the queue.	false	boolean
visibilityTimeout (consumer)	The duration (in seconds) that the received messages are hidden from subsequent retrieve requests after being retrieved by a ReceiveMessage request to set in the com.amazonaws.services.sqs.model.SetQueueAttributesRequest. This only make sense if its different from defaultVisibilityTimeout. It changes the queue visibility timeout attribute permanently.		Integer
waitTimeSeconds (consumer)	Duration in seconds (0 to 20) that the ReceiveMessage action call will wait until a message is in the queue to include in the response.		Integer

Name	Description	Default	Type
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
batchSeparator (producer)	Set the separator when passing a String to send batch message operation.	,	String
delaySeconds (producer)	Delay sending messages for a number of seconds.		Integer
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
messageDeduplicationIdStrategy (producer)	Only for FIFO queues. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message. Enum values: • useExchangeId • useContentBasedDeduplication	useExchangeId	String
messageGroupIdStrategy (producer)	Only for FIFO queues. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used. Enum values: • useConstant • useExchangeId • usePropertyValue		String
operation (producer)	The operation to do in case the user don't want to send only a message. Enum values: • sendBatchMessage • deleteMessage • listQueues • purgeQueue • deleteQueue		Sqs2Operations
delayQueue (advanced)	Define if you want to apply delaySeconds option to the queue or on single messages.	false	boolean
queueUrl (advanced)	To define the queueUrl explicitly. All other parameters, which would influence the queueUrl, are ignored. This parameter is intended to be used, to connect to a mock implementation of SQS, for testing purposes.		String

Name	Description	Default	Type
proxyHost (proxy)	To define a proxy host when instantiating the SQS client.		String
proxyPort (proxy)	To define a proxy port when instantiating the SQS client.		Integer
maximumMessageSize (queue)	The maximumMessageSize (in bytes) an SQS message can contain for this queue.		Integer
messageRetentionPeriod (queue)	The messageRetentionPeriod (in seconds) a message will be retained by SQS for this queue.		Integer
policy (queue)	The policy for this queue. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
receiveMessageWaitTimeSeconds (queue)	If you do not specify WaitTimeSeconds in the request, the queue attribute ReceiveMessageWaitTimeSeconds is used to determine how long to wait.		Integer
redrivePolicy (queue)	Specify the policy that send message to DeadLetter queue. See detail at Amazon docs.		String
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int

Name	Description	Default	Type
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean

Name	Description	Default	Type
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Amazon AWS Access Key.		String
secretKey (security)	Amazon AWS Secret Key.		String

Required SQS component options

You have to provide the amazonSQSClient in the Registry or your accessKey and secretKey to access the Amazon's SQS.

8.5. BATCH CONSUMER

This component implements the Batch Consumer.

This allows you for instance to know how many messages exists in this batch and for instance let the Aggregator aggregate this number of messages.

8.6. USAGE

8.6.1. Static credentials vs Default Credential Provider

You have the possibility of avoiding the usage of explicit static credentials, by specifying the **useDefaultCredentialsProvider** option and set it to true.

- Java system properties - **aws.accessKeyId** and **aws.secretKey**
- Environment variables - **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY**.
- Web Identity Token from AWS STS.

- The shared credentials and config files.
- Amazon ECS container credentials - loaded from the Amazon ECS if the environment variable **AWS_CONTAINER_CREDENTIALS_RELATIVE_URI** is set.
- Amazon EC2 Instance profile credentials.

For more information about this you can look at AWS credentials documentation

8.6.2. Message headers set by the SQS producer

Header	Type	Description
CamelAwsSqsMD5OfBody	**String**	The MD5 checksum of the Amazon SQS message.
CamelAwsSqsMessageId	**String**	The Amazon SQS message ID.
CamelAwsSqsDelaySeconds	**Integer**	The delay seconds that the Amazon SQS message can be see by others.

8.6.3. Message headers set by the SQS consumer

Header	Type	Description
CamelAwsSqsMD5OfBody	**String**	The MD5 checksum of the Amazon SQS message.
CamelAwsSqsMessageId	**String**	The Amazon SQS message ID.
CamelAwsSqsReceiptHandle	**String**	The Amazon SQS message receipt handle.
CamelAwsSqsMessageAttributes	**Map<String, String>**	The Amazon SQS message attributes.

8.6.4. Advanced AmazonSQS configuration

If your Camel Application is running behind a firewall or if you need to have more control over the **SqsClient** instance configuration, you can create your own instance:

```
from("aws2-sqs://MyQueue?amazonSQSClient=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

8.6.5. Creating or updating an SQS Queue

In the SQS Component, when an endpoint is started, a check is executed to obtain information about the existence of the queue or not. You're able to customize the creation through the **QueueAttributeName** mapping with the **SQSConfiguration** option.

```
from("aws2-sqs://MyQueue?amazonSQSClient=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

In this example if the **MyQueue** queue is not already created on AWS (and the **autoCreateQueue** option is set to true), it will be created with default parameters from the SQS configuration. If it's already up on AWS, the SQS configuration options will be used to override the existent AWS configuration.

8.6.6. DelayQueue VS Delay for Single message

When the option **delayQueue** is set to true, the SQS Queue will be a **DelayQueue** with the **DelaySeconds** option as delay. For more information about **DelayQueue** you can read the AWS SQS documentation. One important information to take into account is the following:

- For standard queues, the per-queue delay setting is not retroactive—changing the setting doesn't affect the delay of messages already in the queue.
- For FIFO queues, the per-queue delay setting is retroactive—changing the setting affects the delay of messages already in the queue.

as stated in the official documentation. If you want to specify a delay on single messages, you can ignore the **delayQueue** option, while you can set this option to true, if you need to add a fixed delay to all messages enqueued.

8.6.7. Server Side Encryption

There is a set of Server Side Encryption attributes for a queue. The related option are **serverSideEncryptionEnabled**, **keyMasterKeyId** and **kmsDataKeyReusePeriod**. The SSE is disabled by default. You need to explicitly set the option to true and set the related parameters as queue attributes.

8.7. JMS-STYLE SELECTORS

SQS does not allow selectors, but you can effectively achieve this by using the Camel Filter EIP and setting an appropriate **visibilityTimeout**. When SQS dispatches a message, it will wait up to the visibility timeout before it will try to dispatch the message to a different consumer unless a **DeleteMessage** is received. By default, Camel will always send the **DeleteMessage** at the end of the route, unless the route ended in failure. To achieve appropriate filtering and not send the **DeleteMessage** even on successful completion of the route, use a Filter:

```
from("aws2-sqs://MyQueue?
amazonSQSClient=#client&defaultVisibilityTimeout=5000&deleteIfFiltered=false&deleteAfterRead=false
)
.filter("${header.login} == true")
  .setProperty(Sqs2Constants.SQS_DELETE_FILTERED, constant(true))
  .to("mock:filter");
```

In the above code, if an exchange doesn't have an appropriate header, it will not make it through the filter AND also not be deleted from the SQS queue. After 5000 milliseconds, the message will become visible to other consumers.

Note we must set the property **Sqs2Constants.SQS_DELETE_FILTERED** to **true** to instruct Camel to send the **DeleteMessage**, if being filtered.

8.8. AVAILABLE PRODUCER OPERATIONS

- single message (default)
- sendBatchMessage
- deleteMessage
- listQueues

8.9. SEND MESSAGE

You can set a **SendMessageBatchRequest** or an **Iterable**

```
from("direct:start")
  .setBody(constant("Camel rocks!"))
  .to("aws2-sqs://camel-1?accessKey=RAW(xxx)&secretKey=RAW(xxx)&region=eu-west-1");
```

8.10. SEND BATCH MESSAGE

You can set a **SendMessageBatchRequest** or an **Iterable**

```
from("direct:start")
  .setHeader(SqsConstants.SQS_OPERATION, constant("sendBatchMessage"))
  .process(new Processor() {
     @Override
     public void process(Exchange exchange) throws Exception {
        Collection c = new ArrayList();
        c.add("team1");
        c.add("team2");
        c.add("team3");
        c.add("team4");
        exchange.getIn().setBody(c);
     }
  })
  .to("aws2-sqs://camel-1?accessKey=RAW(xxx)&secretKey=RAW(xxx)&region=eu-west-1");
```

As result you'll get an exchange containing a **SendMessageBatchResponse** instance, that you can examinate to check what messages were successfull and what not. The id set on each message of the batch will be a Random UUID.

8.11. DELETE SINGLE MESSAGE

Use **deleteMessage** operation to delete a single message. You'll need to set a receipt handle header for the message you want to delete.

```
from("direct:start")
  .setHeader(SqsConstants.SQS_OPERATION, constant("deleteMessage"))
  .setHeader(SqsConstants.RECEIPT_HANDLE, constant("123456"))
  .to("aws2-sqs://camel-1?accessKey=RAW(xxx)&secretKey=RAW(xxx)&region=eu-west-1");
```

As result you'll get an exchange containing a **DeleteMessageResponse** instance, that you can use to check if the message was deleted or not.

8.12. LIST QUEUES

Use **listQueues** operation to list queues.

```
from("direct:start")
  .setHeader(SqsConstants.SQS_OPERATION, constant("listQueues"))
  .to("aws2-sqs://camel-1?accessKey=RAW(xxx)&secretKey=RAW(xxx)&region=eu-west-1");
```

As result you'll get an exchange containing a **ListQueuesResponse** instance, that you can examinate to check the actual queues.

8.13. PURGE QUEUE

Use **purgeQueue** operation to purge queue.

```
from("direct:start")
  .setHeader(SqsConstants.SQS_OPERATION, constant("purgeQueue"))
  .to("aws2-sqs://camel-1?accessKey=RAW(xxx)&secretKey=RAW(xxx)&region=eu-west-1");
```

As result you'll get an exchange containing a **PurgeQueueResponse** instance.

8.14. QUEUE AUTOCREATION

With the option **autoCreateQueue** users are able to avoid the autocreation of an SQS Queue in case it doesn't exist. The default for this option is **true**. If set to false any operation on a not-existent queue in AWS won't be successful and an error will be returned.

8.15. SEND BATCH MESSAGE AND MESSAGE DEDUPLICATION STRATEGY

In case you're using a **SendBatchMessage** Operation, you can set two different kind of Message Deduplication Strategy: - useExchangeId - useContentBasedDeduplication

The first one will use a **ExchangeIdMessageDeduplicationIdStrategy**, that will use the Exchange ID as parameter The other one will use a **NullMessageDeduplicationIdStrategy**, that will use the body as deduplication element.

In case of send batch message operation, you'll need to use the **useContentBasedDeduplication** and on the Queue you're pointing you'll need to enable the **content based deduplication** option.

8.16. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-aws2-sqs</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **3.18.3** must be replaced by the actual version of Camel.

8.17. SPRING BOOT AUTO-CONFIGURATION

When using aws2-sqs with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-aws2-sqs-starter</artifactId>
</dependency>
```

The component supports 44 options, which are listed below.

Name	Description	Default	Type
camel.component.aws2-sqs.access-key	Amazon AWS Access Key.		String
camel.component.aws2-sqs.amazon-a-w-s-host	The hostname of the Amazon AWS cloud.	amazonaws.com	String
camel.component.aws2-sqs.amazon-s-q-s-client	To use the AmazonSQS as client. The option is a software.amazon.awssdk.services.sqs.SqsClient type.		SqsClient
camel.component.aws2-sqs.attribute-names	A list of attribute names to receive when consuming. Multiple names can be separated by comma.		String
camel.component.aws2-sqs.auto-create-queue	Setting the autocreation of the queue.	false	Boolean
camel.component.aws2-sqs.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.aws2-sqs.batch-separator	Set the separator when passing a String to send batch message operation.	,	String

Name	Description	Default	Type
camel.component.aws2-sqs.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.aws2-sqs.concurrent-consumers	Allows you to use multiple threads to poll the sqs queue to increase throughput.	1	Integer
camel.component.aws2-sqs.configuration	The AWS SQS default configuration. The option is a org.apache.camel.component.aws2.sqs.Sqs2Configuration type.		Sqs2Configuration
camel.component.aws2-sqs.default-visibility-timeout	The default visibility timeout (in seconds).		Integer
camel.component.aws2-sqs.delay-queue	Define if you want to apply delaySeconds option to the queue or on single messages.	false	Boolean
camel.component.aws2-sqs.delay-seconds	Delay sending messages for a number of seconds.		Integer
camel.component.aws2-sqs.delete-after-read	Delete message from SQS after it has been read.	true	Boolean
camel.component.aws2-sqs.delete-if-filtered	Whether or not to send the DeleteMessage to the SQS queue if the exchange has property with key Sqs2Constants#SQS_DELETE_FILTERED (CamelAwsSqsDeleteFiltered) set to true.	true	Boolean
camel.component.aws2-sqs.enabled	Whether to enable auto configuration of the aws2-sqs component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.aws2-sqs.extend-message-visibility	If enabled then a scheduled background task will keep extending the message visibility on SQS. This is needed if it takes a long time to process the message. If set to true defaultVisibilityTimeout must be set. See details at Amazon docs.	false	Boolean
camel.component.aws2-sqs.kms-data-key-reuse-period-seconds	The length of time, in seconds, for which Amazon SQS can reuse a data key to encrypt or decrypt messages before calling AWS KMS again. An integer representing seconds, between 60 seconds (1 minute) and 86,400 seconds (24 hours). Default: 300 (5 minutes).		Integer
camel.component.aws2-sqs.kms-master-key-id	The ID of an AWS-managed customer master key (CMK) for Amazon SQS or a custom CMK.		String
camel.component.aws2-sqs.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.aws2-sqs.maximum-message-size	The maximumMessageSize (in bytes) an SQS message can contain for this queue.		Integer
camel.component.aws2-sqs.message-attribute-names	A list of message attribute names to receive when consuming. Multiple names can be separated by comma.		String
camel.component.aws2-sqs.message-deduplication-id-strategy	Only for FIFO queues. Strategy for setting the messageDeduplicationId on the message. Can be one of the following options: useExchangeId, useContentBasedDeduplication. For the useContentBasedDeduplication option, no messageDeduplicationId will be set on the message.	useExchangeId	String

Name	Description	Default	Type
camel.component.aws2-sqs.message-group-id-strategy	Only for FIFO queues. Strategy for setting the messageGroupId on the message. Can be one of the following options: useConstant, useExchangeId, usePropertyValue. For the usePropertyValue option, the value of property CamelAwsMessageGroupId will be used.		String
camel.component.aws2-sqs.message-retention-period	The messageRetentionPeriod (in seconds) a message will be retained by SQS for this queue.		Integer
camel.component.aws2-sqs.operation	The operation to do in case the user don't want to send only a message.		Sqs2Operations
camel.component.aws2-sqs.override-endpoint	Set the need for overidding the endpoint. This option needs to be used in combination with uriEndpointOverride option.	false	Boolean
camel.component.aws2-sqs.policy	The policy for this queue. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
camel.component.aws2-sqs.protocol	The underlying protocol used to communicate with SQS.	https	String
camel.component.aws2-sqs.proxy-host	To define a proxy host when instantiating the SQS client.		String
camel.component.aws2-sqs.proxy-port	To define a proxy port when instantiating the SQS client.		Integer
camel.component.aws2-sqs.proxy-protocol	To define a proxy protocol when instantiating the SQS client.		Protocol
camel.component.aws2-sqs.queue-owner-a-w-s-account-id	Specify the queue owner aws account id when you need to connect the queue with different account owner.		String

Name	Description	Default	Type
camel.component.aws2-sqs.queue-url	To define the queueUrl explicitly. All other parameters, which would influence the queueUrl, are ignored. This parameter is intended to be used, to connect to a mock implementation of SQS, for testing purposes.		String
camel.component.aws2-sqs.receive-message-wait-time-seconds	If you do not specify WaitTimeSeconds in the request, the queue attribute ReceiveMessageWaitTimeSeconds is used to determine how long to wait.		Integer
camel.component.aws2-sqs.redrive-policy	Specify the policy that send message to DeadLetter queue. See detail at Amazon docs.		String
camel.component.aws2-sqs.region	The region in which SQS client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1) You'll need to use the name Region.EU_WEST_1.id().		String
camel.component.aws2-sqs.secret-key	Amazon AWS Secret Key.		String
camel.component.aws2-sqs.server-side-encryption-enabled	Define if Server Side Encryption is enabled or not on the queue.	false	Boolean
camel.component.aws2-sqs.trust-all-certificates	If we want to trust all certificates in case of overriding the endpoint.	false	Boolean
camel.component.aws2-sqs.uri-endpoint-override	Set the overriding uri endpoint. This option needs to be used in combination with overrideEndpoint option.		String
camel.component.aws2-sqs.use-default-credentials-provider	Set whether the SQS client should expect to load credentials on an AWS infra instance or to expect static credentials to be passed in.	false	Boolean

Name	Description	Default	Type
camel.component.aws2-sqs.visibility-timeout	The duration (in seconds) that the received messages are hidden from subsequent retrieve requests after being retrieved by a ReceiveMessage request to set in the com.amazonaws.services.sqs.model.SetQueueAttributesRequest. This only make sense if its different from defaultVisibilityTimeout. It changes the queue visibility timeout attribute permanently.		Integer
camel.component.aws2-sqs.wait-time-seconds	Duration in seconds (0 to 20) that the ReceiveMessage action call will wait until a message is in the queue to include in the response.		Integer

CHAPTER 9. AZURE SERVICEBUS

Since Camel 3.12

Both producer and consumer are supported

The azure-servicebus component that integrates Azure ServiceBus. Azure ServiceBus is a fully managed enterprise integration message broker. Service Bus can decouple applications and services. Service Bus offers a reliable and secure platform for asynchronous transfer of data and state. Data is transferred between different applications and services using messages.

Prerequisites

You must have a valid Windows Azure Storage account. More information is available at Azure Documentation Portal.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure-servicebus</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

9.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

9.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

9.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

9.2. COMPONENT OPTIONS

The Azure ServiceBus component supports 25 options, which are listed below.

Name	Description	Default	Type
amqpRetryOptions (common)	Sets the retry options for Service Bus clients. If not specified, the default retry options are used.		AmqpRetryOptions
amqpTransportType (common)	Sets the transport type by which all the communication with Azure Service Bus occurs. Default value is AmqpTransportType#AMQP. Enum values: • Amqp • AmqpWebSockets	AMQP	AmqpTransportType
clientOptions (common)	Sets the ClientOptions to be sent from the client built from this builder, enabling customization of certain properties, as well as support the addition of custom header information. Refer to the ClientOptions documentation for more information.		ClientOptions
configuration (common)	The component configurations.		ServiceBusConfiguration
proxyOptions (common)	Sets the proxy configuration to use for ServiceBusSenderAsyncClient. When a proxy is configured, AmqpTransportType#AMQP_WEB_SOCKETS must be used for the transport type.		ProxyOptions
serviceBusType (common)	**Required** The service bus type of connection to execute. Queue is for typical queue option and topic for subscription based model. Enum values: • queue • topic	queue	ServiceBusType

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
consumerOperation (consumer)	Sets the desired operation to be used in the consumer. Enum values: ● receiveMessages ● peekMessages	receiveMessages	ServiceBusConsumerOperationDefinition
disableAutoComplete (consumer)	Disables auto-complete and auto-abandon of received messages. By default, a successfully processed message is \\{link ServiceBusReceiverAsyncClient#complete(ServiceBusReceivedMessage) completed}. If an error happens when the message is processed, it is \\{link ServiceBusReceiverAsyncClient#abandon(ServiceBusReceivedMessage) abandoned}.	false	boolean
maxAutoLockRenewDuration (consumer)	Sets the amount of time to continue auto-renewing the lock. Setting Duration#ZERO or null disables auto-renewal. For \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} mode, auto-renewal is disabled.	5m	Duration
peekNumMaxMessages (consumer)	Set the max number of messages to be peeked during the peek operation.		Integer

Name	Description	Default	Type
prefetchCount (consumer)	Sets the prefetch count of the receiver. For both \\{link ServiceBusReceiveMode#PEEK_LOCK PEEK_LOCK} and \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} modes the default value is 1. Prefetch speeds up the message flow by aiming to have a message readily available for local retrieval when and before the application asks for one using ServiceBusReceiverAsyncClient#receiveMessages(). Setting a non-zero value will prefetch that number of messages. Setting the value to zero turns prefetch off.		int
receiverAsyncClient (consumer)	**Autowired** Sets the receiverAsyncClient in order to consume messages by the consumer.		ServiceBusReceiverAsyncClient
serviceBusReceiveMode (consumer)	Sets the receive mode for the receiver. Enum values: ● PEEK_LOCK ● RECEIVE_AND_DELETE	PEEK_LOCK	ServiceBusReceiveMode
subQueue (consumer)	Sets the type of the SubQueue to connect to. Enum values: ● NONE ● DEAD_LETTER_QUEUE ● TRANSFER_DEAD_LETTER_QUEUE		SubQueue
subscriptionName (consumer)	Sets the name of the subscription in the topic to listen to. topicOrQueueName and serviceBusType=topic must also be set. This property is required if serviceBusType=topic and the consumer is in use.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
producerOperation (producer)	Sets the desired operation to be used in the producer. Enum values: • sendMessages • scheduleMessages	sendMessages	ServiceBusProducerOperationDefinition
scheduledEnqueueTime (producer)	Sets OffsetDateTime at which the message should appear in the Service Bus queue or topic.		OffsetDateTime
senderAsyncClient (producer)	**Autowired** Sets SenderAsyncClient to be used in the producer.		ServiceBusSenderAsyncClient
serviceBusTransactionContext (producer)	Represents transaction in service. This object just contains transaction id.		ServiceBusTransactionContext
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
connectionString (security)	Sets the connection string for a Service Bus namespace or a specific Service Bus resource.		String
fullyQualifiedNamespace (security)	Fully Qualified Namespace of the service bus.		String
tokenCredential (security)	A TokenCredential for Azure AD authentication, implemented in com.azure.identity.		TokenCredential

9.3. ENDPOINT OPTIONS

The Azure ServiceBus endpoint is configured using URI syntax:

```
azure-servicebus:topicOrQueueName
```

with the following path and query parameters:

9.3.1. Path Parameters (1 parameters)

Name	Description	Default	Type
topicOrQueueName (common)	Selected topic name or the queue name, that is depending on serviceBusType config. For example if serviceBusType=queue, then this will be the queue name and if serviceBusType=topic, this will be the topic name.		String

9.3.2. Query Parameters (25 parameters)

Name	Description	Default	Type
amqpRetryOptions (common)	Sets the retry options for Service Bus clients. If not specified, the default retry options are used.		AmqpRetryOptions
amqpTransportType (common)	Sets the transport type by which all the communication with Azure Service Bus occurs. Default value is AmqpTransportType#AMQP. Enum values: • Amqp • AmqpWebSockets	AMQP	AmqpTransportType
clientOptions (common)	Sets the ClientOptions to be sent from the client built from this builder, enabling customization of certain properties, as well as support the addition of custom header information. Refer to the ClientOptions documentation for more information.		ClientOptions
proxyOptions (common)	Sets the proxy configuration to use for ServiceBusSenderAsyncClient. When a proxy is configured, AmqpTransportType#AMQP_WEB_SOCKETS must be used for the transport type.		ProxyOptions

Name	Description	Default	Type
serviceBusType (common)	**Required** The service bus type of connection to execute. Queue is for typical queue option and topic for subscription based model. Enum values: • queue • topic	queue	ServiceBusType
consumerOperation (consumer)	Sets the desired operation to be used in the consumer. Enum values: • receiveMessages • peekMessages	receiveMessages	ServiceBusConsumerOperationDefinition
disableAutoComplete (consumer)	Disables auto-complete and auto-abandon of received messages. By default, a successfully processed message is \\{link ServiceBusReceiverAsyncClient#complete(ServiceBusReceivedMessage) completed}. If an error happens when the message is processed, it is \\{link ServiceBusReceiverAsyncClient#abandon(ServiceBusReceivedMessage) abandoned}.	false	boolean
maxAutoLockRenewDuration (consumer)	Sets the amount of time to continue auto-renewing the lock. Setting Duration#ZERO or null disables auto-renewal. For \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} mode, auto-renewal is disabled.	5m	Duration
peekNumMaxMessages (consumer)	Set the max number of messages to be peeked during the peek operation.		Integer

Name	Description	Default	Type
prefetchCount (consumer)	Sets the prefetch count of the receiver. For both \\{link ServiceBusReceiveMode#PEEK_LOCK PEEK_LOCK} and \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} modes the default value is 1. Prefetch speeds up the message flow by aiming to have a message readily available for local retrieval when and before the application asks for one using ServiceBusReceiverAsyncClient#receiveMessages(). Setting a non-zero value will prefetch that number of messages. Setting the value to zero turns prefetch off.		int
receiverAsyncClient (consumer)	**Autowired** Sets the receiverAsyncClient in order to consume messages by the consumer.		ServiceBusReceiverAsyncClient
serviceBusReceiveMode (consumer)	Sets the receive mode for the receiver. Enum values: • PEEK_LOCK • RECEIVE_AND_DELETE	PEEK_LOCK	ServiceBusReceiveMode
subQueue (consumer)	Sets the type of the SubQueue to connect to. Enum values: • NONE • DEAD_LETTER_QUEUE • TRANSFER_DEAD_LETTER_QUEUE		SubQueue
subscriptionName (consumer)	Sets the name of the subscription in the topic to listen to. topicOrQueueName and serviceBusType=topic must also be set. This property is required if serviceBusType=topic and the consumer is in use.		String
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
producerOperation (producer)	Sets the desired operation to be used in the producer. Enum values: • sendMessages • scheduleMessages	sendMessages	ServiceBusProducerOperationDefinition
scheduledEnqueueTime (producer)	Sets OffsetDateTime at which the message should appear in the Service Bus queue or topic.		OffsetDateTime
senderAsyncClient (producer)	**Autowired** Sets SenderAsyncClient to be used in the producer.		ServiceBusSenderAsyncClient
serviceBusTransactionContext (producer)	Represents transaction in service. This object just contains transaction id.		ServiceBusTransactionContext
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
connectionString (security)	Sets the connection string for a Service Bus namespace or a specific Service Bus resource.		String

Name	Description	Default	Type
fullyQualifiedNamespace (security)	Fully Qualified Namespace of the service bus.		String
tokenCredential (security)	A TokenCredential for Azure AD authentication, implemented in com.azure.identity.		TokenCredential

9.4. ASYNC CONSUMER AND PRODUCER

This component implements the async Consumer and producer. This allows camel route to consume and produce events asynchronously without blocking any threads.

9.5. MESSAGE HEADERS

The Azure ServiceBus component supports 25 message header(s), which is/are listed below:

Name	Description	Default	Type
CamelAzureServiceBusApplicationProperties (common) Constant: APPLICATION_PROPERTIES	The application properties (also known as custom properties) on messages sent and received by the producer and consumer, respectively.		Map
CamelAzureServiceBusContentType (consumer) Constant: CONTENT_TYPE	Gets the content type of the message.		String
CamelAzureServiceBusCorrelationId (consumer) Constant: CORRELATION_ID	Gets a correlation identifier.		String

Name	Description	Default	Type
CamelAzureServiceBusDeadLetterErrorDescription (consumer) Constant: DEAD_LETTER_ERROR_DESCRIPTION	Gets the description for a message that has been dead-lettered.		String
CamelAzureServiceBusDeadLetterReason (consumer) Constant: DEAD_LETTER_REASON	Gets the reason a message was dead-lettered.		String
CamelAzureServiceBusDeadLetterSource (consumer) Constant: DEAD_LETTER_SOURCE	Gets the name of the queue or subscription that this message was enqueued on, before it was dead-lettered.		String
CamelAzureServiceBusDeliveryCount (consumer) Constant: DELIVERY_COUNT	Gets the number of the times this message was delivered to clients.		long
CamelAzureServiceBusEnqueuedSequenceNumber (consumer) Constant: ENQUEUED_SEQUENCE_NUMBER	Gets the enqueued sequence number assigned to a message by Service Bus.		long

Name	Description	Default	Type
CamelAzureServiceBusEnqueuedTime (consumer) Constant: ENQUEUED_TIME	Gets the datetime at which this message was enqueued in Azure Service Bus.		OffsetDateTime
CamelAzureServiceBusExpiresAt (consumer) Constant: EXPIRES_AT	Gets the datetime at which this message will expire.		OffsetDateTime
CamelAzureServiceBusLockToken (consumer) Constant: LOCK_TOKEN	Gets the lock token for the current message.		String
CamelAzureServiceBusLockedUntil (consumer) Constant: LOCKED_UNTIL	Gets the datetime at which the lock of this message expires.		OffsetDateTime
CamelAzureServiceBusMessageId (consumer) Constant: MESSAGE_ID	Gets the identifier for the message.		String
CamelAzureServiceBusPartitionKey (consumer) Constant: PARTITION_KEY	Gets the partition key for sending a message to a partitioned entity.		String
CamelAzureServiceBusRawAmqpMessage (consumer) Constant: RAW_AMQP_MESSAGE	The representation of message as defined by AMQP protocol.		AmqpAnnotatedMessage

Name	Description	Default	Type
CamelAzureServiceBusReplyTo (consumer) Constant: REPLY_TO	Gets the address of an entity to send replies to.		String
CamelAzureServiceBusReplyToSessionId (consumer) Constant: REPLY_TO_SESSION_ID	Gets or sets a session identifier augmenting the ReplyTo address.		String
CamelAzureServiceBusSequenceNumber (consumer) Constant: SEQUENCE_NUMBER	Gets the unique number assigned to a message by Service Bus.		long
CamelAzureServiceBusSessionId (consumer) Constant: SESSION_ID	Gets the session id of the message.		String
CamelAzureServiceBusSubject (consumer) Constant: SUBJECT	Gets the subject for the message.		String
CamelAzureServiceBusTimeToLive (consumer) Constant: TIME_TO_LIVE	Gets the duration before this message expires.		Duration
CamelAzureServiceBusTo (consumer) Constant: TO	Gets the to address.		String

Name	Description	Default	Type
CamelAzureServiceBusScheduledEnqueueTime (common) Constant: SCHEDULED_ENQUEUE_TIME	(producer)Overrides the OffsetDateTime at which the message should appear in the Service Bus queue or topic. (consumer) Gets the scheduled enqueue time of this message.		OffsetDateTime
CamelAzureServiceBusServiceBusTransactionContext (producer) Constant: SERVICE_BUS_TRANSACTION_CONTEXT	Overrides the transaction in service. This object just contains transaction id.		ServiceBusTransactionContext
CamelAzureServiceBusProducerOperation (producer) Constant: PRODUCER_OPERATION	Overrides the desired operation to be used in the producer. Enum values: • sendMessages • scheduleMessages		ServiceBusProducerOperationDefinition

9.5.1. Message Body

In the producer, this component accepts message body of **String** type or **List<String>** to send batch messages.

In the consumer, the returned message body will be of type `String.

9.5.2. Azure ServiceBus Producer operations

Operation	Description
sendMessages	Sends a set of messages to a Service Bus queue or topic using a batched approach.
scheduleMessages	Sends a scheduled message to the Azure Service Bus entity this sender is connected to. A scheduled message is enqueued and made available to receivers only at the scheduled enqueue time.

9.5.3. Azure ServiceBus Consumer operations

Operation	Description
receiveMessages	Receives an infinite stream of messages from the Service Bus entity.
peekMessages	Reads the next batch of active messages without changing the state of the receiver or the message source.

9.5.3.1. Examples

- **sendMessages**

```
from("direct:start")
  .process(exchange -> {
        final List<Object> inputBatch = new LinkedList<>();
          inputBatch.add("test batch 1");
          inputBatch.add("test batch 2");
          inputBatch.add("test batch 3");
          inputBatch.add(123456);

          exchange.getIn().setBody(inputBatch);
      })
  .to("azure-servicebus:test//?connectionString=test")
  .to("mock:result");
```

- **scheduleMessages**

```
from("direct:start")
  .process(exchange -> {
        final List<Object> inputBatch = new LinkedList<>();
          inputBatch.add("test batch 1");
          inputBatch.add("test batch 2");
          inputBatch.add("test batch 3");
          inputBatch.add(123456);

          exchange.getIn().setHeader(ServiceBusConstants.SCHEDULED_ENQUEUE_TIME,
OffsetDateTime.now());
          exchange.getIn().setBody(inputBatch);
      })
  .to("azure-servicebus:test//?connectionString=test&producerOperation=scheduleMessages")
  .to("mock:result");
```

- **receiveMessages**

```
from("azure-servicebus:test//?connectionString=test")
  .log("${body}")
  .to("mock:result");
```

- **peekMessages**

```
from("azure-servicebus:test//?
connectionString=test&consumerOperation=peekMessages&peekNumMaxMessages=3")
  .log("${body}")
  .to("mock:result");
```

9.6. SPRING BOOT AUTO-CONFIGURATION

When using azure-servicebus with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-azure-servicebus-starter</artifactId>
</dependency>
```

The component supports 26 options, which are listed below.

Name	Description	Default	Type
camel.component.azure-servicebus.amqp-retry-options	Sets the retry options for Service Bus clients. If not specified, the default retry options are used. The option is a com.azure.core.amqp.AmqpRetryOptions type.		AmqpRetryOptions
camel.component.azure-servicebus.amqp-transport-type	Sets the transport type by which all the communication with Azure Service Bus occurs. Default value is AmqpTransportType#AMQP.		AmqpTransportType
camel.component.azure-servicebus.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.azure-servicebus.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.azure-servicebus.client-options	Sets the ClientOptions to be sent from the client built from this builder, enabling customization of certain properties, as well as support the addition of custom header information. Refer to the ClientOptions documentation for more information. The option is a com.azure.core.util.ClientOptions type.		ClientOptions
camel.component.azure-servicebus.configuration	The component configurations. The option is a org.apache.camel.component.azure.servicebus.ServiceBusConfiguration type.		ServiceBusConfiguration
camel.component.azure-servicebus.connection-string	Sets the connection string for a Service Bus namespace or a specific Service Bus resource.		String
camel.component.azure-servicebus.consumer-operation	Sets the desired operation to be used in the consumer.		ServiceBusConsumerOperationDefinition
camel.component.azure-servicebus.disable-auto-complete	Disables auto-complete and auto-abandon of received messages. By default, a successfully processed message is \\{link ServiceBusReceiverAsyncClient#complete(ServiceBusReceivedMessage) completed}. If an error happens when the message is processed, it is \\{link ServiceBusReceiverAsyncClient#abandon(ServiceBusReceivedMessage) abandoned}.	false	Boolean
camel.component.azure-servicebus.enabled	Whether to enable auto configuration of the azure-servicebus component. This is enabled by default.		Boolean
camel.component.azure-servicebus.fully-qualified-namespace	Fully Qualified Namespace of the service bus.		String

Name	Description	Default	Type
camel.component.azure-servicebus.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.azure-servicebus.max-auto-lock-renew-duration	Sets the amount of time to continue auto-renewing the lock. Setting Duration#ZERO or null disables auto-renewal. For \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} mode, auto-renewal is disabled. The option is a java.time.Duration type.		Duration
camel.component.azure-servicebus.peek-num-max-messages	Set the max number of messages to be peeked during the peek operation.		Integer
camel.component.azure-servicebus.prefetch-count	Sets the prefetch count of the receiver. For both \\{link ServiceBusReceiveMode#PEEK_LOCK PEEK_LOCK} and \\{link ServiceBusReceiveMode#RECEIVE_AND_DELETE RECEIVE_AND_DELETE} modes the default value is 1. Prefetch speeds up the message flow by aiming to have a message readily available for local retrieval when and before the application asks for one using ServiceBusReceiverAsyncClient#receiveMessages(). Setting a non-zero value will prefetch that number of messages. Setting the value to zero turns prefetch off.		Integer
camel.component.azure-servicebus.producer-operation	Sets the desired operation to be used in the producer.		ServiceBusProducerOperationDefinition

Name	Description	Default	Type
camel.component.azure-servicebus.proxy-options	Sets the proxy configuration to use for ServiceBusSenderAsyncClient. When a proxy is configured, AmqpTransportType#AMQP_WEB_SOCKETS must be used for the transport type. The option is a com.azure.core.amqp.ProxyOptions type.		ProxyOptions
camel.component.azure-servicebus.receiver-async-client	Sets the receiverAsyncClient in order to consume messages by the consumer. The option is a com.azure.messaging.servicebus.ServiceBusReceiverAsyncClient type.		ServiceBusReceiverAsyncClient
camel.component.azure-servicebus.scheduled-enqueue-time	Sets OffsetDateTime at which the message should appear in the Service Bus queue or topic. The option is a java.time.OffsetDateTime type.		OffsetDateTime
camel.component.azure-servicebus.sender-async-client	Sets SenderAsyncClient to be used in the producer. The option is a com.azure.messaging.servicebus.ServiceBusSenderAsyncClient type.		ServiceBusSenderAsyncClient
camel.component.azure-servicebus.service-bus-receive-mode	Sets the receive mode for the receiver.		ServiceBusReceiveMode
camel.component.azure-servicebus.service-bus-transaction-context	Represents transaction in service. This object just contains transaction id. The option is a com.azure.messaging.servicebus.ServiceBusTransactionContext type.		ServiceBusTransactionContext
camel.component.azure-servicebus.service-bus-type	The service bus type of connection to execute. Queue is for typical queue option and topic for subscription based model.		ServiceBusType
camel.component.azure-servicebus.sub-queue	Sets the type of the SubQueue to connect to.		SubQueue

Name	Description	Default	Type
camel.component.azure-servicebus.subscription-name	Sets the name of the subscription in the topic to listen to. topicOrQueueName and serviceBusType=topic must also be set. This property is required if serviceBusType=topic and the consumer is in use.		String
camel.component.azure-servicebus.token-credential	A TokenCredential for Azure AD authentication, implemented in com.azure.identity. The option is a com.azure.core.credential.TokenCredential type.		TokenCredential

CHAPTER 10. AZURE STORAGE BLOB SERVICE

Both producer and consumer are supported

The Azure Storage Blob component is used for storing and retrieving blobs from Azure Storage Blob Service using **Azure APIs v12**. However in case of versions above v12, we will see if this component can adopt these changes depending on how much breaking changes can result.

Prerequisites

You must have a valid Windows Azure Storage account. More information is available at Azure Documentation Portal .

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure-storage-blob</artifactId>
  <version>{CamelSBVersion}</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

10.1. URI FORMAT

```
azure-storage-blob://accountName[/containerName][?options]
```

In case of consumer, **accountName**, **containerName** are required. In case of producer, it depends on the operation that being requested, for example if operation is on a container level, for example, **createContainer**, **accountName** and **containerName** are only required, but in case of operation being requested in blob level, for example, **getBlob**, **accountName**, **containerName** and **blobName** are required.

The blob will be created if it does not already exist. You can append query options to the URI in the following format,

?options=value&option2=value&...

10.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

10.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

10.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

10.3. COMPONENT OPTIONS

The Azure Storage Blob Service component supports 31 options, which are listed below.

Name	Description	Default	Type
blobName (common)	The blob name, to consume specific blob from a container. However on producer, is only required for the operations on the blob level.		String
blobOffset (common)	Set the blob offset for the upload or download operations, default is 0.	0	long
blobType (common)	The blob type in order to initiate the appropriate settings for each blob type. Enum values: • blockblob • appendblob • pageblob	blockblob	BlobType
closeStreamAfterRead (common)	Close the stream after read or keep it open, default is true.	true	boolean
configuration (common)	The component configurations.		BlobConfiguration
credentials (common)	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information.		StorageSharedKeyCredential

Name	Description	Default	Type
dataCount (common)	How many bytes to include in the range. Must be greater than or equal to 0 if specified.		Long
fileDir (common)	The file directory where the downloaded blobs will be saved to, this can be used in both, producer and consumer.		String
maxResultsPerPage (common)	Specifies the maximum number of blobs to return, including all BlobPrefix elements. If the request does not specify maxResultsPerPage or specifies a value greater than 5,000, the server will return up to 5,000 items.		Integer
maxRetryRequests (common)	Specifies the maximum number of additional HTTP Get requests that will be made while reading the data from a response body.	0	int
prefix (common)	Filters the results to return only blobs whose names begin with the specified prefix. May be null to return all blobs.		String
regex (common)	Filters the results to return only blobs whose names match the specified regular expression. May be null to return all if both prefix and regex are set, regex takes the priority and prefix is ignored.		String
serviceClient (common)	**Autowired** Client to a storage account. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. It may also be used to construct URLs to blobs and containers. This client contains operations on a service account. Operations on a container are available on BlobContainerClient through BlobServiceClient#getBlobContainerClient(String), and operations on a blob are available on BlobClient through BlobContainerClient#getBlobClient(String).		BlobServiceClient
timeout (common)	An optional timeout value beyond which a RuntimeException will be raised.		Duration

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
blobSequenceNumber (producer)	A user-controlled value that you can use to track requests. The value of the sequence number must be between 0 and 263 - 1.The default value is 0.	0	Long
blockListType (producer)	Specifies which type of blocks to return. Enum values: • committed • uncommitted • all	COMMITTED	BlockListType
changeFeedContext (producer)	When using getChangeFeed producer operation, this gives additional context that is passed through the Http pipeline during the service call.		Context
changeFeedEndTime (producer)	When using getChangeFeed producer operation, this filters the results to return events approximately before the end time. Note: A few events belonging to the next hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the end time up by an hour.		OffsetDateTime
changeFeedStartTime (producer)	When using getChangeFeed producer operation, this filters the results to return events approximately after the start time. Note: A few events belonging to the previous hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the start time down by an hour.		OffsetDateTime
closeStreamAfterWrite (producer)	Close the stream after write or keep it open, default is true.	true	boolean

Name	Description	Default	Type
commitBlockListLater (producer)	When is set to true, the staged blocks will not be committed directly.	true	boolean
createAppendBlob (producer)	When is set to true, the append blocks will be created when committing append blocks.	true	boolean
createPageBlob (producer)	When is set to true, the page blob will be created when uploading page blob.	true	boolean
downloadLinkExpiration (producer)	Override the default expiration (millis) of URL download link.		Long
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	The blob operation that can be used with this component on the producer. Enum values: • listBlobContainers • createBlobContainer • deleteBlobContainer • listBlobs • getBlob • deleteBlob • downloadBlobToFile • downloadLink • uploadBlockBlob • stageBlockBlobList • commitBlobBlockList • getBlobBlockList • createAppendBlob • commitAppendBlob • createPageBlob • uploadPageBlob • resizePageBlob • clearPageBlob • getPageBlobRanges	listBlobContainers	BlobOperationsDefinition
pageBlobSize (producer)	Specifies the maximum size for the page blob, up to 8 TB. The page blob size must be aligned to a 512-byte boundary.	512	Long
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
accessKey (security)	Access key for the associated azure account name to be used for authentication with azure blob services.		String
sourceBlobAccessKey (security)	Source Blob Access Key: for copyblob operation, sadly, we need to have an accessKey for the source blob we want to copy Passing an accessKey as header, it's unsafe so we could set as key.		String

10.4. ENDPOINT OPTIONS

The Azure Storage Blob Service endpoint is configured using URI syntax:

```
azure-storage-blob:accountName/containerName
```

with the following path and query parameters:

10.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
accountName (common)	Azure account name to be used for authentication with azure blob services.		String
containerName (common)	The blob container name.		String

10.4.2. Query Parameters (48 parameters)

Name	Description	Default	Type
blobName (common)	The blob name, to consume specific blob from a container. However on producer, is only required for the operations on the blob level.		String
blobOffset (common)	Set the blob offset for the upload or download operations, default is 0.	0	long

Name	Description	Default	Type
blobServiceClient (common)	Client to a storage account. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. It may also be used to construct URLs to blobs and containers. This client contains operations on a service account. Operations on a container are available on BlobContainerClient through getBlobContainerClient(String), and operations on a blob are available on BlobClient through getBlobContainerClient(String).getBlobClient(String).		BlobServiceClient
blobType (common)	The blob type in order to initiate the appropriate settings for each blob type. Enum values: • blockblob • appendblob • pageblob	blockblob	BlobType
closeStreamAfterRead (common)	Close the stream after read or keep it open, default is true.	true	boolean
credentials (common)	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information.		StorageSharedKeyCredential
dataCount (common)	How many bytes to include in the range. Must be greater than or equal to 0 if specified.		Long
fileDir (common)	The file directory where the downloaded blobs will be saved to, this can be used in both, producer and consumer.		String
maxResultsPerPage (common)	Specifies the maximum number of blobs to return, including all BlobPrefix elements. If the request does not specify maxResultsPerPage or specifies a value greater than 5,000, the server will return up to 5,000 items.		Integer
maxRetryRequests (common)	Specifies the maximum number of additional HTTP Get requests that will be made while reading the data from a response body.	0	int

Name	Description	Default	Type
prefix (common)	Filters the results to return only blobs whose names begin with the specified prefix. May be null to return all blobs.		String
regex (common)	Filters the results to return only blobs whose names match the specified regular expression. May be null to return all if both prefix and regex are set, regex takes the priority and prefix is ignored.		String
serviceClient (common)	**Autowired** Client to a storage account. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. It may also be used to construct URLs to blobs and containers. This client contains operations on a service account. Operations on a container are available on BlobContainerClient through BlobServiceClient#getBlobContainerClient(String), and operations on a blob are available on BlobClient through BlobContainerClient#getBlobClient(String).		BlobServiceClient
timeout (common)	An optional timeout value beyond which a RuntimeException will be raised.		Duration
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
blobSequenceNumber (producer)	A user-controlled value that you can use to track requests. The value of the sequence number must be between 0 and 263 - 1.The default value is 0.	0	Long
blockListType (producer)	Specifies which type of blocks to return. Enum values: • committed • uncommitted • all	COMMITTED	BlockListType
changeFeedContext (producer)	When using getChangeFeed producer operation, this gives additional context that is passed through the Http pipeline during the service call.		Context
changeFeedEndTime (producer)	When using getChangeFeed producer operation, this filters the results to return events approximately before the end time. Note: A few events belonging to the next hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the end time up by an hour.		OffsetDateTime

Name	Description	Default	Type
changeFeedStartTime (producer)	When using getChangeFeed producer operation, this filters the results to return events approximately after the start time. Note: A few events belonging to the previous hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the start time down by an hour.		OffsetDateTime
closeStreamAfterWrite (producer)	Close the stream after write or keep it open, default is true.	true	boolean
commitBlockListLater (producer)	When is set to true, the staged blocks will not be committed directly.	true	boolean
createAppendBlob (producer)	When is set to true, the append blocks will be created when committing append blocks.	true	boolean
createPageBlob (producer)	When is set to true, the page blob will be created when uploading page blob.	true	boolean
downloadLinkExpiration (producer)	Override the default expiration (millis) of URL download link.		Long
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	The blob operation that can be used with this component on the producer. Enum values: • listBlobContainers • createBlobContainer • deleteBlobContainer • listBlobs • getBlob • deleteBlob • downloadBlobToFile • downloadLink • uploadBlockBlob • stageBlockBlobList • commitBlobBlockList • getBlobBlockList • createAppendBlob • commitAppendBlob • createPageBlob • uploadPageBlob • resizePageBlob • clearPageBlob • getPageBlobRanges	listBlobContainers	BlobOperationsDefinition
pageBlobSize (producer)	Specifies the maximum size for the page blob, up to 8 TB. The page blob size must be aligned to a 512-byte boundary.	512	Long
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object

Name	Description	Default	Type
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Access key for the associated azure account name to be used for authentication with azure blob services.		String
sourceBlobAccessKey (security)	Source Blob Access Key: for copyblob operation, sadly, we need to have an accessKey for the source blob we want to copy Passing an accessKey as header, it's unsafe so we could set as key.		String

Required information options

To use this component, you have 3 options in order to provide the required Azure authentication information:

- Provide **accountName** and **accessKey** for your Azure account, this is the simplest way to get started. The accessKey can be generated through your Azure portal.
- Provide a StorageSharedKeyCredential instance which can be provided into **credentials** option.
- Provide a BlobServiceClient instance which can be provided into **blobServiceClient**. Note: You don't need to create a specific client, e.g: BlockBlobClient, the BlobServiceClient represents the upper level which can be used to retrieve lower level clients.

10.5. USAGE

For example, in order to download a blob content from the block blob **hello.txt** located on the **container1** in the **camelazure** storage account, use the following snippet:

```
from("azure-storage-blob://camelazure/container1?
blobName=hello.txt&accessKey=yourAccessKey").
to("file://blobdirectory");
```

10.5.1. Message headers evaluated by the component producer

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobTimeout	**BlobConstants.TIMEOUT**	**Duration**	All	An optional timeout value beyond which a {@link RuntimeException } will be raised.
CamelAzureStorageBlobMetadata	**BlobConstants.METADATA**	**Map<String,String>**	Operations related to container and blob	Metadata to associate with the container or blob.
CamelAzureStorageBlobPublicAccessType	**BlobConstants.PUBLIC_ACCESS_TYPE**	**PublicAccessType**	**createContainer**	Specifies how the data in this container is available to the public. Pass **null** for no public access.
CamelAzureStorageBlobRequestCondition	**BlobConstants.BLOB_REQUEST_CONDITION**	**BlobRequestConditions**	Operations related to container and blob	This contains values which will restrict the successful operation of a variety of requests to the conditions present. These conditions are entirely optional.
CamelAzureStorageBlobListDetails	**BlobConstants.BLOB_LIST_DETAILS**	**BlobListDetails**	**listBlobs**	The details for listing specific blobs

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobPrefix	**BlobConstants.PREFIX**	**String**	**listBlobs**,**getBlob**	Filters the results to return only blobs whose names begin with the specified prefix. May be null to return all blobs.
CamelAzureStorageBlobMaxResultsPerPage	**BlobConstants.MAX_RESULTS_PER_PAGE**	**Integer**	**listBlobs**	Specifies the maximum number of blobs to return, including all BlobPrefix elements. If the request does not specify maxResultsPerPage or specifies a value greater than 5,000, the server will return up to 5,000 items.
CamelAzureStorageBlobListBlobOptions	**BlobConstants.LIST_BLOB_OPTIONS**	**ListBlobsOptions**	**listBlobs**	Defines options available to configure the behavior of a call to listBlobsFlatSegment on a {@link BlobContainerClient} object.
CamelAzureStorageBlobHttpHeaders	**BlobConstants.BLOB_HTTP_HEADERS**	**BlobHttpHeaders**	**uploadBlockBlob**, **commitBlobBlockList**, **createAppendBlob**, **createPageBlob**	Additional parameters for a set of operations.
CamelAzureStorageBlobAccessTier	**BlobConstants.ACCESS_TIER**	**AccessTier**	**uploadBlockBlob**, **commitBlobBlockList**	Defines values for AccessTier.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobContentMD5	**BlobConstants.CONTENT_MD5**	**byte[]**	Most operations related to upload blob	An MD5 hash of the block content. This hash is used to verify the integrity of the block during transport. When this header is specified, the storage service compares the hash of the content that has arrived with this header value. Note that this MD5 hash is not stored with the blob. If the two hashes do not match, the operation will fail.
CamelAzureStorageBlobPageBlobRange	**BlobConstants.PAGE_BLOB_RANGE**	**PageRange**	Operations related to page blob	A {@link PageRange} object. Given that pages must be aligned with 512-byte boundaries, the start offset must be a modulus of 512 and the end offset must be a modulus of 512 - 1. Examples of valid byte ranges are 0-511, 512-1023, etc.
CamelAzureStorageBlobCommitBlobBlockListLater	**BlobConstants.COMMIT_BLOCK_LIST_LATER**	**boolean**	**stageBlockBlobList**	When is set to **true**, the staged blocks will not be committed directly.
CamelAzureStorageBlobCreateAppendBlob	**BlobConstants.CREATE_APPEND_BLOB**	**boolean**	**commitAppendBlob**	When is set to **true**, the append blocks will be created when committing append blocks.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobCreatePageBlob	**BlobConstants.CREATE_PAGE_BLOB**	**boolean**	**uploadPageBlob**	When is set to **true**, the page blob will be created when uploading page blob.
CamelAzureStorageBlobBlockListType	**BlobConstants.BLOCK_LIST_TYPE**	**BlockListType**	**getBlobBlockList**	Specifies which type of blocks to return.
CamelAzureStorageBlobPageBlobSize	**BlobConstants.PAGE_BLOB_SIZE**	**Long**	**createPageBlob**, **resizePageBlob**	Specifies the maximum size for the page blob, up to 8 TB. The page blob size must be aligned to a 512-byte boundary.
CamelAzureStorageBlobSequenceNumber	**BlobConstants.BLOB_SEQUENCE_NUMBER**	**Long**	**createPageBlob**	A user-controlled value that you can use to track requests. The value of the sequence number must be between 0 and 2^63 - 1.The default value is 0.
CamelAzureStorageBlobDeleteSnapshotsOptionType	**BlobConstants.DELETE_SNAPSHOT_OPTION_TYPE**	**DeleteSnapshotsOptionType**	**deleteBlob**	Specifies the behavior for deleting the snapshots on this blob. \{@code Include} will delete the base blob and all snapshots. \{@code Only} will delete only the snapshots. If a snapshot is being deleted, you must pass null.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobListBlobContainersOptions	**BlobConstants.LIST_BLOB_CONTAINERS_OPTIONS**	**ListBlobContainersOptions**	**listBlobContainers**	A {@link ListBlobContainersOptions} which specifies what data should be returned by the service.
CamelAzureStorageBlobParallelTransferOptions	**BlobConstants.PARALLEL_TRANSFER_OPTIONS**	**ParallelTransferOptions**	**downloadBlobToFile**	{@link ParallelTransferOptions} to use to download to file. Number of parallel transfers parameter is ignored.
CamelAzureStorageBlobFileDir	**BlobConstants.FILE_DIR**	**String**	**downloadBlobToFile**	The file directory where the downloaded blobs will be saved to.
CamelAzureStorageBlobDownloadLinkExpiration	**BlobConstants.DOWNLOAD_LINK_EXPIRATION**	**Long**	**downloadLink**	Override the default expiration (millis) of URL download link.
CamelAzureStorageBlobBlobName	**BlobConstants.BLOB_NAME**	**String**	Operations related to blob	Override/set the blob name on the exchange headers.
CamelAzureStorageBlobContainerName	**BlobConstants.BLOB_CONTAINER_NAME**	**String**	Operations related to container and blob	Override/set the container name on the exchange headers.
CamelAzureStorageBlobOperation	**BlobConstants.BLOB_OPERATION**	**BlobOperationsDefinition**	All	Specify the producer operation to execute, please see the doc on this page related to producer operation.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobRegex	**BlobConstants.REGEX**	**String**	**listBlobs,getBlob**	Filters the results to return only blobs whose names match the specified regular expression. May be null to return all. If both prefix and regex are set, regex takes the priority and prefix is ignored.
CamelAzureStorageBlobChangeFeedStartTime	**BlobConstants.CHANGE_FEED_START_TIME**	**OffsetDateTime**	**getChangeFeed**	It filters the results to return events approximately after the start time. Note: A few events belonging to the previous hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the start time down by an hour.
CamelAzureStorageBlobChangeFeedEndTime	**BlobConstants.CHANGE_FEED_END_TIME**	**OffsetDateTime**	**getChangeFeed**	It filters the results to return events approximately before the end time. Note: A few events belonging to the next hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the end time up by an hour.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageBlobChangeFeedContext	**BlobConstants.CHANGE_FEED_CONTEXT**	**Context**	**getChangeFeed**	This gives additional context that is passed through the Http pipeline during the service call.
CamelAzureStorageBlobSourceBlobAccountName	**BlobConstants.SOURCE_BLOB_ACCOUNT_NAME**	**String**	**copyBlob**	The source blob account name to be used as source account name in a copy blob operation
CamelAzureStorageBlobSourceBlobContainerName	**BlobConstants.SOURCE_BLOB_CONTAINER_NAME**	**String**	**copyBlob**	The source blob container name to be used as source container name in a copy blob operation

10.5.2. Message headers set by either component producer or consumer

Header	Variable Name	Type	Description
CamelAzureStorageBlobAccessTier	**BlobConstants.ACCESS_TIER**	**AccessTier**	Access tier of the blob.
CamelAzureStorageBlobAccessTierChangeTime	**BlobConstants.ACCESS_TIER_CHANGE_TIME**	**OffsetDateTime**	Datetime when the access tier of the blob last changed.
CamelAzureStorageBlobArchiveStatus	**BlobConstants.ARCHIVE_STATUS**	**ArchiveStatus**	Archive status of the blob.
CamelAzureStorageBlobCreationTime	**BlobConstants.CREATION_TIME**	**OffsetDateTime**	Creation time of the blob.
CamelAzureStorageBlobSequenceNumber	**BlobConstants.BLOB_SEQUENCE_NUMBER**	**Long**	The current sequence number for a page blob.

Header	Variable Name	Type	Description
CamelAzureStorageBlobBlobSize	**BlobConstants.BLOB_SIZE**	**long**	The size of the blob.
CamelAzureStorageBlobBlobType	**BlobConstants.BLOB_TYPE**	**BlobType**	The type of the blob.
CamelAzureStorageBlobCacheControl	**BlobConstants.CACHE_CONTROL**	**String**	Cache control specified for the blob.
CamelAzureStorageBlobCommittedBlockCount	**BlobConstants.COMMITTED_BLOCK_COUNT**	**Integer**	Number of blocks committed to an append blob
CamelAzureStorageBlobContentDisposition	**BlobConstants.CONTENT_DISPOSITION**	**String**	Content disposition specified for the blob.
CamelAzureStorageBlobContentEncoding	**BlobConstants.CONTENT_ENCODING**	**String**	Content encoding specified for the blob.
CamelAzureStorageBlobContentLanguage	**BlobConstants.CONTENT_LANGUAGE**	**String**	Content language specified for the blob.
CamelAzureStorageBlobContentMd5	**BlobConstants.CONTENT_MD5**	**byte[]**	Content MD5 specified for the blob.
CamelAzureStorageBlobContentType	**BlobConstants.CONTENT_TYPE**	**String**	Content type specified for the blob.
CamelAzureStorageBlobCopyCompletionTime	**BlobConstants.COPY_COMPILATION_TIME**	**OffsetDateTime**	Datetime when the last copy operation on the blob completed.
CamelAzureStorageBlobCopyDestinationSnapshot	**BlobConstants.COPY_DESTINATION_SNAPSHOT**	**String**	Snapshot identifier of the last incremental copy snapshot for the blob.
CamelAzureStorageBlobCopyId	**BlobConstants.COPY_ID**	**String**	Identifier of the last copy operation performed on the blob.

Header	Variable Name	Type	Description
CamelAzureStorageBlobCopyProgress	**BlobConstants.COPY_PROGRESS**	**String**	Progress of the last copy operation performed on the blob.
CamelAzureStorageBlobCopySource	**BlobConstants.COPY_SOURCE**	**String**	Source of the last copy operation performed on the blob.
CamelAzureStorageBlobCopyStatus	**BlobConstants.COPY_STATUS**	**CopyStatusType**	Status of the last copy operation performed on the blob.
CamelAzureStorageBlobCopyStatusDescription	**BlobConstants.COPY_STATUS_DESCRIPTION**	**String**	Description of the last copy operation on the blob.
CamelAzureStorageBlobETag	**BlobConstants.E_TAG**	**String**	The E Tag of the blob
CamelAzureStorageBlobIsAccessTierInferred	**BlobConstants.IS_ACCESS_TIER_INFRRRED**	**boolean**	Flag indicating if the access tier of the blob was inferred from properties of the blob.
CamelAzureStorageBlobIsIncrementalCopy	**BlobConstants.IS_INCREMENTAL_COPY**	**boolean**	Flag indicating if the blob was incrementally copied.
CamelAzureStorageBlobIsServerEncrypted	**BlobConstants.IS_SERVER_ENCRYPTED**	**boolean**	Flag indicating if the blob's content is encrypted on the server.
CamelAzureStorageBlobLastModified	**BlobConstants.LAST_MODIFIED**	**OffsetDateTime**	Datetime when the blob was last modified.
CamelAzureStorageBlobLeaseDuration	**BlobConstants.LEASE_DURATION**	**LeaseDurationType**	Type of lease on the blob.
CamelAzureStorageBlobLeaseState	**BlobConstants.LEASE_STATE**	**LeaseStateType**	State of the lease on the blob.
CamelAzureStorageBlobLeaseStatus	**BlobConstants.LEASE_STATUS**	**LeaseStatusType**	Status of the lease on the blob.

Header	Variable Name	Type	Description
CamelAzureStorageBlobMetadata	**BlobConstants.METADATA**	**Map<String, String>**	Additional metadata associated with the blob.
CamelAzureStorageBlobAppendOffset	**BlobConstants.APPEND_OFFSET**	**String**	The offset at which the block was committed to the block blob.
CamelAzureStorageBlobFileName	**BlobConstants.FILE_NAME**	**String**	The downloaded filename from the operation **downloadBlobToFile**.
CamelAzureStorageBlobDownloadLink	**BlobConstants.DOWNLOAD_LINK**	**String**	The download link generated by **downloadLink** operation.
CamelAzureStorageBlobRawHttpHeaders	**BlobConstants.RAW_HTTP_HEADERS**	**HttpHeaders**	Returns non-parsed httpHeaders that can be used by the user.

10.5.3. Advanced Azure Storage Blob configuration

If your Camel Application is running behind a firewall or if you need to have more control over the **BlobServiceClient** instance configuration, you can create your own instance:

```
StorageSharedKeyCredential credential = new StorageSharedKeyCredential("yourAccountName",
"yourAccessKey");
String uri = String.format("https://%s.blob.core.windows.net", "yourAccountName");

BlobServiceClient client = new BlobServiceClientBuilder()
                    .endpoint(uri)
                    .credential(credential)
                    .buildClient();
// This is camel context
context.getRegistry().bind("client", client);
```

Then refer to this instance in your Camel **azure-storage-blob** component configuration:

```
from("azure-storage-blob://cameldev/container1?blobName=myblob&serviceClient=#client")
.to("mock:result");
```

10.5.4. Automatic detection of BlobServiceClient client in registry

The component is capable of detecting the presence of an BlobServiceClient bean into the registry. If it's the only instance of that type it will be used as client and you won't have to define it as uri parameter, like the example above. This may be really useful for smarter configuration of the endpoint.

10.5.5. Azure Storage Blob Producer operations

Camel Azure Storage Blob component provides wide range of operations on the producer side:

Operations on the service level

For these operations, **accountName** is **required**.

Operation	Description
listBlobContainers	Get the content of the blob. You can restrict the output of this operation to a blob range.
getChangeFeed	Returns transaction logs of all the changes that occur to the blobs and the blob metadata in your storage account. The change feed provides ordered, guaranteed, durable, immutable, read-only log of these changes.

Operations on the container level

For these operations, **accountName** and **containerName** are **required**.

Operation	Description
createBlobContainer	Creates a new container within a storage account. If a container with the same name already exists, the producer will ignore it.
deleteBlobContainer	Deletes the specified container in the storage account. If the container doesn't exist the operation fails.
listBlobs	Returns a list of blobs in this container, with folder structures flattened.

Operations on the blob level

For these operations, **accountName**, **containerName** and **blobName** are **required**.

Operation	Blob Type	Description
getBlob	Common	Get the content of the blob. You can restrict the output of this operation to a blob range.
deleteBlob	Common	Delete a blob.
downloadBlobToFile	Common	Downloads the entire blob into a file specified by the path.The file will be created and must not exist, if the file already exists a {@link FileAlreadyExistsException} will be thrown.
downloadLink	Common	Generates the download link for the specified blob using shared access signatures (SAS). This by default only limit to 1hour of allowed access. However, you can override the default expiration duration through the headers.

Operation	Blob Type	Description
uploadBlockBlob	BlockBlob	Creates a new block blob, or updates the content of an existing block blob. Updating an existing block blob overwrites any existing metadata on the blob. Partial updates are not supported with PutBlob; the content of the existing blob is overwritten with the new content.
stageBlockBlobList	**BlockBlob**	Uploads the specified block to the block blob's "staging area" to be later committed by a call to commitBlobBlockList. However in case header **CamelAzureStorageBlobCommitBlobBlockListLater** or config **commitBlockListLater** is set to false, this will commit the blocks immediately after staging the blocks.
commitBlobBlockList	**BlockBlob**	Writes a blob by specifying the list of block IDs that are to make up the blob. In order to be written as part of a blob, a block must have been successfully written to the server in a prior **stageBlockBlobList** operation. You can call **commitBlobBlockList** to update a blob by uploading only those blocks that have changed, then committing the new and existing blocks together. Any blocks not specified in the block list and permanently deleted.
getBlobBlockList	**BlockBlob**	Returns the list of blocks that have been uploaded as part of a block blob using the specified block list filter.
createAppendBlob	**AppendBlob**	Creates a 0-length append blob. Call commitAppendBlo`b operation to append data to an append blob.
commitAppendBlob	**AppendBlob**	Commits a new block of data to the end of the existing append blob. In case of header **CamelAzureStorageBlobCreateAppendBlob** or config **createAppendBlob** is set to true, it will attempt to create the appendBlob through internal call to **createAppendBlob** operation first before committing.
createPageBlob	**PageBlob**	Creates a page blob of the specified length. Call **uploadPageBlob** operation to upload data data to a page blob.

Operation	Blob Type	Description
uploadPageBlob	**PageBlob**	Writes one or more pages to the page blob. The write size must be a multiple of 512. In case of header **CamelAzureStorageBlobCreatePageBlob** or config **createPageBlob** is set to true, it will attempt to create the appendBlob through internal call to **createPageBlob** operation first before uploading.
resizePageBlob	**PageBlob**	Resizes the page blob to the specified size (which must be a multiple of 512).
clearPageBlob	**PageBlob**	Frees the specified pages from the page blob. The size of the range must be a multiple of 512.
getPageBlobRanges	**PageBlob**	Returns the list of valid page ranges for a page blob or snapshot of a page blob.
copyBlob	**Common**	Copy a blob from one container to another one, even from different accounts.

Refer to the example section in this page to learn how to use these operations into your camel application.

10.5.6. Consumer Examples

To consume a blob into a file using file component, this can be done like this:

```
from("azure-storage-blob://camelazure/container1?
blobName=hello.txt&accountName=yourAccountName&accessKey=yourAccessKey").
to("file://blobdirectory");
```

However, you can also write to file directly without using the file component, you will need to specify **fileDir** folder path in order to save your blob in your machine.

```
from("azure-storage-blob://camelazure/container1?
blobName=hello.txt&accountName=yourAccountName&accessKey=yourAccessKey&fileDir=/var/to/awe
ome/dir").
to("mock:results");
```

Also, the component supports batch consumer, hence you can consume multiple blobs with only specifying the container name, the consumer will return multiple exchanges depending on the number of the blobs in the container.

Example

```
from("azure-storage-blob://camelazure/container1?
accountName=yourAccountName&accessKey=yourAccessKey&fileDir=/var/to/awesome/dir").
to("mock:results");
```

10.5.7. Producer Operations Examples

- **listBlobContainers**

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.LIST_BLOB_CONTAINERS_OPTIONS, new
ListBlobContainersOptions().setMaxResultsPerPage(10));
  })
  .to("azure-storage-blob://camelazure?operation=listBlobContainers&client&serviceClient=#client")
  .to("mock:result");
```

- **createBlobContainer**

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_CONTAINER_NAME, "newContainerName");
  })
  .to("azure-storage-blob://camelazure/container1?
operation=createBlobContainer&serviceClient=#client")
  .to("mock:result");
```

- **deleteBlobContainer**:

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_CONTAINER_NAME, "overridenName");
  })
  .to("azure-storage-blob://camelazure/container1?
operation=deleteBlobContainer&serviceClient=#client")
  .to("mock:result");
```

- **listBlobs**:

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_CONTAINER_NAME, "overridenName");
  })
  .to("azure-storage-blob://camelazure/container1?operation=listBlobs&serviceClient=#client")
  .to("mock:result");
```

- **getBlob**:

We can either set an **outputStream** in the exchange body and write the data to it. E.g:

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_CONTAINER_NAME, "overridenName");

    // set our body
    exchange.getIn().setBody(outputStream);
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=getBlob&serviceClient=#client")
  .to("mock:result");
```

If we don't set a body, then this operation will give us an **InputStream** instance which can proceeded further downstream:

```
from("direct:start")
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=getBlob&serviceClient=#client")
  .process(exchange -> {
     InputStream inputStream = exchange.getMessage().getBody(InputStream.class);
     // We use Apache common IO for simplicity, but you are free to do whatever dealing
     // with inputStream
     System.out.println(IOUtils.toString(inputStream, StandardCharsets.UTF_8.name()));
  })
  .to("mock:result");
```

- **deleteBlob**:

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_NAME, "overridenName");
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=deleteBlob&serviceClient=#client")
  .to("mock:result");
```

- **downloadBlobToFile**:

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_NAME, "overridenName");
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=downloadBlobToFile&fileDir=/var/mydir&serviceClient=#client")
  .to("mock:result");
```

- **downloadLink**

```
from("direct:start")
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=downloadLink&serviceClient=#client")
  .process(exchange -> {
      String link = exchange.getMessage().getHeader(BlobConstants.DOWNLOAD_LINK,
String.class);
      System.out.println("My link " + link);
  })
  .to("mock:result");
```

- **uploadBlockBlob**

```
from("direct:start")
  .process(exchange -> {
    // set the header you want the producer to evaluate, refer to the previous
    // section to learn about the headers that can be set
    // e.g:
    exchange.getIn().setHeader(BlobConstants.BLOB_NAME, "overridenName");
    exchange.getIn().setBody("Block Blob");
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=uploadBlockBlob&serviceClient=#client")
  .to("mock:result");
```

- **stageBlockBlobList**

```
from("direct:start")
  .process(exchange -> {
      final List<BlobBlock> blocks = new LinkedList<>();
      blocks.add(BlobBlock.createBlobBlock(new ByteArrayInputStream("Hello".getBytes())));
      blocks.add(BlobBlock.createBlobBlock(new ByteArrayInputStream("From".getBytes())));
      blocks.add(BlobBlock.createBlobBlock(new ByteArrayInputStream("Camel".getBytes())));

      exchange.getIn().setBody(blocks);
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=stageBlockBlobList&serviceClient=#client")
  .to("mock:result");
```

- **commitBlockBlobList**

```
from("direct:start")
  .process(exchange -> {
      // We assume here you have the knowledge of these blocks you want to commit
      final List<Block> blocksIds = new LinkedList<>();
      blocksIds.add(new Block().setName("id-1"));
      blocksIds.add(new Block().setName("id-2"));
      blocksIds.add(new Block().setName("id-3"));

      exchange.getIn().setBody(blocksIds);
  })
```

```
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=commitBlockBlobList&serviceClient=#client")
  .to("mock:result");
```

- **getBlobBlockList**

```
from("direct:start")
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=getBlobBlockList&serviceClient=#client")
  .log("${body}")
  .to("mock:result");
```

- **createAppendBlob**

```
from("direct:start")
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=createAppendBlob&serviceClient=#client")
  .to("mock:result");
```

- **commitAppendBlob**

```
from("direct:start")
  .process(exchange -> {
    final String data = "Hello world from my awesome tests!";
    final InputStream dataStream = new
ByteArrayInputStream(data.getBytes(StandardCharsets.UTF_8));

    exchange.getIn().setBody(dataStream);

    // of course you can set whatever headers you like, refer to the headers section to learn more
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=commitAppendBlob&serviceClient=#client")
  .to("mock:result");
```

- **createPageBlob**

```
from("direct:start")
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=createPageBlob&serviceClient=#client")
  .to("mock:result");
```

- **uploadPageBlob**

```
from("direct:start")
  .process(exchange -> {
    byte[] dataBytes = new byte[512]; // we set range for the page from 0-511
    new Random().nextBytes(dataBytes);
    final InputStream dataStream = new ByteArrayInputStream(dataBytes);
    final PageRange pageRange = new PageRange().setStart(0).setEnd(511);

    exchange.getIn().setHeader(BlobConstants.PAGE_BLOB_RANGE, pageRange);
    exchange.getIn().setBody(dataStream);
  })
```

```
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=uploadPageBlob&serviceClient=#client")
  .to("mock:result");
```

- **resizePageBlob**

```
from("direct:start")
  .process(exchange -> {
    final PageRange pageRange = new PageRange().setStart(0).setEnd(511);

    exchange.getIn().setHeader(BlobConstants.PAGE_BLOB_RANGE, pageRange);
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=resizePageBlob&serviceClient=#client")
  .to("mock:result");
```

- **clearPageBlob**

```
from("direct:start")
  .process(exchange -> {
    final PageRange pageRange = new PageRange().setStart(0).setEnd(511);

    exchange.getIn().setHeader(BlobConstants.PAGE_BLOB_RANGE, pageRange);
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=clearPageBlob&serviceClient=#client")
  .to("mock:result");
```

- **getPageBlobRanges**

```
from("direct:start")
  .process(exchange -> {
    final PageRange pageRange = new PageRange().setStart(0).setEnd(511);

    exchange.getIn().setHeader(BlobConstants.PAGE_BLOB_RANGE, pageRange);
  })
  .to("azure-storage-blob://camelazure/container1?
blobName=blob&operation=getPageBlobRanges&serviceClient=#client")
  .log("${body}")
  .to("mock:result");
```

- **copyBlob**

```
from("direct:copyBlob")
  .process(exchange -> {
    exchange.getIn().setHeader(BlobConstants.BLOB_NAME, "file.txt");
    exchange.getMessage().setHeader(BlobConstants.SOURCE_BLOB_CONTAINER_NAME,
"containerblob1");
    exchange.getMessage().setHeader(BlobConstants.SOURCE_BLOB_ACCOUNT_NAME,
"account");
  })
  .to("azure-storage-blob://account/containerblob2?
operation=copyBlob&sourceBlobAccessKey=RAW(accessKey)")
  .to("mock:result");
```

In this way the file.txt in the container containerblob1 of the account 'account', will be copied to the container containerblob2 of the same account.

10.5.8. Development Notes (Important)

All integration tests use Testcontainers and run by default. Obtaining of Azure accessKey and accountName is needed to be able to run all integration tests using Azure services. In addition to the mocked unit tests you **will need to run the integration tests with every change you make or even client upgrade as the Azure client can break things even on minor versions upgrade.** To run the integration tests, on this component directory, run the following maven command:

```
mvn verify -PfullTests -DaccountName=myacc -DaccessKey=mykey
```

Whereby **accountName** is your Azure account name and **accessKey** is the access key being generated from Azure portal.

10.6. SPRING BOOT AUTO-CONFIGURATION

When using azure-storage-blob with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-azure-storage-blob-starter</artifactId>
</dependency>
```

The component supports 32 options, which are listed below.

Name	Description	Default	Type
camel.component.azure-storage-blob.access-key	Access key for the associated azure account name to be used for authentication with azure blob services.		String
camel.component.azure-storage-blob.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.azure-storage-blob.blob-name	The blob name, to consume specific blob from a container. However on producer, is only required for the operations on the blob level.		String
camel.component.azure-storage-blob.blob-offset	Set the blob offset for the upload or download operations, default is 0.	0	Long

Name	Description	Default	Type
camel.component.azure-storage-blob.blob-sequence-number	A user-controlled value that you can use to track requests. The value of the sequence number must be between 0 and 263 - 1.The default value is 0.	0	Long
camel.component.azure-storage-blob.blob-type	The blob type in order to initiate the appropriate settings for each blob type.		BlobType
camel.component.azure-storage-blob.block-list-type	Specifies which type of blocks to return.		BlockListType
camel.component.azure-storage-blob.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.azure-storage-blob.change-feed-context	When using getChangeFeed producer operation, this gives additional context that is passed through the Http pipeline during the service call. The option is a com.azure.core.util.Context type.		Context
camel.component.azure-storage-blob.change-feed-end-time	When using getChangeFeed producer operation, this filters the results to return events approximately before the end time. Note: A few events belonging to the next hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the end time up by an hour. The option is a java.time.OffsetDateTime type.		OffsetDateTime
camel.component.azure-storage-blob.change-feed-start-time	When using getChangeFeed producer operation, this filters the results to return events approximately after the start time. Note: A few events belonging to the previous hour can also be returned. A few events belonging to this hour can be missing; to ensure all events from the hour are returned, round the start time down by an hour. The option is a java.time.OffsetDateTime type.		OffsetDateTime

Name	Description	Default	Type
camel.component.azure-storage-blob.close-stream-after-read	Close the stream after read or keep it open, default is true.	true	Boolean
camel.component.azure-storage-blob.close-stream-after-write	Close the stream after write or keep it open, default is true.	true	Boolean
camel.component.azure-storage-blob.commit-block-list-later	When is set to true, the staged blocks will not be committed directly.	true	Boolean
camel.component.azure-storage-blob.configuration	The component configurations. The option is a org.apache.camel.component.azure.storage.blob.BlobConfiguration type.		BlobConfiguration
camel.component.azure-storage-blob.create-append-blob	When is set to true, the append blocks will be created when committing append blocks.	true	Boolean
camel.component.azure-storage-blob.create-page-blob	When is set to true, the page blob will be created when uploading page blob.	true	Boolean
camel.component.azure-storage-blob.credentials	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information. The option is a com.azure.storage.common.StorageSharedKeyCredential type.		StorageSharedKeyCredential
camel.component.azure-storage-blob.data-count	How many bytes to include in the range. Must be greater than or equal to 0 if specified.		Long
camel.component.azure-storage-blob.download-link-expiration	Override the default expiration (millis) of URL download link.		Long

Name	Description	Default	Type
camel.component.azure-storage-blob.enabled	Whether to enable auto configuration of the azure-storage-blob component. This is enabled by default.		Boolean
camel.component.azure-storage-blob.file-dir	The file directory where the downloaded blobs will be saved to, this can be used in both, producer and consumer.		String
camel.component.azure-storage-blob.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.azure-storage-blob.max-results-per-page	Specifies the maximum number of blobs to return, including all BlobPrefix elements. If the request does not specify maxResultsPerPage or specifies a value greater than 5,000, the server will return up to 5,000 items.		Integer
camel.component.azure-storage-blob.max-retry-requests	Specifies the maximum number of additional HTTP Get requests that will be made while reading the data from a response body.	0	Integer
camel.component.azure-storage-blob.operation	The blob operation that can be used with this component on the producer.		BlobOperationsDefinition
camel.component.azure-storage-blob.page-blob-size	Specifies the maximum size for the page blob, up to 8 TB. The page blob size must be aligned to a 512-byte boundary.	512	Long
camel.component.azure-storage-blob.prefix	Filters the results to return only blobs whose names begin with the specified prefix. May be null to return all blobs.		String
camel.component.azure-storage-blob.regex	Filters the results to return only blobs whose names match the specified regular expression. May be null to return all if both prefix and regex are set, regex takes the priority and prefix is ignored.		String

Name	Description	Default	Type
camel.component.azure-storage-blob.service-client	Client to a storage account. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. It may also be used to construct URLs to blobs and containers. This client contains operations on a service account. Operations on a container are available on BlobContainerClient through BlobServiceClient#getBlobContainerClient(String), and operations on a blob are available on BlobClient through BlobContainerClient#getBlobClient(String). The option is a com.azure.storage.blob.BlobServiceClient type.		BlobServiceClient
camel.component.azure-storage-blob.source-blob-access-key	Source Blob Access Key: for copyblob operation, sadly, we need to have an accessKey for the source blob we want to copy Passing an accessKey as header, it's unsafe so we could set as key.		String
camel.component.azure-storage-blob.timeout	An optional timeout value beyond which a RuntimeException will be raised. The option is a java.time.Duration type.		Duration

CHAPTER 11. AZURE STORAGE QUEUE SERVICE

Both producer and consumer are supported

The Azure Storage Queue component supports storing and retrieving the messages to/from Azure Storage Queue service using **Azure APIs v12** However in case of versions above v12, we will see if this component can adopt these changes depending on how much breaking changes can result.

Prerequisites

You must have a valid Windows Azure Storage account. More information is available at Azure Documentation Portal.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-azure-storage-queue</artifactId>
  <version>{CamelSBVersion}</version>
 <!-- use the same version as your Camel core version -->
</dependency>
```

11.1. URI FORMAT

```
azure-storage-queue://accountName[/queueName][?options]
```

In case of consumer, accountName and queueName are required. In case of producer, it depends on the operation that being requested, for example if operation is on a service level, e.b: listQueues, only accountName is required, but in case of operation being requested on the queue level, for example, createQueue, sendMessage.. etc, both accountName and queueName are required.

The queue will be created if it does not already exist. You can append query options to the URI in the following format,

?options=value&option2=value&…

11.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

11.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

11.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

11.3. COMPONENT OPTIONS

The Azure Storage Queue Service component supports 15 options, which are listed below.

Name	Description	Default	Type
configuration (common)	The component configurations.		QueueConfiguration
serviceClient (common)	**Autowired** Service client to a storage account to interact with the queue service. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. This client contains all the operations for interacting with a queue account in Azure Storage. Operations allowed by the client are creating, listing, and deleting queues, retrieving and updating properties of the account, and retrieving statistics of the account.		QueueServiceClient
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
createQueue (producer)	When is set to true, the queue will be automatically created when sending messages to the queue.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
operation (producer)	Queue service operation hint to the producer. Enum values: • listQueues • createQueue • deleteQueue • clearQueue • sendMessage • deleteMessage • receiveMessages • peekMessages • updateMessage		QueueOperationDefinition
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
maxMessages (queue)	Maximum number of messages to get, if there are less messages exist in the queue than requested all the messages will be returned. If left empty only 1 message will be retrieved, the allowed range is 1 to 32 messages.	1	Integer
messageId (queue)	The ID of the message to be deleted or updated.		String

Name	Description	Default	Type
popReceipt (queue)	Unique identifier that must match for the message to be deleted or updated.		String
timeout (queue)	An optional timeout applied to the operation. If a response is not returned before the timeout concludes a RuntimeException will be thrown.		Duration
timeToLive (queue)	How long the message will stay alive in the queue. If unset the value will default to 7 days, if -1 is passed the message will not expire. The time to live must be -1 or any positive number. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe.		Duration
visibilityTimeout (queue)	The timeout period for how long the message is invisible in the queue. The timeout must be between 1 seconds and 7 days. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe.		Duration
accessKey (security)	Access key for the associated azure account name to be used for authentication with azure queue services.		String
credentials (security)	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information.		StorageSharedKeyCredential

11.4. ENDPOINT OPTIONS

The Azure Storage Queue Service endpoint is configured using URI syntax:

```
azure-storage-queue:accountName/queueName
```

with the following path and query parameters:

11.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
accountName (common)	Azure account name to be used for authentication with azure queue services.		String
queueName (common)	The queue resource name.		String

11.4.2. Query Parameters (31 parameters)

Name	Description	Default	Type
serviceClient (common)	**Autowired** Service client to a storage account to interact with the queue service. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. This client contains all the operations for interacting with a queue account in Azure Storage. Operations allowed by the client are creating, listing, and deleting queues, retrieving and updating properties of the account, and retrieving statistics of the account.		QueueServiceClient
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
createQueue (producer)	When is set to true, the queue will be automatically created when sending messages to the queue.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	Queue service operation hint to the producer. Enum values: • listQueues • createQueue • deleteQueue • clearQueue • sendMessage • deleteMessage • receiveMessages • peekMessages • updateMessage		QueueOperationDefinition
maxMessages (queue)	Maximum number of messages to get, if there are less messages exist in the queue than requested all the messages will be returned. If left empty only 1 message will be retrieved, the allowed range is 1 to 32 messages.	1	Integer
messageId (queue)	The ID of the message to be deleted or updated.		String
popReceipt (queue)	Unique identifier that must match for the message to be deleted or updated.		String
timeout (queue)	An optional timeout applied to the operation. If a response is not returned before the timeout concludes a RuntimeException will be thrown.		Duration
timeToLive (queue)	How long the message will stay alive in the queue. If unset the value will default to 7 days, if -1 is passed the message will not expire. The time to live must be -1 or any positive number. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe.		Duration

Name	Description	Default	Type
visibilityTimeout (queue)	The timeout period for how long the message is invisible in the queue. The timeout must be between 1 seconds and 7 days. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe.		Duration
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Access key for the associated azure account name to be used for authentication with azure queue services.		String
credentials (security)	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information.		StorageSharedKeyCredential

Required information options

To use this component, you have 3 options in order to provide the required Azure authentication information:

- Provide **accountName** and **accessKey** for your Azure account, this is the simplest way to get started. The accessKey can be generated through your Azure portal.
- Provide a StorageSharedKeyCredential instance which can be provided into **credentials** option.
- Provide a QueueServiceClient instance which can be provided into **serviceClient**. Note: You don't need to create a specific client, e.g: QueueClient, the QueueServiceClient represents the upper level which can be used to retrieve lower level clients.

11.5. USAGE

For example in order to get a message content from the queue **messageQueue** in the **storageAccount** storage account and, use the following snippet:

```
from("azure-storage-queue://storageAccount/messageQueue?accessKey=yourAccessKey").
to("file://queuedirectory");
```

11.5.1. Message headers evaluated by the component producer

Header	Variable Name	Type	Operations	Description
CamelAzureStorageQueueSegmentOptions	**QueueConstants.QUEUES_SEGMENT_OPTIONS**	**QueuesSegmentOptions**	**listQueues**	Options for listing queues
CamelAzureStorageQueueTimeout	**QueueConstants.TIMEOUT**	**Duration**	All	An optional timeout value beyond which a \{@link RuntimeException} will be raised.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageQueueMetadata	**QueueConstants.METADATA**	**Map<String,String>**	**createQueue**	Metadata to associate with the queue
CamelAzureStorageQueueTimeToLive	**QueueConstants.TIME_TO_LIVE**	**Duration**	**sendMessage**	How long the message will stay alive in the queue. If unset the value will default to 7 days, if -1 is passed the message will not expire. The time to live must be -1 or any positive number.
CamelAzureStorageQueueVisibilityTimeout	**QueueConstants.VISIBILITY_TIMEOUT**	**Duration**	**sendMessage**, **receiveMessages**, **updateMessage**	The timeout period for how long the message is invisible in the queue. If unset the value will default to 0 and the message will be instantly visible. The timeout must be between 0 seconds and 7 days.
CamelAzureStorageQueueCreateQueue	**QueueConstants.CREATE_QUEUE**	**boolean**	**sendMessage**	When is set to **true**, the queue will be automatically created when sending messages to the queue.
CamelAzureStorageQueuePopReceipt	**QueueConstants.POP_RECEIPT**	**String**	**deleteMessage**, **updateMessage**	Unique identifier that must match for the message to be deleted or updated.
CamelAzureStorageQueueMessageId	**QueueConstants.MESSAGE_ID**	**String**	**deleteMessage**, **updateMessage**	The ID of the message to be deleted or updated.

Header	Variable Name	Type	Operations	Description
CamelAzureStorageQueueMaxMessages	**QueueConstants.MAX_MESSAGES**	**Integer**	**receiveMessages**, **peekMessages**	Maximum number of messages to get, if there are less messages exist in the queue than requested all the messages will be returned. If left empty only 1 message will be retrieved, the allowed range is 1 to 32 messages.
CamelAzureStorageQueueOperation	**QueueConstants.QUEUE_OPERATION**	**QueueOperationDefinition**	All	Specify the producer operation to execute, please see the doc on this page related to producer operation.
CamelAzureStorageQueueName	**QueueConstants.QUEUE_NAME**	**String**	All	Override the queue name.

11.5.2. Message headers set by either component producer or consumer

Header	Variable Name	Type	Description
CamelAzureStorageQueueMessageId	**QueueConstants.MESSAGE_ID**	**String**	The ID of message that being sent to the queue.
CamelAzureStorageQueueInsertionTime	**QueueConstants.INSERTION_TIME**	**OffsetDateTime**	The time the Message was inserted into the Queue.
CamelAzureStorageQueueExpirationTime	**QueueConstants.EXPIRATION_TIME**	**OffsetDateTime**	The time that the Message will expire and be automatically deleted.
CamelAzureStorageQueuePopReceipt	**QueueConstants.POP_RECEIPT**	**String**	This value is required to delete/update the Message. If deletion fails using this popreceipt then the message has been dequeued by another client.

Header	Variable Name	Type	Description
CamelAzureStorageQueueTimeNextVisible	**QueueConstants.TIME_NEXT_VISIBLE**	**OffsetDateTime**	The time that the message will again become visible in the Queue.
CamelAzureStorageQueueDequeueCount	**QueueConstants.DEQUEUE_COUNT**	**long**	The number of times the message has been dequeued.
CamelAzureStorageQueueRawHttpHeaders	**QueueConstants.RAW_HTTP_HEADERS**	**HttpHeaders**	Returns non-parsed httpHeaders that can be used by the user.

11.5.3. Advanced Azure Storage Queue configuration

If your Camel Application is running behind a firewall or if you need to have more control over the **QueueServiceClient** instance configuration, you can create your own instance:

```
StorageSharedKeyCredential credential = new StorageSharedKeyCredential("yourAccountName",
"yourAccessKey");
String uri = String.format("https://%s.queue.core.windows.net", "yourAccountName");

QueueServiceClient client = new QueueServiceClientBuilder()
                    .endpoint(uri)
                    .credential(credential)
                    .buildClient();
// This is camel context
context.getRegistry().bind("client", client);
```

Then refer to this instance in your Camel **azure-storage-queue** component configuration:

```
from("azure-storage-queue://cameldev/queue1?serviceClient=#client")
.to("file://outputFolder?fileName=output.txt&fileExist=Append");
```

11.5.4. Automatic detection of QueueServiceClient client in registry

The component is capable of detecting the presence of an QueueServiceClient bean into the registry. If it's the only instance of that type it will be used as client and you won't have to define it as uri parameter, like the example above. This may be really useful for smarter configuration of the endpoint.

11.5.5. Azure Storage Queue Producer operations

Camel Azure Storage Queue component provides wide range of operations on the producer side:

Operations on the service level

For these operations, **accountName** is **required**.

Operation	Description
listQueues	Lists the queues in the storage account that pass the filter starting at the specified marker.

Operations on the queue level

For these operations, **accountName** and **queueName** are **required**.

Operation	Description
createQueue	Creates a new queue.
deleteQueue	Permanently deletes the queue.
clearQueue	Deletes all messages in the queue..
sendMessage	**Default Producer Operation** Sends a message with a given time-to-live and a timeout period where the message is invisible in the queue. The message text is evaluated from the exchange message body. By default, if the queue doesn`t exist, it will create an empty queue first. If you want to disable this, set the config **createQueue** or header **CamelAzureStorageQueueCreateQueue** to **false**.
deleteMessage	Deletes the specified message in the queue.
receiveMessages	Retrieves up to the maximum number of messages from the queue and hides them from other operations for the timeout period. However it will not dequeue the message from the queue due to reliability reasons.
peekMessages	Peek messages from the front of the queue up to the maximum number of messages.
updateMessage	Updates the specific message in the queue with a new message and resets the visibility timeout. The message text is evaluated from the exchange message body.

Refer to the example section in this page to learn how to use these operations into your camel application.

11.5.6. Consumer Examples

To consume a queue into a file component with maximum 5 messages in one batch, this can be done like this:

```
from("azure-storage-queue://cameldev/queue1?serviceClient=#client&maxMessages=5")
.to("file://outputFolder?fileName=output.txt&fileExist=Append");
```

11.5.7. Producer Operations Examples

- **listQueues**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g, to only returns list of queues with 'awesome' prefix:
      exchange.getIn().setHeader(QueueConstants.QUEUES_SEGMENT_OPTIONS, new
QueuesSegmentOptions().setPrefix("awesome"));
    })
    .to("azure-storage-queue://cameldev?serviceClient=#client&operation=listQueues")
    .log("${body}")
    .to("mock:result");
```

- **createQueue**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g:
      exchange.getIn().setHeader(QueueConstants.QUEUE_NAME, "overrideName");
    })
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=createQueue");
```

- **deleteQueue**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g:
      exchange.getIn().setHeader(QueueConstants.QUEUE_NAME, "overrideName");
    })
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=deleteQueue");
```

- **clearQueue**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g:
      exchange.getIn().setHeader(QueueConstants.QUEUE_NAME, "overrideName");
    })
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=clearQueue");
```

- **sendMessage**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g:
```

```
        exchange.getIn().setBody("message to send");
        // we set a visibility of 1min
        exchange.getIn().setHeader(QueueConstants.VISIBILITY_TIMEOUT, Duration.ofMinutes(1));
      })
      .to("azure-storage-queue://cameldev/test?serviceClient=#client");
```

- **deleteMessage**:

```
from("direct:start")
    .process(exchange -> {
      // set the header you want the producer to evaluate, refer to the previous
      // section to learn about the headers that can be set
      // e.g:
      // Mandatory header:
      exchange.getIn().setHeader(QueueConstants.MESSAGE_ID, "1");
      // Mandatory header:
      exchange.getIn().setHeader(QueueConstants.POP_RECEIPT, "PAAAAHEEERXXX-1");
    })
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=deleteMessage");
```

- **receiveMessages**:

```
from("direct:start")
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=receiveMessages")
    .process(exchange -> {
        final List<QueueMessageItem> messageItems = exchange.getMessage().getBody(List.class);
        messageItems.forEach(messageItem -> System.out.println(messageItem.getMessageText()));
    })
    .to("mock:result");
```

- **peekMessages**:

```
from("direct:start")
    .to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=peekMessages")
    .process(exchange -> {
        final List<PeekedMessageItem> messageItems = exchange.getMessage().getBody(List.class);
        messageItems.forEach(messageItem -> System.out.println(messageItem.getMessageText()));
    })
    .to("mock:result");
```

- **updateMessage**:

```
from("direct:start")
    .process(exchange -> {
        // set the header you want the producer to evaluate, refer to the previous
        // section to learn about the headers that can be set
        // e.g:
        exchange.getIn().setBody("new message text");
        // Mandatory header:
        exchange.getIn().setHeader(QueueConstants.MESSAGE_ID, "1");
        // Mandatory header:
        exchange.getIn().setHeader(QueueConstants.POP_RECEIPT, "PAAAAHEEERXXX-1");
        // Mandatory header:
```

```
    exchange.getIn().setHeader(QueueConstants.VISIBILITY_TIMEOUT, Duration.ofMinutes(1));
})
.to("azure-storage-queue://cameldev/test?serviceClient=#client&operation=updateMessage");
```

11.5.8. Development Notes (Important)

When developing on this component, you will need to obtain your Azure accessKey in order to run the integration tests. In addition to the mocked unit tests you **will need to run the integration tests with every change you make or even client upgrade as the Azure client can break things even on minor versions upgrade.** To run the integration tests, on this component directory, run the following maven command:

```
mvn verify -PfullTests -DaccountName=myacc -DaccessKey=mykey
```

Whereby **accountName** is your Azure account name and **accessKey** is the access key being generated from Azure portal.

11.6. SPRING BOOT AUTO-CONFIGURATION

When using azure-storage-queue with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-azure-storage-queue-starter</artifactId>
</dependency>
```

The component supports 16 options, which are listed below.

Name	Description	Default	Type
camel.component.azure-storage-queue.access-key	Access key for the associated azure account name to be used for authentication with azure queue services.		String
camel.component.azure-storage-queue.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.azure-storage-queue.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.azure-storage-queue.configuration	The component configurations. The option is a org.apache.camel.component.azure.storage.queue.QueueConfiguration type.		QueueConfiguration
camel.component.azure-storage-queue.create-queue	When is set to true, the queue will be automatically created when sending messages to the queue.	false	Boolean
camel.component.azure-storage-queue.credentials	StorageSharedKeyCredential can be injected to create the azure client, this holds the important authentication information. The option is a com.azure.storage.common.StorageSharedKeyCredential type.		StorageSharedKeyCredential
camel.component.azure-storage-queue.enabled	Whether to enable auto configuration of the azure-storage-queue component. This is enabled by default.		Boolean
camel.component.azure-storage-queue.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.azure-storage-queue.max-messages	Maximum number of messages to get, if there are less messages exist in the queue than requested all the messages will be returned. If left empty only 1 message will be retrieved, the allowed range is 1 to 32 messages.	1	Integer

Name	Description	Default	Type
camel.component.azure-storage-queue.message-id	The ID of the message to be deleted or updated.		String
camel.component.azure-storage-queue.operation	Queue service operation hint to the producer.		QueueOperationDefinition
camel.component.azure-storage-queue.pop-receipt	Unique identifier that must match for the message to be deleted or updated.		String
camel.component.azure-storage-queue.service-client	Service client to a storage account to interact with the queue service. This client does not hold any state about a particular storage account but is instead a convenient way of sending off appropriate requests to the resource on the service. This client contains all the operations for interacting with a queue account in Azure Storage. Operations allowed by the client are creating, listing, and deleting queues, retrieving and updating properties of the account, and retrieving statistics of the account. The option is a com.azure.storage.queue.QueueServiceClient type.		QueueServiceClient
camel.component.azure-storage-queue.time-to-live	How long the message will stay alive in the queue. If unset the value will default to 7 days, if -1 is passed the message will not expire. The time to live must be -1 or any positive number. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe. The option is a java.time.Duration type.		Duration
camel.component.azure-storage-queue.timeout	An optional timeout applied to the operation. If a response is not returned before the timeout concludes a RuntimeException will be thrown. The option is a java.time.Duration type.		Duration

Name	Description	Default	Type
camel.component.azure-storage-queue.visibility-timeout	The timeout period for how long the message is invisible in the queue. The timeout must be between 1 seconds and 7 days. The format should be in this form: PnDTnHnMn.nS., e.g: PT20.345S – parses as 20.345 seconds, P2D – parses as 2 days However, in case you are using EndpointDsl/ComponentDsl, you can do something like Duration.ofSeconds() since these Java APIs are typesafe. The option is a java.time.Duration type.		Duration

CHAPTER 12. BEAN

Only producer is supported

The Bean component binds beans to Camel message exchanges.

12.1. URI FORMAT

```
bean:beanName[?options]
```

Where **beanID** can be any string which is used to look up the bean in the Registry

12.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

12.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

12.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

12.3. COMPONENT OPTIONS

The Bean component supports 4 options, which are listed below.

Name	Description	Default	Type
cache (producer)	**Deprecated** Use singleton option instead.	true	Boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
scope (producer)	Scope of bean. When using singleton scope (default) the bean is created or looked up only once and reused for the lifetime of the endpoint. The bean should be thread-safe in case concurrent threads is calling the bean at the same time. When using request scope the bean is created or looked up once per request (exchange). This can be used if you want to store state on a bean while processing a request and you want to call the same bean instance multiple times while processing the request. The bean does not have to be thread-safe as the instance is only called from the same request. When using delegate scope, then the bean will be looked up or created per call. However in case of lookup then this is delegated to the bean registry such as Spring or CDI (if in use), which depends on their configuration can act as either singleton or prototype scope. so when using prototype then this depends on the delegated registry. Enum values: • Singleton • Request • Prototype	Singleton	BeanScope
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

12.4. ENDPOINT OPTIONS

The Bean endpoint is configured using URI syntax:

```
bean:beanName
```

with the following path and query parameters:

12.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
beanName (common)	**Required** Sets the name of the bean to invoke.		String

12.4.2. Query Parameters (5 parameters)

Name	Description	Default	Type
cache (common)	**Deprecated** Use scope option instead.		Boolean
method (common)	Sets the name of the method to invoke on the bean.		String

Name	Description	Default	Type
scope (common)	Scope of bean. When using singleton scope (default) the bean is created or looked up only once and reused for the lifetime of the endpoint. The bean should be thread-safe in case concurrent threads is calling the bean at the same time. When using request scope the bean is created or looked up once per request (exchange). This can be used if you want to store state on a bean while processing a request and you want to call the same bean instance multiple times while processing the request. The bean does not have to be thread-safe as the instance is only called from the same request. When using prototype scope, then the bean will be looked up or created per call. However in case of lookup then this is delegated to the bean registry such as Spring or CDI (if in use), which depends on their configuration can act as either singleton or prototype scope. so when using prototype then this depends on the delegated registry. Enum values: • Singleton • Request • Prototype	Singleton	BeanScope
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
parameters (advanced)	Used for configuring additional properties on the bean.		Map

12.5. USING

The object instance that is used to consume messages must be explicitly registered with the Registry. For example, if you are using Spring you must define the bean in the Spring configuration XML file.

You can also register beans manually via Camel's **Registry** with the **bind** method.

Once an endpoint has been registered, you can build Camel routes that use it to process exchanges.

A **bean:** endpoint cannot be defined as the input to the route; i.e. you cannot consume from it, you can only route from some inbound message Endpoint to the bean endpoint as output. So consider using a **direct:** or **queue:** endpoint as the input.

You can use the **createProxy()** methods on ProxyHelper to create a proxy that will generate exchanges and send them to any endpoint:

And the same route using XML DSL:

```
<route>
  <from uri="direct:hello"/>
  <to uri="bean:bye"/>
</route>
```

12.6. BEAN AS ENDPOINT

Camel also supports invoking Bean as an Endpoint. What happens is that when the exchange is routed to the **myBean** Camel will use the Bean Binding to invoke the bean. The source for the bean is just a plain POJO.

Camel will use Bean Binding to invoke the **sayHello** method, by converting the Exchange's In body to the **String** type and storing the output of the method on the Exchange Out body.

12.7. JAVA DSL BEAN SYNTAX

Java DSL comes with syntactic sugar for the component. Instead of specifying the bean explicitly as the endpoint (i.e. **to("bean:beanName")**) you can use the following syntax:

```
// Send message to the bean endpoint
// and invoke method resolved using Bean Binding.
from("direct:start").bean("beanName");

// Send message to the bean endpoint
// and invoke given method.
from("direct:start").bean("beanName", "methodName");
```

Instead of passing name of the reference to the bean (so that Camel will lookup for it in the registry), you can specify the bean itself:

```
// Send message to the given bean instance.
from("direct:start").bean(new ExampleBean());

// Explicit selection of bean method to be invoked.
from("direct:start").bean(new ExampleBean(), "methodName");

// Camel will create the instance of bean and cache it for you.
from("direct:start").bean(ExampleBean.class);
```

12.8. BEAN BINDING

How bean methods to be invoked are chosen (if they are not specified explicitly through the **method** parameter) and how parameter values are constructed from the Message are all defined by the Bean Binding mechanism which is used throughout all of the various Bean Integration mechanisms in Camel.

12.9. SPRING BOOT AUTO-CONFIGURATION

When using bean with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-bean-starter</artifactId>
</dependency>
```

The component supports 13 options, which are listed below.

Name	Description	Default	Type
camel.component.bean.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.bean.enabled	Whether to enable auto configuration of the bean component. This is enabled by default.		Boolean
camel.component.bean.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.bean.scope	Scope of bean. When using singleton scope (default) the bean is created or looked up only once and reused for the lifetime of the endpoint. The bean should be thread-safe in case concurrent threads is calling the bean at the same time. When using request scope the bean is created or looked up once per request (exchange). This can be used if you want to store state on a bean while processing a request and you want to call the same bean instance multiple times while processing the request. The bean does not have to be thread-safe as the instance is only called from the same request. When using delegate scope, then the bean will be looked up or created per call. However in case of lookup then this is delegated to the bean registry such as Spring or CDI (if in use), which depends on their configuration can act as either singleton or prototype scope. so when using prototype then this depends on the delegated registry.		BeanScope
camel.component.class.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.class.enabled	Whether to enable auto configuration of the class component. This is enabled by default.		Boolean
camel.component.class.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.class.scope	Scope of bean. When using singleton scope (default) the bean is created or looked up only once and reused for the lifetime of the endpoint. The bean should be thread-safe in case concurrent threads is calling the bean at the same time. When using request scope the bean is created or looked up once per request (exchange). This can be used if you want to store state on a bean while processing a request and you want to call the same bean instance multiple times while processing the request. The bean does not have to be thread-safe as the instance is only called from the same request. When using delegate scope, then the bean will be looked up or created per call. However in case of lookup then this is delegated to the bean registry such as Spring or CDI (if in use), which depends on their configuration can act as either singleton or prototype scope. so when using prototype then this depends on the delegated registry.		BeanScope
camel.language.bean.enabled	Whether to enable auto configuration of the bean language. This is enabled by default.		Boolean
camel.language.bean.scope	Scope of bean. When using singleton scope (default) the bean is created or looked up only once and reused for the lifetime of the endpoint. The bean should be thread-safe in case concurrent threads is calling the bean at the same time. When using request scope the bean is created or looked up once per request (exchange). This can be used if you want to store state on a bean while processing a request and you want to call the same bean instance multiple times while processing the request. The bean does not have to be thread-safe as the instance is only called from the same request. When using prototype scope, then the bean will be looked up or created per call. However in case of lookup then this is delegated to the bean registry such as Spring or CDI (if in use), which depends on their configuration can act as either singleton or prototype scope. So when using prototype scope then this depends on the bean registry implementation.	Singleton	String
camel.language.bean.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.component.bean.cache	**Deprecated** Use singleton option instead.	true	Boolean

Name	Description	Default	Type
camel.component.class.cache	**Deprecated** Use singleton option instead.	true	Boolean

CHAPTER 13. BEAN VALIDATOR

Only producer is supported

The Validator component performs bean validation of the message body using the Java Bean Validation API (). Camel uses the reference implementation, which is Hibernate Validator.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bean-validator</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

13.1. URI FORMAT

```
bean-validator:label[?options]
```

Where **label** is an arbitrary text value describing the endpoint. You can append query options to the URI in the following format,

?option=value&option=value&...

13.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

13.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

13.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

13.3. COMPONENT OPTIONS

The Bean Validator component supports 8 options, which are listed below.

Name	Description	Default	Type
ignoreXmlConfiguration (producer)	Whether to ignore data from the META-INF/validation.xml file.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
constraintValidatorFactory (advanced)	To use a custom ConstraintValidatorFactory.		ConstraintValidatorFactory
messageInterpolator (advanced)	To use a custom MessageInterpolator.		MessageInterpolator
traversableResolver (advanced)	To use a custom TraversableResolver.		TraversableResolver
validationProviderResolver (advanced)	To use a a custom ValidationProviderResolver.		ValidationProviderResolver

Name	Description	Default	Type
validatorFactory (advanced)	**Autowired** To use a custom ValidatorFactory.		ValidatorFactory

13.4. ENDPOINT OPTIONS

The Bean Validator endpoint is configured using URI syntax:

```
bean-validator:label
```

with the following path and query parameters:

13.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
label (producer)	**Required** Where label is an arbitrary text value describing the endpoint.		String

13.4.2. Query Parameters (8 parameters)

Name	Description	Default	Type
group (producer)	To use a custom validation group.	javax.validation.groups.Default	String
ignoreXmlConfiguration (producer)	Whether to ignore data from the META-INF/validation.xml file.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
constraintValidatorFactory (advanced)	To use a custom ConstraintValidatorFactory.		ConstraintValidatorFactory
messageInterpolator (advanced)	To use a custom MessageInterpolator.		MessageInterpolator
traversableResolver (advanced)	To use a custom TraversableResolver.		TraversableResolver
validationProviderResolver (advanced)	To use a a custom ValidationProviderResolver.		ValidationProviderResolver
validatorFactory (advanced)	To use a custom ValidatorFactory.		ValidatorFactory

13.5. OSGI DEPLOYMENT

To use Hibernate Validator in the OSGi environment use dedicated **ValidationProviderResolver** implementation, just as **org.apache.camel.component.bean.validator.HibernateValidationProviderResolver**. The snippet below demonstrates this approach. You can also use **HibernateValidationProviderResolver**.

Using HibernateValidationProviderResolver

```
from("direct:test").
  to("bean-validator://ValidationProviderResolverTest?
validationProviderResolver=#myValidationProviderResolver");
```

```
<bean id="myValidationProviderResolver"
class="org.apache.camel.component.bean.validator.HibernateValidationProviderResolver"/>
```

If no custom **ValidationProviderResolver** is defined and the validator component has been deployed into the OSGi environment, the **HibernateValidationProviderResolver** will be automatically used.

13.6. EXAMPLE

Assumed we have a java bean with the following annotations

Car.java

```
public class Car {

    @NotNull
    private String manufacturer;
```

```
    @NotNull
    @Size(min = 5, max = 14, groups = OptionalChecks.class)
    private String licensePlate;

    // getter and setter
}
```

and an interface definition for our custom validation group

OptionalChecks.java

```
public interface OptionalChecks {
}
```

with the following Camel route, only the @**NotNull** constraints on the attributes manufacturer and licensePlate will be validated (Camel uses the default group **javax.validation.groups.Default**).

```
from("direct:start")
.to("bean-validator://x")
.to("mock:end")
```

If you want to check the constraints from the group **OptionalChecks**, you have to define the route like this

```
from("direct:start")
.to("bean-validator://x?group=OptionalChecks")
.to("mock:end")
```

If you want to check the constraints from both groups, you have to define a new interface first

AllChecks.java

```
@GroupSequence({Default.class, OptionalChecks.class})
public interface AllChecks {
}
```

and then your route definition should looks like this

```
from("direct:start")
.to("bean-validator://x?group=AllChecks")
.to("mock:end")
```

And if you have to provide your own message interpolator, traversable resolver and constraint validator factory, you have to write a route like this

```
<bean id="myMessageInterpolator" class="my.ConstraintValidatorFactory" />
<bean id="myTraversableResolver" class="my.TraversableResolver" />
<bean id="myConstraintValidatorFactory" class="my.ConstraintValidatorFactory" />
```

```
from("direct:start")
.to("bean-validator://x?group=AllChecks&messageInterpolator=#myMessageInterpolator
&traversableResolver=#myTraversableResolver&constraintValidatorFactory=#myConstraintValidatorFac
```

```
tory")
.to("mock:end")
```

It's also possible to describe your constraints as XML and not as Java annotations. In this case, you have to provide the file **META-INF/validation.xml** which could looks like this

validation.xml

```
<validation-config
    xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration">

    <default-provider>org.hibernate.validator.HibernateValidator</default-provider>
    <message-
interpolator>org.hibernate.validator.engine.ResourceBundleMessageInterpolator</message-
interpolator>
    <traversable-
resolver>org.hibernate.validator.engine.resolver.DefaultTraversableResolver</traversable-resolver>
    <constraint-validator-
factory>org.hibernate.validator.engine.ConstraintValidatorFactoryImpl</constraint-validator-factory>
    <constraint-mapping>/constraints-car.xml</constraint-mapping>

</validation-config>
```

and the **constraints-car.xml** file

constraints-car.xml

```
<constraint-mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/mapping validation-mapping-1.0.xsd"
    xmlns="http://jboss.org/xml/ns/javax/validation/mapping">

    <default-package>org.apache.camel.component.bean.validator</default-package>

    <bean class="CarWithoutAnnotations" ignore-annotations="true">
        <field name="manufacturer">
            <constraint annotation="javax.validation.constraints.NotNull" />
        </field>

        <field name="licensePlate">
            <constraint annotation="javax.validation.constraints.NotNull" />

            <constraint annotation="javax.validation.constraints.Size">
                <groups>
                    <value>org.apache.camel.component.bean.validator.OptionalChecks</value>
                </groups>
                <element name="min">5</element>
                <element name="max">14</element>
            </constraint>
        </field>
    </bean>
</constraint-mappings>
```

Here is the XML syntax for the example route definition for OrderedChecks.

Note that the body should include an instance of a class to validate.

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

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:start"/>
            <to uri="bean-validator://x?
group=org.apache.camel.component.bean.validator.OrderedChecks"/>
        </route>
    </camelContext>
</beans>
```

13.7. SPRING BOOT AUTO-CONFIGURATION

When using bean-validator with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-bean-validator-starter</artifactId>
</dependency>
```

The component supports 9 options, which are listed below.

Name	Description	Default	Type
camel.component.bean-validator.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.bean-validator.constraint-validator-factory	To use a custom ConstraintValidatorFactory. The option is a javax.validation.ConstraintValidatorFactory type.		ConstraintValidatorFactory
camel.component.bean-validator.enabled	Whether to enable auto configuration of the bean-validator component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.bean-validator.ignore-xml-configuration	Whether to ignore data from the META-INF/validation.xml file.	false	Boolean
camel.component.bean-validator.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.bean-validator.message-interpolator	To use a custom MessageInterpolator. The option is a javax.validation.MessageInterpolator type.		MessageInterpolator
camel.component.bean-validator.traversable-resolver	To use a custom TraversableResolver. The option is a javax.validation.TraversableResolver type.		TraversableResolver
camel.component.bean-validator.validation-provider-resolver	To use a a custom ValidationProviderResolver. The option is a javax.validation.ValidationProviderResolver type.		ValidationProviderResolver
camel.component.bean-validator.validator-factory	To use a custom ValidatorFactory. The option is a javax.validation.ValidatorFactory type.		ValidatorFactory

CHAPTER 14. BROWSE

Both producer and consumer are supported

The Browse component provides a simple BrowsableEndpoint which can be useful for testing, visualisation tools or debugging. The exchanges sent to the endpoint are all available to be browsed.

14.1. URI FORMAT

```
browse:someName[?options]
```

Where **someName** can be any string to uniquely identify the endpoint.

14.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

14.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

14.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

14.3. COMPONENT OPTIONS

The Browse component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

14.4. ENDPOINT OPTIONS

The Browse endpoint is configured using URI syntax:

```
browse:name
```

with the following path and query parameters:

14.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** A name which can be any string to uniquely identify the endpoint.		String

14.4.2. Query Parameters (4 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

14.5. SAMPLE

In the route below, we insert a **browse:** component to be able to browse the Exchanges that are passing through:

```
from("activemq:order.in").to("browse:orderReceived").to("bean:processOrder");
```

We can now inspect the received exchanges from within the Java code:

```
private CamelContext context;
```

```
public void inspectReceivedOrders() {
    BrowsableEndpoint browse = context.getEndpoint("browse:orderReceived",
BrowsableEndpoint.class);
    List<Exchange> exchanges = browse.getExchanges();

    // then we can inspect the list of received exchanges from Java
    for (Exchange exchange : exchanges) {
        String payload = exchange.getIn().getBody();
        // do something with payload
    }
}
```

14.6. SPRING BOOT AUTO-CONFIGURATION

When using browse with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-browse-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.browse.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.browse.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.browse.enabled	Whether to enable auto configuration of the browse component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.browse.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 15. CASSANDRA CQL

Both producer and consumer are supported

Apache Cassandra is an open source NoSQL database designed to handle large amounts on commodity hardware. Like Amazon's DynamoDB, Cassandra has a peer-to-peer and master-less architecture to avoid single point of failure and garanty high availability. Like Google's BigTable, Cassandra data is structured using column families which can be accessed through the Thrift RPC API or a SQL-like API called CQL.

NOTE

This component aims at integrating Cassandra 2.0+ using the CQL3 API (not the Thrift API). It's based on Cassandra Java Driver provided by DataStax.

15.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

15.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

15.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

15.2. COMPONENT OPTIONS

The Cassandra CQL component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

15.3. ENDPOINT OPTIONS

The Cassandra CQL endpoint is configured using URI syntax:

```
cql:beanRef:hosts:port/keyspace
```

with the following path and query parameters:

15.3.1. Path Parameters (4 parameters)

Name	Description	Default	Type
beanRef (common)	beanRef is defined using bean:id.		String

Name	Description	Default	Type
hosts (common)	Hostname(s) Cassandra server(s). Multiple hosts can be separated by comma.		String
port (common)	Port number of Cassandra server(s).		Integer
keyspace (common)	Keyspace to use.		String

15.3.2. Query Parameters (30 parameters)

Name	Description	Default	Type
clusterName (common)	Cluster name.		String
consistencyLevel (common)	Consistency level to use. Enum values: • ANY • ONE • TWO • THREE • QUORUM • ALL • LOCAL_ONE • LOCAL_QUORUM • EACH_QUORUM • SERIAL • LOCAL_SERIAL		DefaultConsistencyLevel
cql (common)	CQL query to perform. Can be overridden with the message header with key CamelCqlQuery.		String
datacenter (common)	Datacenter to use.	datacenter1	String
loadBalancingPolicyClass (common)	To use a specific LoadBalancingPolicyClass.		String

Name	Description	Default	Type
password (common)	Password for session authentication.		String
prepareStatements (common)	Whether to use PreparedStatements or regular Statements.	true	boolean
resultSetConversionStrategy (common)	To use a custom class that implements logic for converting ResultSet into message body ALL, ONE, LIMIT_10, LIMIT_100...		ResultSetConversionStrategy
session (common)	To use the Session instance (you would normally not use this option).		CqlSession
username (common)	Username for session authentication.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean

15.4. ENDPOINT CONNECTION SYNTAX

The endpoint can initiate the Cassandra connection or use an existing one.

URI	Description
cql:localhost/keyspace	Single host, default port, usual for testing
cql:host1,host2/keyspace	Multi host, default port
cql:host1,host2:9042/keyspace	Multi host, custom port
cql:host1,host2	Default port and keyspace
cql:bean:sessionRef	Provided Session reference
cql:bean:clusterRef/keyspace	Provided Cluster reference

To fine tune the Cassandra connection (SSL options, pooling options, load balancing policy, retry policy, reconnection policy...), create your own Cluster instance and give it to the Camel endpoint.

15.5. MESSAGES

15.5.1. Incoming Message

The Camel Cassandra endpoint expects a bunch of simple objects (**Object** or **Object[]** or **Collection<Object>**) which will be bound to the CQL statement as query parameters. If message body is null or empty, then CQL query will be executed without binding parameters.

Headers

- **CamelCqlQuery** (optional, **String** or **RegularStatement**)
  CQL query either as a plain String or built using the **QueryBuilder**.

15.5.2. Outgoing Message

The Camel Cassandra endpoint produces one or many a Cassandra Row objects depending on the **resultSetConversionStrategy**:

- **List<Row>** if **resultSetConversionStrategy** is **ALL** or **LIMIT_[0-9]+**
- Single` Row` if **resultSetConversionStrategy** is **ONE**

- Anything else, if **resultSetConversionStrategy** is a custom implementation of the **ResultSetConversionStrategy**

15.6. REPOSITORIES

Cassandra can be used to store message keys or messages for the idempotent and aggregation EIP.

Cassandra might not be the best tool for queuing use cases yet, read Cassandra anti-patterns queues and queue like datasets. It's advised to use LeveledCompaction and a small GC grace setting for these tables to allow tombstoned rows to be removed quickly.

15.7. IDEMPOTENT REPOSITORY

The **NamedCassandraIdempotentRepository** stores messages keys in a Cassandra table like this:

CAMEL_IDEMPOTENT.cql

```
CREATE TABLE CAMEL_IDEMPOTENT (
  NAME varchar,   -- Repository name
  KEY varchar,    -- Message key
  PRIMARY KEY (NAME, KEY)
) WITH compaction = {'class':'LeveledCompactionStrategy'}
  AND gc_grace_seconds = 86400;
```

This repository implementation uses lightweight transactions (also known as Compare and Set) and requires Cassandra 2.0.7+.

Alternatively, the **CassandraIdempotentRepository** does not have a **NAME** column and can be extended to use a different data model.

Option	Default	Description
table	**CAMEL_IDEMPOTENT**	Table name
pkColumns	**NAME**,` KEY`	Primary key columns
name		Repository name, value used for **NAME** column
ttl		Key time to live
writeConsistencyLevel		Consistency level used to insert/delete key: **ANY**, **ONE**, **TWO**, **QUORUM**, **LOCAL_QUORUM**...
readConsistencyLevel		Consistency level used to read/check key: **ONE**, **TWO**, **QUORUM**, **LOCAL_QUORUM**...

15.8. AGGREGATION REPOSITORY

The **NamedCassandraAggregationRepository** stores exchanges by correlation key in a Cassandra table like this:

CAMEL_AGGREGATION.cql

```
CREATE TABLE CAMEL_AGGREGATION (
  NAME varchar,        -- Repository name
  KEY varchar,         -- Correlation id
  EXCHANGE_ID varchar, -- Exchange id
  EXCHANGE blob,       -- Serialized exchange
  PRIMARY KEY (NAME, KEY)
) WITH compaction = {'class':'LeveledCompactionStrategy'}
  AND gc_grace_seconds = 86400;
```

Alternatively, the **CassandraAggregationRepository** does not have a **NAME** column and can be extended to use a different data model.

Option	Default	Description
table	**CAMEL_AGGREGATION**	Table name
pkColumns	**NAME**,**KEY**	Primary key columns
exchangeIdColumn	**EXCHANGE_ID**	Exchange Id column
exchangeColumn	**EXCHANGE**	Exchange content column
name		Repository name, value used for **NAME** column
ttl		Exchange time to live
writeConsistencyLevel		Consistency level used to insert/delete exchange: **ANY**, **ONE**, **TWO**, **QUORUM**, **LOCAL_QUORUM**...
readConsistencyLevel		Consistency level used to read/check exchange: **ONE**, **TWO**, **QUORUM**, **LOCAL_QUORUM**...

15.9. EXAMPLES

To insert something on a table you can use the following code:

```
String CQL = "insert into camel_user(login, first_name, last_name) values (?, ?, ?)";
from("direct:input")
    .to("cql://localhost/camel_ks?cql=" + CQL);
```

At this point you should be able to insert data by using a list as body

```
Arrays.asList("davsclaus", "Claus", "Ibsen")
```

The same approach can be used for updating or querying the table.

15.10. SPRING BOOT AUTO-CONFIGURATION

When using cql with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-cassandraql-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.cql.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.cql.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.cql.enabled	Whether to enable auto configuration of the cql component. This is enabled by default.		Boolean
camel.component.cql.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 16. CONTROL BUS

Only producer is supported

The Control Bus from the EIP patterns allows for the integration system to be monitored and managed from within the framework.

Use a Control Bus to manage an enterprise integration system. The Control Bus uses the same messaging mechanism used by the application data, but uses separate channels to transmit data that is relevant to the management of components involved in the message flow.

In Camel you can manage and monitor using JMX, or by using a Java API from the **CamelContext**, or from the **org.apache.camel.api.management** package, or use the event notifier which has an example here.

The ControlBus component provides easy management of Camel applications based on the Control Bus EIP pattern. For example, by sending a message to an Endpoint you can control the lifecycle of routes, or gather performance statistics.

```
controlbus:command[?options]
```

Where **command** can be any string to identify which type of command to use.

16.1. COMMANDS

Command	Description
route	To control routes using the **routeId** and **action** parameter.
language	Allows you to specify a to use for evaluating the message body. If there is any result from the evaluation, then the result is put in the message body.

16.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

16.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

16.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

16.3. COMPONENT OPTIONS

The Control Bus component supports 2 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

16.4. ENDPOINT OPTIONS

The Control Bus endpoint is configured using URI syntax:

```
controlbus:command:language
```

with the following path and query parameters:

16.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
command (producer)	**Required** Command can be either route or language. Enum values: • route • language		String
language (producer)	Allows you to specify the name of a Language to use for evaluating the message body. If there is any result from the evaluation, then the result is put in the message body. Enum values: • bean • constant • el • exchangeProperty • file • groovy • header • jsonpath • mvel • ognl • ref • simple • spel • sql • terser • tokenize • xpath • xquery • xtokenize		Language

16.4.1.1. Query Parameters (6 parameters)

Name	Description	Default	Type
action (producer)	To denote an action that can be either: start, stop, or status. To either start or stop a route, or to get the status of the route as output in the message body. You can use suspend and resume from Camel 2.11.1 onwards to either suspend or resume a route. And from Camel 2.11.1 onwards you can use stats to get performance statics returned in XML format; the routeId option can be used to define which route to get the performance stats for, if routeId is not defined, then you get statistics for the entire CamelContext. The restart action will restart the route. Enum values: • start • stop • suspend • resume • restart • status • stats		String
async (producer)	Whether to execute the control bus task asynchronously. Important: If this option is enabled, then any result from the task is not set on the Exchange. This is only possible if executing tasks synchronously.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
loggingLevel (producer)	Logging level used for logging when task is done, or if any exceptions occurred during processing the task. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	INFO	LoggingLevel
restartDelay (producer)	The delay in millis to use when restarting a route.	1000	int
routeId (producer)	To specify a route by its id. The special keyword current indicates the current route.		String

16.5. USING ROUTE COMMAND

The route command allows you to do common tasks on a given route very easily, for example to start a route, you can send an empty message to this endpoint:

```
template.sendBody("controlbus:route?routeId=foo&action=start", null);
```

To get the status of the route, you can do:

```
String status = template.requestBody("controlbus:route?routeId=foo&action=status", null,
String.class);
```

16.6. GETTING PERFORMANCE STATISTICS

This requires JMX to be enabled (is by default) then you can get the performance statistics per route, or for the CamelContext. For example to get the statistics for a route named foo, we can do:

```
String xml = template.requestBody("controlbus:route?routeId=foo&action=stats", null, String.class);
```

The returned statistics is in XML format. Its the same data you can get from JMX with the **dumpRouteStatsAsXml** operation on the **ManagedRouteMBean**.

To get statistics for the entire CamelContext you just omit the routeId parameter as shown below:

```
String xml = template.requestBody("controlbus:route?action=stats", null, String.class);
```

16.7. USING SIMPLE LANGUAGE

You can use the Simple language with the control bus, for example to stop a specific route, you can send a message to the **"controlbus:language:simple"** endpoint containing the following message:

```
template.sendBody("controlbus:language:simple",
"${camelContext.getRouteController().stopRoute('myRoute')}");
```

As this is a void operation, no result is returned. However, if you want the route status you can do:

```
String status = template.requestBody("controlbus:language:simple",
"${camelContext.getRouteStatus('myRoute')}", String.class);
```

It's easier to use the **route** command to control lifecycle of routes. The **language** command allows you to execute a language script that has stronger powers such as Groovy or to some extend the Simple language.

For example to shutdown Camel itself you can do:

```
template.sendBody("controlbus:language:simple?async=true", "${camelContext.stop()}");
```

We use **async=true** to stop Camel asynchronously as otherwise we would be trying to stop Camel while it was in-flight processing the message we sent to the control bus component.

NOTE

You can also use other languages such as Groovy, etc.

16.8. SPRING BOOT AUTO-CONFIGURATION

When using controlbus with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-controlbus-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type
camel.component.controlbus.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.controlbus.enabled	Whether to enable auto configuration of the controlbus component. This is enabled by default.		Boolean
camel.component.controlbus.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 17. CRON

Only consumer is supported

The Cron component is a generic interface component that allows triggering events at specific time interval specified using the Unix cron syntax (e.g. **0/2 * * * * ?** to trigger an event every two seconds).

Being an interface component, the Cron component does not contain a default implementation, instead it requires that the users plug the implementation of their choice.

The following standard Camel components support the Cron endpoints:

- Camel-quartz
- Camel-spring

The Cron component is also supported in **Camel K**, which can use the Kubernetes scheduler to trigger the routes when required by the cron expression. Camel K does not require additional libraries to be plugged when using cron expressions compatible with Kubernetes cron syntax.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cron</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Additional libraries may be needed in order to plug a specific implementation.

17.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

17.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

17.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

17.2. COMPONENT OPTIONS

The Cron component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
cronService (advanced)	The id of the CamelCronService to use when multiple implementations are provided.		String

17.3. ENDPOINT OPTIONS

The Cron endpoint is configured using URI syntax:

```
cron:name
```

with the following path and query parameters:

17.3.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (consumer)	**Required** The name of the cron trigger.		String

17.3.2. Query Parameters (4 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
schedule (consumer)	**Required** A cron expression that will be used to generate events.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

17.4. USAGE

The component can be used to trigger events at specified times, as in the following example:

```
from("cron:tab?schedule=0/1+*+*+*+*+?")
.setBody().constant("event")
.log("${body}");
```

The schedule expression **0/3+10+*+?** can be also written as **0/3 10 * * * ?** and triggers an event every three seconds only in the tenth minute of each hour.

Parts in the schedule expression means (in order):

- Seconds (optional)
- Minutes
- Hours
- Day of month
- Month
- Day of week
- Year (optional)

Schedule expressions can be made of 5 to 7 parts. When expressions are composed of 6 parts, the first items is the "seconds" part (and year is considered missing).

Other valid examples of schedule expressions are:

- **0/2 * * * ?** (5 parts, an event every two minutes)
- **0 0/2 * * * MON-FRI 2030** (7 parts, an event every two minutes only in year 2030)

Routes can also be written using the XML DSL.

```
<route>
  <from uri="cron:tab?schedule=0/1+*+*+*+*+?"/>
  <setBody>
    <constant>event</constant>
  </setBody>
  <to uri="log:info"/>
</route>
```

17.5. SPRING BOOT AUTO-CONFIGURATION

When using cron with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-cron-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type

Name	Description	Default	Type
camel.component.cron.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.cron.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.cron.cron-service	The id of the CamelCronService to use when multiple implementations are provided.		String
camel.component.cron.enabled	Whether to enable auto configuration of the cron component. This is enabled by default.		Boolean

CHAPTER 18. CXF

Both producer and consumer are supported

The CXF component provides integration with Apache CXF for connecting to JAX-WS services hosted in CXF.

TIP

When using CXF in streaming modes (see DataFormat option), then also read about Stream caching.

Maven users must add the following dependency to their **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-cxf-soap</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

18.1. URI FORMAT

There are two URI formats for this endpoint: **cxfEndpoint** and **someAddress**.

```
cxf:bean:cxfEndpoint[?options]
```

Where **cxfEndpoint** represents a bean ID that references a bean in the Spring bean registry. With this URI format, most of the endpoint details are specified in the bean definition.

```
cxf://someAddress[?options]
```

Where **someAddress** specifies the CXF endpoint's address. With this URI format, most of the endpoint details are specified using options.

For either style above, you can append options to the URI as follows:

```
cxf:bean:cxfEndpoint?wsdlURL=wsdl/hello_world.wsdl&dataFormat=PAYLOAD
```

18.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

18.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

18.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a *type safe* way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

18.3. COMPONENT OPTIONS

The CXF component supports 6 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
allowStreaming (advanced)	This option controls whether the CXF component, when running in PAYLOAD mode, will DOM parse the incoming messages into DOM Elements or keep the payload as a javax.xml.transform.Source object that would allow streaming in some cases.		Boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean

18.4. ENDPOINT OPTIONS

The CXF endpoint is configured using URI syntax:

```
cxf:beanId:address
```

with the following path and query parameters:

18.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
beanId (common)	To lookup an existing configured CxfEndpoint. Must used bean: as prefix.		String
address (service)	The service publish address.		String

18.4.2. Query Parameters (35 parameters)

Name	Description	Default	Type
dataFormat (common)	The data type messages supported by the CXF endpoint. Enum values: • PAYLOAD • RAW • MESSAGE • CXF_MESSAGE • POJO	POJO	DataFormat
wrappedStyle (common)	The WSDL style that describes how parameters are represented in the SOAP body. If the value is false, CXF will chose the document-literal unwrapped style, If the value is true, CXF will chose the document-literal wrapped style.		Boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
cookieHandler (producer)	Configure a cookie handler to maintain a HTTP session.		CookieHandler

Name	Description	Default	Type
defaultOperationName (producer)	This option will set the default operationName that will be used by the CxfProducer which invokes the remote service.		String
defaultOperationNamespace (producer)	This option will set the default operationNamespace that will be used by the CxfProducer which invokes the remote service.		String
hostnameVerifier (producer)	The hostname verifier to be used. Use the # notation to reference a HostnameVerifier from the registry.		HostnameVerifier
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
sslContextParameters (producer)	The Camel SSL setting reference. Use the # notation to reference the SSL Context.		SSLContextParameters
wrapped (producer)	Which kind of operation that CXF endpoint producer will invoke.	false	boolean
synchronous (producer (advanced))	Sets whether synchronous processing should be strictly used.	false	boolean
allowStreaming (advanced)	This option controls whether the CXF component, when running in PAYLOAD mode, will DOM parse the incoming messages into DOM Elements or keep the payload as a javax.xml.transform.Source object that would allow streaming in some cases.		Boolean
bus (advanced)	To use a custom configured CXF Bus.		Bus
continuationTimeout (advanced)	This option is used to set the CXF continuation timeout which could be used in CxfConsumer by default when the CXF server is using Jetty or Servlet transport.	30000	long

Name	Description	Default	Type
cxfBinding (advanced)	To use a custom CxfBinding to control the binding between Camel Message and CXF Message.		CxfBinding
cxfConfigurer (advanced)	This option could apply the implementation of org.apache.camel.component.cxf.CxfEndpointConfigurer which supports to configure the CXF endpoint in programmatic way. User can configure the CXF server and client by implementing configure{ServerClient} method of CxfEndpointConfigurer.		CxfConfigurer
defaultBus (advanced)	Will set the default bus when CXF endpoint create a bus by itself.	false	boolean
headerFilterStrategy (advanced)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
mergeProtocolHeaders (advanced)	Whether to merge protocol headers. If enabled then propagating headers between Camel and CXF becomes more consistent and similar. For more details see CAMEL-6393.	false	boolean
mtomEnabled (advanced)	To enable MTOM (attachments). This requires to use POJO or PAYLOAD data format mode.	false	boolean
properties (advanced)	To set additional CXF options using the key/value pairs from the Map. For example to turn on stacktraces in SOAP faults, properties.faultStackTraceEnabled=true.		Map
skipPayloadMessagePartCheck (advanced)	Sets whether SOAP message validation should be disabled.	false	boolean
loggingFeatureEnabled (logging)	This option enables CXF Logging Feature which writes inbound and outbound SOAP messages to log.	false	boolean
loggingSizeLimit (logging)	To limit the total size of number of bytes the logger will output when logging feature has been enabled and -1 for no limit.	49152	int
skipFaultLogging (logging)	This option controls whether the PhaseInterceptorChain skips logging the Fault that it catches.	false	boolean
password (security)	This option is used to set the basic authentication information of password for the CXF client.		String

Name	Description	Default	Type
username (security)	This option is used to set the basic authentication information of username for the CXF client.		String
bindingId (service)	The bindingId for the service model to use.		String
portName (service)	The endpoint name this service is implementing, it maps to the wsdl:portname. In the format of ns:PORT_NAME where ns is a namespace prefix valid at this scope.		String
publishedEndpointUrl (service)	This option can override the endpointUrl that published from the WSDL which can be accessed with service address url plus wsd.		String
serviceClass (service)	The class name of the SEI (Service Endpoint Interface) class which could have JSR181 annotation or not.		Class
serviceName (service)	The service name this service is implementing, it maps to the wsdl:servicename.		String
wsdlURL (service)	The location of the WSDL. Can be on the classpath, file system, or be hosted remotely.		String

The **serviceName** and **portName** are QNames, so if you provide them be sure to prefix them with their {namespace} as shown in the examples above.

18.4.3. Descriptions of the dataformats

In Apache Camel, the Camel CXF component is the key to integrating routes with Web services. You can use the Camel CXF component to create a CXF endpoint, which can be used in either of the following ways:

- **Consumer** – (at the start of a route) represents a Web service instance, which integrates with the route. The type of payload injected into the route depends on the value of the endpoint's dataFormat option.
- **Producer** – (at other points in the route) represents a WS client proxy, which converts the current exchange object into an operation invocation on a remote Web service. The format of the current exchange must match the endpoint's dataFormat setting.

DataFormat	Description
POJO	POJOs (Plain old Java objects) are the Java parameters to the method being invoked on the target server. Both Protocol and Logical JAX-WS handlers are supported.

DataFormat	Description
PAYLOAD	**PAYLOAD** is the message payload (the contents of the**soap:body**) after message configuration in the CXF endpoint is applied. Only Protocol JAX-WS handler is supported. Logical JAX-WS handler is not supported.
RAW	**RAW** mode provides the raw message stream that is received from the transport layer. It is not possible to touch or change the stream, some of the CXF interceptors will be removed if you are using this kind of DataFormat, so you can't see any soap headers after the camel-cxf consumer. JAX-WS handler is not supported.
CXF_MESSAGE	**CXF_MESSAGE** allows for invoking the full capabilities of CXF interceptors by converting the message from the transport layer into a raw SOAP message

You can determine the data format mode of an exchange by retrieving the exchange property, **CamelCXFDataFormat**. The exchange key constant is defined in **org.apache.camel.component.cxf.common.message.CxfConstants.DATA_FORMAT_PROPERTY**.

18.4.4. How to enable CXF's LoggingOutInterceptor in RAW mode

CXF's **LoggingOutInterceptor** outputs outbound message that goes on the wire to logging system (Java Util Logging). Since the **LoggingOutInterceptor** is in **PRE_STREAM** phase (but **PRE_STREAM** phase is removed in **RAW** mode), you have to configure **LoggingOutInterceptor** to be run during the **WRITE** phase. The following is an example.

```
@Bean
public CxfEndpoint serviceEndpoint(LoggingOutInterceptor loggingOutInterceptor) {
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setAddress("http://localhost:" + port
                        + "/services" + SERVICE_ADDRESS);
    cxfEndpoint.setServiceClass(org.apache.camel.component.cxf.HelloService.class);
    Map<String, Object> properties = new HashMap<String, Object>();
    properties.put("dataFormat", "RAW");
    cxfEndpoint.setProperties(properties);
    cxfEndpoint.getOutInterceptors().add(loggingOutInterceptor);
    return cxfEndpoint;
}


@Bean
public LoggingOutInterceptor loggingOutInterceptor() {
    LoggingOutInterceptor logger = new LoggingOutInterceptor("write");
    return logger;
}
```

18.4.5. Description of relayHeaders option

There are *in-band* and *out-of-band* on-the-wire headers from the perspective of a JAXWS WSDL-first developer.

The *in-band* headers are headers that are explicitly defined as part of the WSDL binding contract for an endpoint such as SOAP headers.

The *out-of-band* headers are headers that are serialized over the wire, but are not explicitly part of the WSDL binding contract.

Headers relaying/filtering is bi-directional.

When a route has a CXF endpoint and the developer needs to have on-the-wire headers, such as SOAP headers, be relayed along the route to be consumed say by another JAXWS endpoint, then **relayHeaders** should be set to **true**, which is the default value.

18.4.6. Available only in POJO mode

The **relayHeaders=true** expresses an intent to relay the headers. The actual decision on whether a given header is relayed is delegated to a pluggable instance that implements the **MessageHeadersRelay** interface. A concrete implementation of **MessageHeadersRelay** will be consulted to decide if a header needs to be relayed or not. There is already an implementation of **SoapMessageHeadersRelay** which binds itself to well-known SOAP name spaces. Currently only out-of-band headers are filtered, and in-band headers will always be relayed when **relayHeaders=true**. If there is a header on the wire whose name space is unknown to the runtime, then a fall back **DefaultMessageHeadersRelay** will be used, which simply allows all headers to be relayed.

The **relayHeaders=false** setting specifies that all headers in-band and out-of-band should be dropped.

You can plugin your own **MessageHeadersRelay** implementations overriding or adding additional ones to the list of relays. In order to override a preloaded relay instance just make sure that your **MessageHeadersRelay** implementation services the same name spaces as the one you looking to override. Also note, that the overriding relay has to service all of the name spaces as the one you looking to override, or else a runtime exception on route start up will be thrown as this would introduce an ambiguity in name spaces to relay instance mappings.

```
<cxf:cxfEndpoint ...>
  <cxf:properties>
    <entry key="org.apache.camel.cxf.message.headers.relays">
      <list>
        <ref bean="customHeadersRelay"/>
      </list>
    </entry>
  </cxf:properties>
 </cxf:cxfEndpoint>
 <bean id="customHeadersRelay"
class="org.apache.camel.component.cxf.soap.headers.CustomHeadersRelay"/>
```

Take a look at the tests that show how you would be able to relay/drop headers here:

CxfMessageHeadersRelayTest

- **POJO** and **PAYLOAD** modes are supported. In **POJO** mode, only out-of-band message headers are available for filtering as the in-band headers have been processed and removed from header list by CXF. The in-band headers are incorporated into the **MessageContentList** in POJO mode. The **camel-cxf** component does make any attempt to remove the in-band headers from the **MessageContentList**. If filtering of in-band headers is required, please use **PAYLOAD** mode or plug in a (pretty straightforward) CXF interceptor/JAXWS Handler to the CXF endpoint.

- The Message Header Relay mechanism has been merged into **CxfHeaderFilterStrategy**. The **relayHeaders** option, its semantics, and default value remain the same, but it is a property of **CxfHeaderFilterStrategy**. Here is an example of configuring it.

```
@Bean
public HeaderFilterStrategy dropAllMessageHeadersStrategy() {
    CxfHeaderFilterStrategy headerFilterStrategy = new CxfHeaderFilterStrategy();
    headerFilterStrategy.setRelayHeaders(false);
    return headerFilterStrategy;
}
```

Then, your endpoint can reference the **CxfHeaderFilterStrategy**.

```
@Bean
public CxfEndpoint routerNoRelayEndpoint(HeaderFilterStrategy dropAllMessageHeadersStrategy) {
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setServiceClass(org.apache.camel.component.cxf.soap.headers.HeaderTester.class);
    cxfEndpoint.setAddress("/CxfMessageHeadersRelayTest/HeaderService/routerNoRelayEndpoint");
    cxfEndpoint.setWsdlURL("soap_header.wsdl");
    cxfEndpoint.setEndpointNameAsQName(
        QName.valueOf("{http://apache.org/camel/component/cxf/soap/headers}SoapPortNoRelay"));
    cxfEndpoint.setServiceNameAsQName(SERVICENAME);
    Map<String, Object> properties = new HashMap<String, Object>();
    properties.put("dataFormat", "PAYLOAD");
    cxfEndpoint.setProperties(properties);
    cxfEndpoint.setHeaderFilterStrategy(dropAllMessageHeadersStrategy);
    return cxfEndpoint;
}

@Bean
public CxfEndpoint serviceNoRelayEndpoint(HeaderFilterStrategy dropAllMessageHeadersStrategy)
{
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setServiceClass(org.apache.camel.component.cxf.soap.headers.HeaderTester.class);
    cxfEndpoint.setAddress("http://localhost:" + port +
"/services/CxfMessageHeadersRelayTest/HeaderService/routerNoRelayEndpointBackend");
    cxfEndpoint.setWsdlURL("soap_header.wsdl");
    cxfEndpoint.setEndpointNameAsQName(
        QName.valueOf("{http://apache.org/camel/component/cxf/soap/headers}SoapPortNoRelay"));
    cxfEndpoint.setServiceNameAsQName(SERVICENAME);
    Map<String, Object> properties = new HashMap<String, Object>();
    properties.put("dataFormat", "PAYLOAD");
    cxfEndpoint.setProperties(properties);
    cxfEndpoint.setHeaderFilterStrategy(dropAllMessageHeadersStrategy);
    return cxfEndpoint;
}
```

Then configure the route as follows:

```
rom("cxf:bean:routerNoRelayEndpoint")
    .to("cxf:bean:serviceNoRelayEndpoint");
```

- The **MessageHeadersRelay** interface has changed slightly and has been renamed to **MessageHeaderFilter**. It is a property of **CxfHeaderFilterStrategy**. Here is an example of configuring user defined Message Header Filters:

```
@Bean
public HeaderFilterStrategy customMessageFilterStrategy() {
    CxfHeaderFilterStrategy headerFilterStrategy = new CxfHeaderFilterStrategy();
    List<MessageHeaderFilter> headerFilterList = new ArrayList<MessageHeaderFilter>();
    headerFilterList.add(new SoapMessageHeaderFilter());
    headerFilterList.add(new CustomHeaderFilter());
    headerFilterStrategy.setMessageHeaderFilters(headerFilterList);
    return headerFilterStrategy;
}
```

- In addition to **relayHeaders**, the following properties can be configured in **CxfHeaderFilterStrategy**.

Name	Required	Description
relayHeaders	No	All message headers will be processed by Message Header Filters *Type*: **boolean** *Default*: **true**
relayAllMessageHeaders	No	All message headers will be propagated (without processing by Message Header Filters) *Type*: **boolean***Default*: **false**
allowFilterNamespaceClash	No	If two filters overlap in activation namespace, the property control how it should be handled. If the value is **true**, last one wins. If the value is **false**, it will throw an exception *Type*: **boolean** *Default*: **false**

18.5. CONFIGURE THE CXF ENDPOINTS WITH SPRING

You can configure the CXF endpoint with the Spring configuration file shown below, and you can also embed the endpoint into the **camelContext** tags. When you are invoking the service endpoint, you can set the **operationName** and **operationNamespace** headers to explicitly state which operation you are calling.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://camel.apache.org/schema/cxf/jaxws"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/cxf/jaxws http://camel.apache.org/schema/cxf/jaxws/camel-
cxf.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">
    <cxf:cxfEndpoint id="routerEndpoint" address="http://localhost:9003/CamelContext/RouterPort"
          serviceClass="org.apache.hello_world_soap_http.GreeterImpl"/>
    <cxf:cxfEndpoint id="serviceEndpoint" address="http://localhost:9000/SoapContext/SoapPort"
          wsdlURL="testutils/hello_world.wsdl"
          serviceClass="org.apache.hello_world_soap_http.Greeter"
          endpointName="s:SoapPort"
          serviceName="s:SOAPService"
       xmlns:s="http://apache.org/hello_world_soap_http" />
    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
```

```
    <route>
      <from uri="cxf:bean:routerEndpoint" />
      <to uri="cxf:bean:serviceEndpoint" />
    </route>
  </camelContext>
</beans>
```

Be sure to include the JAX-WS **schemaLocation** attribute specified on the root beans element. This allows CXF to validate the file and is required. Also note the namespace declarations at the end of the **<cxf:cxfEndpoint/>** tag. These declarations are required because the combined **{namespace}localName** syntax is presently not supported for this tag's attribute values.

The **cxf:cxfEndpoint** element supports many additional attributes:

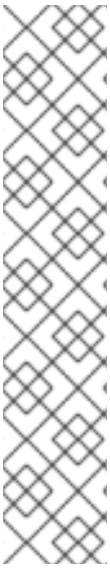
Name	Value
PortName	The endpoint name this service is implementing, it maps to the **wsdl:port@name**. In the format of**ns:PORT_NAME** where **ns** is a namespace prefix valid at this scope.
serviceName	The service name this service is implementing, it maps to the **wsdl:service@name**. In the format of**ns:SERVICE_NAME** where **ns** is a namespace prefix valid at this scope.
wsdlURL	The location of the WSDL. Can be on the classpath, file system, or be hosted remotely.
bindingId	The **bindingId** for the service model to use.
address	The service publish address.
bus	The bus name that will be used in the JAX-WS endpoint.
serviceClass	The class name of the SEI (Service Endpoint Interface) class which could have JSR181 annotation or not.

It also supports many child elements:

Name	Value
cxf:inInterceptors	The incoming interceptors for this endpoint. A list of **<bean>** or **<ref>**.
cxf:inFaultInterceptors	The incoming fault interceptors for this endpoint. A list of **<bean>** or **<ref>**.
cxf:outInterceptors	The outgoing interceptors for this endpoint. A list of **<bean>** or **<ref>**.
cxf:outFaultInterceptors	The outgoing fault interceptors for this endpoint. A list of **<bean>** or **<ref>**.

Name	Value
cxf:properties	A properties map which should be supplied to the JAX-WS endpoint. See below.
cxf:handlers	A JAX-WS handler list which should be supplied to the JAX-WS endpoint. See below.
cxf:dataBinding	You can specify the which **DataBinding** will be use in the endpoint. This can be supplied using the Spring **<bean class="MyDataBinding"/>** syntax.
cxf:binding	You can specify the **BindingFactory** for this endpoint to use. This can be supplied using the Spring **<bean class="MyBindingFactory"/>** syntax.
cxf:features	The features that hold the interceptors for this endpoint. A list of beans or refs
cxf:schemaLocations	The schema locations for endpoint to use. A list of schemaLocations
cxf:serviceFactory	The service factory for this endpoint to use. This can be supplied using the Spring **<bean class="MyServiceFactory"/>** syntax

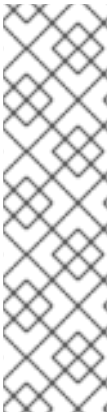
You can find more advanced examples that show how to provide interceptors, properties and handlers on the CXF JAX-WS Configuration page.



NOTE

You can use cxf:properties to set the camel-cxf endpoint's dataFormat and setDefaultBus properties from spring configuration file.

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/router"
    serviceClass="org.apache.camel.component.cxf.HelloService"
    endpointName="s:PortName"
    serviceName="s:ServiceName"
    xmlns:s="http://www.example.com/test">
    <cxf:properties>
      <entry key="dataFormat" value="RAW"/>
      <entry key="setDefaultBus" value="true"/>
    </cxf:properties>
  </cxf:cxfEndpoint>
```



NOTE

In SpringBoot, you can use Spring XML files to configure **camel-cxf** and use code similar to the following example to create XML configured beans:

```
@ImportResource({
  "classpath:spring-configuration.xml"
})
```

However, the use of Java code configured beans (as shown in other examples) is best practice in SpringBoot.

18.6. HOW TO MAKE THE CAMEL-CXF COMPONENT USE LOG4J INSTEAD OF JAVA.UTIL.LOGGING

CXF's default logger is **java.util.logging**. If you want to change it to log4j, proceed as follows. Create a file, in the classpath, named **META-INF/cxf/org.apache.cxf.logger**. This file should contain the fully-qualified name of the class, **org.apache.cxf.common.logging.Log4jLogger**, with no comments, on a single line.

18.7. HOW TO LET CAMEL-CXF RESPONSE START WITH XML PROCESSING INSTRUCTION

If you are using some SOAP client such as PHP, you will get this kind of error, because CXF doesn't add the XML processing instruction **<?xml version="1.0" encoding="utf-8"?>**:

```
Error:sendSms: SoapFault exception: [Client] looks like we got no XML document in [...]
```

To resolve this issue, you just need to tell StaxOutInterceptor to write the XML start document for you, as in the WriteXmlDeclarationInterceptor below:

```
public class WriteXmlDeclarationInterceptor extends AbstractPhaseInterceptor<SoapMessage> {
    public WriteXmlDeclarationInterceptor() {
        super(Phase.PRE_STREAM);
        addBefore(StaxOutInterceptor.class.getName());
    }

    public void handleMessage(SoapMessage message) throws Fault {
        message.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
    }

}
```

As an alternative you can add a message header for it as demonstrated in CxfConsumerTest:

```
// set up the response context which force start document
Map<String, Object> map = new HashMap<String, Object>();
map.put("org.apache.cxf.stax.force-start-document", Boolean.TRUE);
exchange.getOut().setHeader(Client.RESPONSE_CONTEXT, map);
```

18.8. HOW TO OVERRIDE THE CXF PRODUCER ADDRESS FROM MESSAGE HEADER

The **camel-cxf** producer supports to override the target service address by setting a message header **CamelDestinationOverrideUrl**.

```
// set up the service address from the message header to override the setting of CXF endpoint
exchange.getIn().setHeader(Exchange.DESTINATION_OVERRIDE_URL,
constant(getServiceAddress()));
```

18.9. HOW TO CONSUME A MESSAGE FROM A CAMEL-CXF ENDPOINT IN POJO DATA FORMAT

The **camel-cxf** endpoint consumer POJO data format is based on the CXF invoker, so the message header has a property with the name of **CxfConstants.OPERATION_NAME** and the message body is a list of the SEI method parameters.

Consider the PersonProcessor example code:

```
public class PersonProcessor implements Processor {

    private static final Logger LOG = LoggerFactory.getLogger(PersonProcessor.class);

    @Override
    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        LOG.info("processing exchange in camel");

        BindingOperationInfo boi = (BindingOperationInfo)
exchange.getProperty(BindingOperationInfo.class.getName());
        if (boi != null) {
            LOG.info("boi.isUnwrapped" + boi.isUnwrapped());
        }
        // Get the parameters list which element is the holder.
        MessageContentsList msgList = (MessageContentsList) exchange.getIn().getBody();
        Holder<String> personId = (Holder<String>) msgList.get(0);
        Holder<String> ssn = (Holder<String>) msgList.get(1);
        Holder<String> name = (Holder<String>) msgList.get(2);

        if (personId.value == null || personId.value.length() == 0) {
            LOG.info("person id 123, so throwing exception");
            // Try to throw out the soap fault message
            org.apache.camel.wsdl_first.types.UnknownPersonFault personFault
                    = new org.apache.camel.wsdl_first.types.UnknownPersonFault();
            personFault.setPersonId("");
            org.apache.camel.wsdl_first.UnknownPersonFault fault
                    = new org.apache.camel.wsdl_first.UnknownPersonFault("Get the null value of person
name", personFault);
            exchange.getMessage().setBody(fault);
            return;
        }

        name.value = "Bonjour";
        ssn.value = "123";
```

```
        LOG.info("setting Bonjour as the response");
        // Set the response message, first element is the return value of the operation,
        // the others are the holders of method parameters
        exchange.getMessage().setBody(new Object[] { null, personId, ssn, name });
    }

}
```

18.10. HOW TO PREPARE THE MESSAGE FOR THE CAMEL-CXF ENDPOINT IN POJO DATA FORMAT

The **camel-cxf** endpoint producer is based on the CXF client API. First you need to specify the operation name in the message header, then add the method parameters to a list, and initialize the message with this parameter list. The response message's body is a messageContentsList, you can get the result from that list.

If you don't specify the operation name in the message header, **CxfProducer** will try to use the **defaultOperationName** from **CxfEndpoint**, if there is no **defaultOperationName** set on **CxfEndpoint**, it will pick up the first operationName from the Operation list.

If you want to get the object array from the message body, you can get the body using **message.getBody(Object[].class)**, as shown in CxfProducerRouterTest.testInvokingSimpleServerWithParams:

```
Exchange senderExchange = new DefaultExchange(context, ExchangePattern.InOut);
final List<String> params = new ArrayList<>();
// Prepare the request message for the camel-cxf procedure
params.add(TEST_MESSAGE);
senderExchange.getIn().setBody(params);
senderExchange.getIn().setHeader(CxfConstants.OPERATION_NAME, ECHO_OPERATION);

Exchange exchange = template.send("direct:EndpointA", senderExchange);

org.apache.camel.Message out = exchange.getMessage();
// The response message's body is an MessageContentsList which first element is the return value of
the operation,
// If there are some holder parameters, the holder parameter will be filled in the reset of List.
// The result will be extract from the MessageContentsList with the String class type
MessageContentsList result = (MessageContentsList) out.getBody();
LOG.info("Received output text: " + result.get(0));
Map<String, Object> responseContext = CastUtils.cast((Map<?, ?>)
out.getHeader(Client.RESPONSE_CONTEXT));
assertNotNull(responseContext);
assertEquals("UTF-8", responseContext.get(org.apache.cxf.message.Message.ENCODING),
        "We should get the response context here");
assertEquals("echo " + TEST_MESSAGE, result.get(0), "Reply body on Camel is wrong");
```

18.11. HOW TO DEAL WITH THE MESSAGE FOR A CAMEL-CXF ENDPOINT IN PAYLOAD DATA FORMAT

PAYLOAD means that you process the payload from the SOAP envelope as a native CxfPayload. **Message.getBody()** will return a **org.apache.camel.component.cxf.CxfPayload** object, with getters for SOAP message headers and the SOAP body.

See CxfConsumerPayloadTest:

```
protected RouteBuilder createRouteBuilder() {
    return new RouteBuilder() {
        public void configure() {
            from(simpleEndpointURI + "&dataFormat=PAYLOAD").to("log:info").process(new Processor()
{
                @SuppressWarnings("unchecked")
                public void process(final Exchange exchange) throws Exception {
                    CxfPayload<SoapHeader> requestPayload =
exchange.getIn().getBody(CxfPayload.class);
                    List<Source> inElements = requestPayload.getBodySources();
                    List<Source> outElements = new ArrayList<>();
                    // You can use a customer toStringConverter to turn a CxfPayLoad message into String
as you want
                    String request = exchange.getIn().getBody(String.class);
                    XmlConverter converter = new XmlConverter();
                    String documentString = ECHO_RESPONSE;

                    Element in = new XmlConverter().toDOMElement(inElements.get(0));
                    // Just check the element namespace
                    if (!in.getNamespaceURI().equals(ELEMENT_NAMESPACE)) {
                        throw new IllegalArgumentException("Wrong element namespace");
                    }
                    if (in.getLocalName().equals("echoBoolean")) {
                        documentString = ECHO_BOOLEAN_RESPONSE;
                        checkRequest("ECHO_BOOLEAN_REQUEST", request);
                    } else {
                        documentString = ECHO_RESPONSE;
                        checkRequest("ECHO_REQUEST", request);
                    }
                    Document outDocument = converter.toDOMDocument(documentString, exchange);
                    outElements.add(new DOMSource(outDocument.getDocumentElement()));
                    // set the payload header with null
                    CxfPayload<SoapHeader> responsePayload = new CxfPayload<>(null, outElements,
null);
                    exchange.getMessage().setBody(responsePayload);
                }
            });
        }
    };
}
```

18.12. HOW TO GET AND SET SOAP HEADERS IN POJO MODE

POJO means that the data format is a "list of Java objects" when the camel-cxf endpoint produces or consumes Camel exchanges. Even though Camel exposes the message body as POJOs in this mode, camel-cxf still provides access to read and write SOAP headers. However, since CXF interceptors remove in-band SOAP headers from the header list after they have been processed, only out-of-band SOAP headers are available to camel-cxf in POJO mode.

The following example illustrates how to get/set SOAP headers. Suppose we have a route that forwards from one Camel-cxf endpoint to another. That is, SOAP Client → Camel → CXF service. We can attach two processors to obtain/insert SOAP headers at (1) before a request goes out to the CXF service and

(2) before the response comes back to the SOAP Client. Processor (1) and (2) in this example are InsertRequestOutHeaderProcessor and InsertResponseOutHeaderProcessor. Our route looks like this:

```
from("cxf:bean:routerRelayEndpointWithInsertion")
    .process(new InsertRequestOutHeaderProcessor())
    .to("cxf:bean:serviceRelayEndpointWithInsertion")
    .process(new InsertResponseOutHeaderProcessor());
```

The Bean **routerRelayEndpointWithInsertion** and **serviceRelayEndpointWithInsertion** are defined as follows:

```
@Bean
public CxfEndpoint routerRelayEndpointWithInsertion() {
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setServiceClass(org.apache.camel.component.cxf.soap.headers.HeaderTester.class);

cxfEndpoint.setAddress("/CxfMessageHeadersRelayTest/HeaderService/routerRelayEndpointWithInsert
ion");
    cxfEndpoint.setWsdlURL("soap_header.wsdl");
    cxfEndpoint.setEndpointNameAsQName(
        QName.valueOf("
{http://apache.org/camel/component/cxf/soap/headers}SoapPortRelayWithInsertion"));
    cxfEndpoint.setServiceNameAsQName(SERVICENAME);
    cxfEndpoint.getFeatures().add(new LoggingFeature());
    return cxfEndpoint;
}

@Bean
public CxfEndpoint serviceRelayEndpointWithInsertion() {
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setServiceClass(org.apache.camel.component.cxf.soap.headers.HeaderTester.class);
    cxfEndpoint.setAddress("http://localhost:" + port +
"/services/CxfMessageHeadersRelayTest/HeaderService/routerRelayEndpointWithInsertionBackend");

    cxfEndpoint.setWsdlURL("soap_header.wsdl");
    cxfEndpoint.setEndpointNameAsQName(
        QName.valueOf("
{http://apache.org/camel/component/cxf/soap/headers}SoapPortRelayWithInsertion"));
    cxfEndpoint.setServiceNameAsQName(SERVICENAME);
    cxfEndpoint.getFeatures().add(new LoggingFeature());
    return cxfEndpoint;
}
```

SOAP headers are propagated to and from Camel Message headers. The Camel message header name is "org.apache.cxf.headers.Header.list" which is a constant defined in CXF (org.apache.cxf.headers.Header.HEADER_LIST). The header value is a List of CXF SoapHeader objects (org.apache.cxf.binding.soap.SoapHeader). The following snippet is the InsertResponseOutHeaderProcessor (that insert a new SOAP header in the response message). The way to access SOAP headers in both InsertResponseOutHeaderProcessor and InsertRequestOutHeaderProcessor are actually the same. The only difference between the two processors is setting the direction of the inserted SOAP header.

You can find the **InsertResponseOutHeaderProcessor** example in CxfMessageHeadersRelayTest:

```
public static class InsertResponseOutHeaderProcessor implements Processor {
```

```
    public void process(Exchange exchange) throws Exception {
        List<SoapHeader> soapHeaders = CastUtils.cast((List<?
>)exchange.getIn().getHeader(Header.HEADER_LIST));

        // Insert a new header
        String xml = "<?xml version=\"1.0\" encoding=\"utf-8\"?><outofbandHeader "
            + "xmlns=\"http://cxf.apache.org/outofband/Header\" hdrAttribute=\"testHdrAttribute\" "
            + "xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\" soap:mustUnderstand=\"1\">"
            + "<name>New_testOobHeader</name><value>New_testOobHeaderValue</value>
</outofbandHeader>";
        SoapHeader newHeader = new SoapHeader(soapHeaders.get(0).getName(),
                    DOMUtils.readXml(new StringReader(xml)).getDocumentElement());
        // make sure direction is OUT since it is a response message.
        newHeader.setDirection(Direction.DIRECTION_OUT);
        //newHeader.setMustUnderstand(false);
        soapHeaders.add(newHeader);

    }

}
```

18.13. HOW TO GET AND SET SOAP HEADERS IN PAYLOAD MODE

We've already shown how to access the SOAP message as CxfPayload object in PAYLOAD mode inm the section How to deal with the message for a camel-cxf endpoint in PAYLOAD data format .

Once you obtain a CxfPayload object, you can invoke the CxfPayload.getHeaders() method that returns a List of DOM Elements (SOAP headers).

For an example see CxfPayLoadSoapHeaderTest:

```
from(getRouterEndpointURI()).process(new Processor() {
    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> payload = exchange.getIn().getBody(CxfPayload.class);
        List<Source> elements = payload.getBodySources();
        assertNotNull(elements, "We should get the elements here");
        assertEquals(1, elements.size(), "Get the wrong elements size");

        Element el = new XmlConverter().toDOMElement(elements.get(0));
        elements.set(0, new DOMSource(el));
        assertEquals("http://camel.apache.org/pizza/types",
                el.getNamespaceURI(), "Get the wrong namespace URI");

        List<SoapHeader> headers = payload.getHeaders();
        assertNotNull(headers, "We should get the headers here");
        assertEquals(1, headers.size(), "Get the wrong headers size");
        assertEquals("http://camel.apache.org/pizza/types",
                ((Element) (headers.get(0).getObject())).getNamespaceURI(), "Get the wrong namespace
URI");
        // alternatively you can also get the SOAP header via the camel header:
        headers = exchange.getIn().getHeader(Header.HEADER_LIST, List.class);
        assertNotNull(headers, "We should get the headers here");
        assertEquals(1, headers.size(), "Get the wrong headers size");
```

```
        assertEquals("http://camel.apache.org/pizza/types",
                ((Element) (headers.get(0).getObject())).getNamespaceURI(), "Get the wrong namespace
URI");

    }

})
.to(getServiceEndpointURI());
```

You can also use the same way as described in sub-chapter "How to get and set SOAP headers in POJO mode" to set or get the SOAP headers. So, you can use the header "org.apache.cxf.headers.Header.list" to get and set a list of SOAP headers.This does also mean that if you have a route that forwards from one Camel-cxf endpoint to another (SOAP Client → Camel → CXF service), now also the SOAP headers sent by the SOAP client are forwarded to the CXF service. If you do not want that these headers are forwarded you have to remove them in the Camel header "org.apache.cxf.headers.Header.list".

18.14. SOAP HEADERS ARE NOT AVAILABLE IN RAW MODE

SOAP headers are not available in RAW mode as SOAP processing is skipped.

18.15. HOW TO THROW A SOAP FAULT FROM CAMEL

If you are using a **camel-cxf** endpoint to consume the SOAP request, you may need to throw the SOAP Fault from the camel context.
Basically, you can use the **throwFault** DSL to do that; it works for **POJO**, **PAYLOAD** and **MESSAGE** data format.
You can define the soap fault as shown in CxfCustomizedExceptionTest:

```
SOAP_FAULT = new SoapFault(EXCEPTION_MESSAGE, SoapFault.FAULT_CODE_CLIENT);
Element detail = SOAP_FAULT.getOrCreateDetail();
Document doc = detail.getOwnerDocument();
Text tn = doc.createTextNode(DETAIL_TEXT);
detail.appendChild(tn);
```

Then throw it as you like

```
from(routerEndpointURI).setFaultBody(constant(SOAP_FAULT));
```

If your CXF endpoint is working in the **MESSAGE** data format, you could set the SOAP Fault message in the message body and set the response code in the message header as demonstrated by CxfMessageStreamExceptionTest

```
from(routerEndpointURI).process(new Processor() {

    public void process(Exchange exchange) throws Exception {
        Message out = exchange.getOut();
        // Set the message body with the
        out.setBody(this.getClass().getResourceAsStream("SoapFaultMessage.xml"));
        // Set the response code here
        out.setHeader(org.apache.cxf.message.Message.RESPONSE_CODE, new Integer(500));
    }

});
```

Same for using POJO data format. You can set the SOAPFault on the out body.

18.16. HOW TO PROPAGATE A CAMEL-CXF ENDPOINT'S REQUEST AND RESPONSE CONTEXT

CXF client API provides a way to invoke the operation with request and response context. If you are using a **camel-cxf** endpoint producer to invoke the outside web service, you can set the request context and get response context with the following code:

```
        CxfExchange exchange = (CxfExchange)template.send(getJaxwsEndpointUri(), new
Processor() {
            public void process(final Exchange exchange) {
                final List<String> params = new ArrayList<String>();
                params.add(TEST_MESSAGE);
                // Set the request context to the inMessage
                Map<String, Object> requestContext = new HashMap<String, Object>();
                requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
JAXWS_SERVER_ADDRESS);
                exchange.getIn().setBody(params);
                exchange.getIn().setHeader(Client.REQUEST_CONTEXT , requestContext);
                exchange.getIn().setHeader(CxfConstants.OPERATION_NAME,
GREET_ME_OPERATION);
            }
        });
        org.apache.camel.Message out = exchange.getOut();
        // The output is an object array, the first element of the array is the return value
        Object\[\] output = out.getBody(Object\[\].class);
        LOG.info("Received output text: " + output\[0\]);
        // Get the response context form outMessage
        Map<String, Object> responseContext =
CastUtils.cast((Map)out.getHeader(Client.RESPONSE_CONTEXT));
        assertNotNull(responseContext);
        assertEquals("Get the wrong wsdl operation name", "
{http://apache.org/hello_world_soap_http}greetMe",
                     responseContext.get("javax.xml.ws.wsdl.operation").toString());
```

18.17. ATTACHMENT SUPPORT

POJO Mode: Both SOAP with Attachment and MTOM are supported (see example in Payload Mode for enabling MTOM). However, SOAP with Attachment is not tested. Since attachments are marshalled and unmarshalled into POJOs, users typically do not need to deal with the attachment themself. Attachments are propagated to Camel message's attachments if the MTOM is not enabled. So, it is possible to retrieve attachments by Camel Message API

```
DataHandler Message.getAttachment(String id)
```

Payload Mode: MTOM is supported by the component. Attachments can be retrieved by Camel Message APIs mentioned above. SOAP with Attachment (SwA) is supported and attachments can be retrieved. SwA is the default (same as setting the CXF endpoint property "mtom-enabled" to false).

To enable MTOM, set the CXF endpoint property "mtom-enabled" to *true*.

```
@Bean
```

```
public CxfEndpoint routerEndpoint() {
    CxfSpringEndpoint cxfEndpoint = new CxfSpringEndpoint();
    cxfEndpoint.setServiceNameAsQName(SERVICE_QNAME);
    cxfEndpoint.setEndpointNameAsQName(PORT_QNAME);
    cxfEndpoint.setAddress("/" + getClass().getSimpleName()+ "/jaxws-mtom/hello");
    cxfEndpoint.setWsdlURL("mtom.wsdl");
    Map<String, Object> properties = new HashMap<String, Object>();
    properties.put("dataFormat", "PAYLOAD");
    properties.put("mtom-enabled", true);
    cxfEndpoint.setProperties(properties);
    return cxfEndpoint;
}
```

You can produce a Camel message with attachment to send to a CXF endpoint in Payload mode.

```
Exchange exchange = context.createProducerTemplate().send("direct:testEndpoint", new
Processor() {

    public void process(Exchange exchange) throws Exception {
        exchange.setPattern(ExchangePattern.InOut);
        List<Source> elements = new ArrayList<Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.REQ_MESSAGE)).getDocumentElement()));
        CxfPayload<SoapHeader> body = new CxfPayload<SoapHeader>(new ArrayList<SoapHeader>
(),
            elements, null);
        exchange.getIn().setBody(body);
        exchange.getIn().addAttachment(MtomTestHelper.REQ_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.REQ_PHOTO_DATA,
"application/octet-stream")));

        exchange.getIn().addAttachment(MtomTestHelper.REQ_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.requestJpeg, "image/jpeg")));

    }

});

// process response

CxfPayload<SoapHeader> out = exchange.getOut().getBody(CxfPayload.class);
Assert.assertEquals(1, out.getBody().size());

Map<String, String> ns = new HashMap<String, String>();
ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
ns.put("xop", MtomTestHelper.XOP_NS);

XPathUtils xu = new XPathUtils(ns);
Element oute = new XmlConverter().toDOMElement(out.getBody().get(0));
Element ele = (Element)xu.getValue("//ns:DetailResponse/ns:photo/xop:Include", oute,
                           XPathConstants.NODE);
String photoId = ele.getAttribute("href").substring(4); // skip "cid:"

ele = (Element)xu.getValue("//ns:DetailResponse/ns:image/xop:Include", oute,
                           XPathConstants.NODE);
String imageId = ele.getAttribute("href").substring(4); // skip "cid:"
```

```
DataHandler dr = exchange.getOut().getAttachment(photoId);
Assert.assertEquals("application/octet-stream", dr.getContentType());
MtomTestHelper.assertEquals(MtomTestHelper.RESP_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

dr = exchange.getOut().getAttachment(imageId);
Assert.assertEquals("image/jpeg", dr.getContentType());

BufferedImage image = ImageIO.read(dr.getInputStream());
Assert.assertEquals(560, image.getWidth());
Assert.assertEquals(300, image.getHeight());
```

You can also consume a Camel message received from a CXF endpoint in Payload mode. The CxfMtomConsumerPayloadModeTest illustrates how this works:

```
public static class MyProcessor implements Processor {

    @SuppressWarnings("unchecked")
    public void process(Exchange exchange) throws Exception {
        CxfPayload<SoapHeader> in = exchange.getIn().getBody(CxfPayload.class);

        // verify request
        Assert.assertEquals(1, in.getBody().size());

        Map<String, String> ns = new HashMap<String, String>();
        ns.put("ns", MtomTestHelper.SERVICE_TYPES_NS);
        ns.put("xop", MtomTestHelper.XOP_NS);

        XPathUtils xu = new XPathUtils(ns);
        Element body = new XmlConverter().toDOMElement(in.getBody().get(0));
        Element ele = (Element)xu.getValue("//ns:Detail/ns:photo/xop:Include", body,
                                           XPathConstants.NODE);
        String photoId = ele.getAttribute("href").substring(4); // skip "cid:"
        Assert.assertEquals(MtomTestHelper.REQ_PHOTO_CID, photoId);

        ele = (Element)xu.getValue("//ns:Detail/ns:image/xop:Include", body,
                                           XPathConstants.NODE);
        String imageId = ele.getAttribute("href").substring(4); // skip "cid:"
        Assert.assertEquals(MtomTestHelper.REQ_IMAGE_CID, imageId);

        DataHandler dr = exchange.getIn().getAttachment(photoId);
        Assert.assertEquals("application/octet-stream", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.REQ_PHOTO_DATA,
IOUtils.readBytesFromStream(dr.getInputStream()));

        dr = exchange.getIn().getAttachment(imageId);
        Assert.assertEquals("image/jpeg", dr.getContentType());
        MtomTestHelper.assertEquals(MtomTestHelper.requestJpeg,
IOUtils.readBytesFromStream(dr.getInputStream()));

        // create response
        List<Source> elements = new ArrayList<Source>();
        elements.add(new DOMSource(DOMUtils.readXml(new
StringReader(MtomTestHelper.RESP_MESSAGE)).getDocumentElement()));
        CxfPayload<SoapHeader> sbody = new CxfPayload<SoapHeader>(new
```

```
ArrayList<SoapHeader>(),
            elements, null);
        exchange.getOut().setBody(sbody);
        exchange.getOut().addAttachment(MtomTestHelper.RESP_PHOTO_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.RESP_PHOTO_DATA,
"application/octet-stream")));

        exchange.getOut().addAttachment(MtomTestHelper.RESP_IMAGE_CID,
            new DataHandler(new ByteArrayDataSource(MtomTestHelper.responseJpeg, "image/jpeg")));

    }
}
```

Raw Mode: Attachments are not supported as it does not process the message at all.

CXF_RAW Mode: MTOM is supported, and Attachments can be retrieved by Camel Message APIs mentioned above. Note that when receiving a multipart (i.e. MTOM) message the default SOAPMessage to String converter will provide the complete multipart payload on the body. If you require just the SOAP XML as a String, you can set the message body with message.getSOAPPart(), and Camel convert can do the rest of work for you.

18.18. STREAMING SUPPORT IN PAYLOAD MODE

The camel-cxf component now supports streaming of incoming messages when using PAYLOAD mode. Previously, the incoming messages would have been completely DOM parsed. For large messages, this is time consuming and uses a significant amount of memory. The incoming messages can remain as a javax.xml.transform.Source while being routed and, if nothing modifies the payload, can then be directly streamed out to the target destination. For common "simple proxy" use cases (example: from("cxf:... ").to("cxf:...")), this can provide very significant performance increases as well as significantly lowered memory requirements.

However, there are cases where streaming may not be appropriate or desired. Due to the streaming nature, invalid incoming XML may not be caught until later in the processing chain. Also, certain actions may require the message to be DOM parsed anyway (like WS-Security or message tracing and such) in which case the advantages of the streaming is limited. At this point, there are two ways to control the streaming:

- Endpoint property: you can add "allowStreaming=false" as an endpoint property to turn the streaming on/off.
- Component property: the CxfComponent object also has an allowStreaming property that can set the default for endpoints created from that component.

Global system property: you can add a system property of "org.apache.camel.component.cxf.streaming" to "false" to turn it off. That sets the global default, but setting the endpoint property above will override this value for that endpoint.

18.19. USING THE GENERIC CXF DISPATCH MODE

The camel-cxf component supports the generic CXF dispatch mode that can transport messages of arbitrary structures (i.e., not bound to a specific XML schema). To use this mode, you simply omit specifying the wsdlURL and serviceClass attributes of the CXF endpoint.

```
<cxf:cxfEndpoint id="testEndpoint" address="http://localhost:9000/SoapContext/SoapAnyPort">
    <cxf:properties>
```

```
      <entry key="dataFormat" value="PAYLOAD"/>
    </cxf:properties>
  </cxf:cxfEndpoint>
```

It is noted that the default CXF dispatch client does not send a specific SOAPAction header. Therefore, when the target service requires a specific SOAPAction value, it is supplied in the Camel header using the key SOAPAction (case-insensitive).

18.20. SPRING BOOT AUTO-CONFIGURATION

When using cxf with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-cxf-soap-starter</artifactId>
</dependency>
```

The component supports 13 options, which are listed below.

Name	Description	Default	Type
camel.component.cxf.allow-streaming	This option controls whether the CXF component, when running in PAYLOAD mode, will DOM parse the incoming messages into DOM Elements or keep the payload as a javax.xml.transform.Source object that would allow streaming in some cases.		Boolean
camel.component.cxf.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.cxf.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.cxf.enabled	Whether to enable auto configuration of the cxf component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.cxf.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.cxf.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.cxf.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.cxfrs.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.cxfrs.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.cxfrs.enabled	Whether to enable auto configuration of the cxfrs component. This is enabled by default.		Boolean
camel.component.cxfrs.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy

Name	Description	Default	Type
camel.component.cxfrs.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.cxfrs.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean

CHAPTER 19. DATA FORMAT

Only producer is supported

The Dataformat component allows to use the Data Format as a Camel Component.

19.1. URI FORMAT

```
dataformat:name:(marshal|unmarshal)[?options]
```

Where **name** is the name of the Data Format. And then followed by the operation which must either be **marshal** or **unmarshal**. The options is used for configuring the Data Format in use. See the Data Format documentation for which options it support.

19.2. DATAFORMAT OPTIONS

19.2.1. Configuring Options

Camel components are configured on two separate levels:

- component level
- endpoint level

19.2.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

19.2.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

19.3. COMPONENT OPTIONS

The Data Format component supports 2 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

19.4. ENDPOINT OPTIONS

The Data Format endpoint is configured using URI syntax:

```
dataformat:name:operation
```

with the following path and query parameters:

19.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
name (producer)	**Required** Name of data format.		String
operation (producer)	**Required** Operation to use either marshal or unmarshal. Enum values: • marshal • unmarshal		String

19.4.2. Query Parameters (1 parameters)

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

19.5. SAMPLES

For example to use the JAXB Data Format we can do as follows:

```
from("activemq:My.Queue").
  to("dataformat:jaxb:unmarshal?contextPath=com.acme.model").
  to("mqseries:Another.Queue");
```

And in XML DSL you do:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="dataformat:jaxb:unmarshal?contextPath=com.acme.model"/>
    <to uri="mqseries:Another.Queue"/>
  </route>
</camelContext>
```

19.6. SPRING BOOT AUTO-CONFIGURATION

When using dataformat with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-dataformat-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type
camel.component.dataformat.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.dataformat.enabled	Whether to enable auto configuration of the dataformat component. This is enabled by default.		Boolean
camel.component.dataformat.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 20. DATASET

Both producer and consumer are supported

Testing of distributed and asynchronous processing is notoriously difficult. The Mock, Test and DataSet endpoints work great with the Camel Testing Framework to simplify your unit and integration testing using Enterprise Integration Patterns and Camel's large range of Components together with the powerful Bean Integration.

The DataSet component provides a mechanism to easily perform load & soak testing of your system. It works by allowing you to create DataSet instances both as a source of messages and as a way to assert that the data set is received.

Camel will use the throughput logger when sending datasets.

20.1. URI FORMAT

```
dataset:name[?options]
```

Where **name** is used to find the DataSet instance in the Registry

Camel ships with a support implementation of **org.apache.camel.component.dataset.DataSet**, the **org.apache.camel.component.dataset.DataSetSupport** class, that can be used as a base for implementing your own DataSet. Camel also ships with some implementations that can be used for testing: **org.apache.camel.component.dataset.SimpleDataSet**, **org.apache.camel.component.dataset.ListDataSet** and **org.apache.camel.component.dataset.FileDataSet**, all of which extend **DataSetSupport**.

20.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

20.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

20.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

20.3. COMPONENT OPTIONS

The Dataset component supports 5 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
log (producer)	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
exchangeFormatter (advanced)	**Autowired** Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter.		ExchangeFormatter

20.4. ENDPOINT OPTIONS

The Dataset endpoint is configured using URI syntax:

```
dataset:name
```

with the following path and query parameters:

20.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** Name of DataSet to lookup in the registry.		DataSet

20.4.2. Query Parameters (21 parameters)

Name	Description	Default	Type
dataSetIndex (common)	Controls the behaviour of the CamelDataSetIndex header. For Consumers: - off = the header will not be set - strict/lenient = the header will be set For Producers: - off = the header value will not be verified, and will not be set if it is not present = strict = the header value must be present and will be verified = lenient = the header value will be verified if it is present, and will be set if it is not present. Enum values: • strict • lenient • off	lenient	String

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
initialDelay (consumer)	Time period in millis to wait before starting sending messages.	1000	long
minRate (consumer)	Wait until the DataSet contains at least this number of messages.	0	int
preloadSize (consumer)	Sets how many messages should be preloaded (sent) before the route completes its initialization.	0	long
produceDelay (consumer)	Allows a delay to be specified which causes a delay when a message is sent by the consumer (to simulate slow processing).	3	long
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern

Name	Description	Default	Type
assertPeriod (producer)	Sets a grace period after which the mock endpoint will re-assert to ensure the preliminary assertion is still valid. This is used for example to assert that exactly a number of messages arrives. For example if expectedMessageCount(int) was set to 5, then the assertion is satisfied when 5 or more message arrives. To ensure that exactly 5 messages arrives, then you would need to wait a little period to ensure no further message arrives. This is what you can use this method for. By default this period is disabled.		long
consumeDelay (producer)	Allows a delay to be specified which causes a delay when a message is consumed by the producer (to simulate slow processing).	0	long
expectedCount (producer)	Specifies the expected number of message exchanges that should be received by this endpoint. Beware: If you want to expect that 0 messages, then take extra care, as 0 matches when the tests starts, so you need to set a assert period time to let the test run for a while to make sure there are still no messages arrived; for that use setAssertPeriod(long). An alternative is to use NotifyBuilder, and use the notifier to know when Camel is done routing some messages, before you call the assertIsSatisfied() method on the mocks. This allows you to not use a fixed assert period, to speedup testing times. If you want to assert that exactly n'th message arrives to this mock endpoint, then see also the setAssertPeriod(long) method for further details.	-1	int
failFast (producer)	Sets whether assertIsSatisfied() should fail fast at the first detected failed expectation while it may otherwise wait for all expected messages to arrive before performing expectations verifications. Is by default true. Set to false to use behavior as in Camel 2.x.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
log (producer)	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	boolean
reportGroup (producer)	A number that is used to turn on throughput logging based on groups of the size.		int
resultMinimumWaitTime (producer)	Sets the minimum expected amount of time (in millis) the assertIsSatisfied() will wait on a latch until it is satisfied.		long
resultWaitTime (producer)	Sets the maximum amount of time (in millis) the assertIsSatisfied() will wait on a latch until it is satisfied.		long
retainFirst (producer)	Specifies to only retain the first n'th number of received Exchanges. This is used when testing with big data, to reduce memory consumption by not storing copies of every Exchange this mock endpoint receives. Important: When using this limitation, then the getReceivedCounter() will still return the actual number of received Exchanges. For example if we have received 5000 Exchanges, and have configured to only retain the first 10 Exchanges, then the getReceivedCounter() will still return 5000 but there is only the first 10 Exchanges in the getExchanges() and getReceivedExchanges() methods. When using this method, then some of the other expectation methods is not supported, for example the expectedBodiesReceived(Object...) sets a expectation on the first number of bodies received. You can configure both setRetainFirst(int) and setRetainLast(int) methods, to limit both the first and last received.	-1	int

Name	Description	Default	Type
retainLast (producer)	Specifies to only retain the last n'th number of received Exchanges. This is used when testing with big data, to reduce memory consumption by not storing copies of every Exchange this mock endpoint receives. Important: When using this limitation, then the getReceivedCounter() will still return the actual number of received Exchanges. For example if we have received 5000 Exchanges, and have configured to only retain the last 20 Exchanges, then the getReceivedCounter() will still return 5000 but there is only the last 20 Exchanges in the getExchanges() and getReceivedExchanges() methods. When using this method, then some of the other expectation methods is not supported, for example the expectedBodiesReceived(Object...) sets a expectation on the first number of bodies received. You can configure both setRetainFirst(int) and setRetainLast(int) methods, to limit both the first and last received.	-1	int
sleepForEmptyTest (producer)	Allows a sleep to be specified to wait to check that this endpoint really is empty when expectedMessageCount(int) is called with zero.		long
copyOnExchange (producer (advanced))	Sets whether to make a deep copy of the incoming Exchange when received at this mock endpoint. Is by default true.	true	boolean

20.5. CONFIGURING DATASET

Camel will lookup in the Registry for a bean implementing the DataSet interface. So you can register your own DataSet as:

```
<bean id="myDataSet" class="com.mycompany.MyDataSet">
  <property name="size" value="100"/>
</bean>
```

20.6. EXAMPLE

For example, to test that a set of messages are sent to a queue and then consumed from the queue without losing any messages:

```
// send the dataset to a queue
from("dataset:foo").to("activemq:SomeQueue");

// now lets test that the messages are consumed correctly
from("activemq:SomeQueue").to("dataset:foo");
```

The above would look in the Registry to find the **foo** DataSet instance which is used to create the messages.

Then you create a DataSet implementation, such as using the **SimpleDataSet** as described below, configuring things like how big the data set is and what the messages look like etc.

20.7. DATASETSUPPORT (ABSTRACT CLASS)

The DataSetSupport abstract class is a nice starting point for new DataSets, and provides some useful features to derived classes.

20.7.1. Properties on DataSetSupport

Property	Type	Default	Description
defaultHeaders	**Map<String,Object>**	**null**	Specifies the default message body. For SimpleDataSet it is a constant payload; though if you want to create custom payloads per message, create your own derivation of **DataSetSupport**.
outputTransformer	**org.apache.camel.Processor**	null	
size	**long**	**10**	Specifies how many messages to send/consume.
reportCount	**long**	**-1**	Specifies the number of messages to be received before reporting progress. Useful for showing progress of a large load test. If < 0, then **size** / 5, if is 0 then **size**, else set to **reportCount** value.

20.8. SIMPLEDATASET

The **SimpleDataSet** extends **DataSetSupport**, and adds a default body.

20.8.1. Additional Properties on SimpleDataSet

Property	Type	Default	Description
defaultBody	**Object**	**<hello>world!</hello>**	Specifies the default message body. By default, the **SimpleDataSet** produces the same constant payload for each exchange. If you want to customize the payload for each exchange, create a Camel **Processor** and configure the **SimpleDataSet** to use it by setting the **outputTransformer** property.

20.9. LISTDATASET

The List`DataSet` extends **DataSetSupport**, and adds a list of default bodies.

20.9.1. Additional Properties on ListDataSet

Property	Type	Default	Description
defaultBodies	**List<Object>**	**empty LinkedList<Object>**	Specifies the default message body. By default, the **ListDataSet** selects a constant payload from the list of **defaultBodies** using the **CamelDataSetIndex**. If you want to customize the payload, create a Camel **Processor** and configure the **ListDataSet** to use it by setting the **outputTransformer** property.
size	**long**	the size of the defaultBodies list	Specifies how many messages to send/consume. This value can be different from the size of the **defaultBodies** list. If the value is less than the size of the **defaultBodies** list, some of the list elements will not be used. If the value is greater than the size of the **defaultBodies** list, the payload for the exchange will be selected using the modulus of the **CamelDataSetIndex** and the size of the **defaultBodies** list (i.e. **CamelDataSetIndex % defaultBodies.size()**)

20.10. FILEDATASET

The **FileDataSet** extends **ListDataSet**, and adds support for loading the bodies from a file.

20.10.1. Additional Properties on FileDataSet

Property	Type	Default	Description
sourceFile	**File**	null	Specifies the source file for payloads
delimiter	**String**	\z	Specifies the delimiter pattern used by a **java.util.Scanner** to split the file into multiple payloads.

20.11. SPRING BOOT AUTO-CONFIGURATION

When using dataset with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-dataset-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

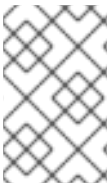
Name	Description	Default	Type
camel.component.dataset-test.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.dataset-test.enabled	Whether to enable auto configuration of the dataset-test component. This is enabled by default.		Boolean
camel.component.dataset-test.exchange-formatter	Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter. The option is a org.apache.camel.spi.ExchangeFormatter type.		ExchangeFormatter
camel.component.dataset-test.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.dataset-test.log	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	Boolean

Name	Description	Default	Type
camel.component.dataset.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.dataset.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.dataset.enabled	Whether to enable auto configuration of the dataset component. This is enabled by default.		Boolean
camel.component.dataset.exchange-formatter	Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter. The option is a org.apache.camel.spi.ExchangeFormatter type.		ExchangeFormatter
camel.component.dataset.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.dataset.log	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	Boolean

CHAPTER 21. DIRECT

Both producer and consumer are supported

The Direct component provides direct, synchronous invocation of any consumers when a producer sends a message exchange.
This endpoint can be used to connect existing routes in the **same** camel context.



NOTE

Asynchronous
The SEDA component provides asynchronous invocation of any consumers when a producer sends a message exchange.

21.1. URI FORMAT

```
direct:someName[?options]
```

Where **someName** can be any string to uniquely identify the endpoint

21.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

21.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

21.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

21.3. COMPONENT OPTIONS

The Direct component supports 5 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
block (producer)	If sending a message to a direct endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
timeout (producer)	The timeout value to use if block is enabled.	30000	long
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

21.4. ENDPOINT OPTIONS

The Direct endpoint is configured using URI syntax:

```
direct:name
```

with the following path and query parameters:

21.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** Name of direct endpoint.		String

21.4.2. Query Parameters (8 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern
block (producer)	If sending a message to a direct endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	boolean
failIfNoConsumers (producer)	Whether the producer should fail by throwing an exception, when sending to a DIRECT endpoint with no active consumers.	true	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
timeout (producer)	The timeout value to use if block is enabled.	30000	long
synchronous (advanced)	Whether synchronous processing is forced. If enabled then the producer thread, will be forced to wait until the message has been completed before the same thread will continue processing. If disabled (default) then the producer thread may be freed and can do other work while the message is continued processed by other threads (reactive).	false	boolean

21.5. SAMPLES

In the route below we use the direct component to link the two routes together:

```
from("activemq:queue:order.in")
    .to("bean:orderServer?method=validate")
    .to("direct:processOrder");

from("direct:processOrder")
    .to("bean:orderService?method=process")
    .to("activemq:queue:order.out");
```

And the sample using spring DSL:

```
<route>
 <from uri="activemq:queue:order.in"/>
 <to uri="bean:orderService?method=validate"/>
 <to uri="direct:processOrder"/>
</route>

<route>
 <from uri="direct:processOrder"/>
```

```
 <to uri="bean:orderService?method=process"/>
 <to uri="activemq:queue:order.out"/>
</route>
```

See also samples from the SEDA component, how they can be used together.

21.6. SPRING BOOT AUTO-CONFIGURATION

When using direct with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-direct-starter</artifactId>
</dependency>
```

The component supports 6 options, which are listed below.

Name	Description	Default	Type
camel.component.direct.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.direct.block	If sending a message to a direct endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	Boolean
camel.component.direct.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.direct.enabled	Whether to enable auto configuration of the direct component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.direct.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.direct.timeout	The timeout value to use if block is enabled.	30000	Long

CHAPTER 22. ELASTICSEARCH

Since Camel 3.18.3

Only producer is supported

The ElasticSearch component allows you to interface with an ElasticSearch 8.x API using the Java API Client library.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-elasticsearch</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

22.1. URI FORMAT

```
elasticsearch://clusterName[?options]
```

22.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

22.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

22.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

22.3. COMPONENT OPTIONS

The Elasticsearch component supports 14 options, which are listed below.

Name	Description	Default	Type
connectionTimeout (producer)	The time in ms to wait before connection will timeout.	30000	int
hostAddresses (producer)	Comma separated list with ip:port formatted remote transport addresses to use. The ip and port options must be left blank for hostAddresses to be considered instead.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
maxRetryTimeout (producer)	The time in ms before retry.	30000	int
socketTimeout (producer)	The timeout in ms to wait before the socket will timeout.	30000	int
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
client (advanced)	**Autowired** To use an existing configured Elasticsearch client, instead of creating a client per endpoint. This allow to customize the client with specific settings.		RestClient

Name	Description	Default	Type
enableSniffer (advanced)	Enable automatically discover nodes from a running Elasticsearch cluster. If this option is used in conjunction with Spring Boot then it's managed by the Spring Boot configuration (see: Disable Sniffer in Spring Boot).	false	boolean
sniffAfterFailureDelay (advanced)	The delay of a sniff execution scheduled after a failure (in milliseconds).	60000	int
snifferInterval (advanced)	The interval between consecutive ordinary sniff executions in milliseconds. Will be honoured when sniffOnFailure is disabled or when there are no failures between consecutive sniff executions.	300000	int
certificatePath (security)	The path of the self-signed certificate to use to access to Elasticsearch.		String
enableSSL (security)	Enable SSL.	false	boolean
password (security)	Password for authenticate.		String
user (security)	Basic authenticate user.		String

22.4. ENDPOINT OPTIONS

The Elasticsearch endpoint is configured using URI syntax:

```
elasticsearch:clusterName
```

with the following path and query parameters:

22.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
clusterName (producer)	**Required** Name of the cluster.		String

22.4.2. Query Parameters (19 parameters)

Name	Description	Default	Type
connectionTimeout (producer)	The time in ms to wait before connection will timeout.	30000	int
disconnect (producer)	Disconnect after it finish calling the producer.	false	boolean
from (producer)	Starting index of the response.		Integer
hostAddresses (producer)	Comma separated list with ip:port formatted remote transport addresses to use.		String
indexName (producer)	The name of the index to act against.		String
maxRetryTimeout (producer)	The time in ms before retry.	30000	int
operation (producer)	What operation to perform. Enum values: • Index • Update • Bulk • GetById • MultiGet • MultiSearch • Delete • DeleteIndex • Search • Exists • Ping		ElasticsearchOperation
scrollKeepAliveMs (producer)	Time in ms during which elasticsearch will keep search context alive.	60000	int
size (producer)	Size of the response.		Integer
socketTimeout (producer)	The timeout in ms to wait before the socket will timeout.	30000	int

Name	Description	Default	Type
useScroll (producer)	Enable scroll usage.	false	boolean
waitForActiveShards (producer)	Index creation waits for the write consistency number of shards to be available.	1	int
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
documentClass (advanced)	The class to use when deserializing the documents.	Object Node	Class
enableSniffer (advanced)	Enable automatically discover nodes from a running Elasticsearch cluster. If this option is used in conjunction with Spring Boot then it's managed by the Spring Boot configuration (see: Disable Sniffer in Spring Boot).	false	boolean
sniffAfterFailureDelay (advanced)	The delay of a sniff execution scheduled after a failure (in milliseconds).	60000	int
snifferInterval (advanced)	The interval between consecutive ordinary sniff executions in milliseconds. Will be honoured when sniffOnFailure is disabled or when there are no failures between consecutive sniff executions.	300000	int
certificatePath (security)	The path of the self-signed certificate to use to access to Elasticsearch.		String
enableSSL (security)	Enable SSL.	false	boolean

22.5. MESSAGE HEADERS

The Elasticsearch component supports 9 message header(s), which is/are listed below:

Name	Description	Default	Type
operation (producer) Constant: PARAM_OPERATION	The operation to perform. Enum values: • Index • Update • Bulk • GetById • MultiGet • MultiSearch • Delete • DeleteIndex • Search • Exists • Ping		ElasticsearchOperation
indexId (producer) Constant: PARAM_INDEX_ID	The id of the indexed document.		String
indexName (producer) Constant: PARAM_INDEX_NAME	The name of the index to act against.		String
documentClass (producer) Constant: PARAM_DOCUMENT_CLASS	The full qualified name of the class of the document to unmarshall.	ObjectNode	Class
waitForActiveShards (producer) Constant: PARAM_WAIT_FOR_ACTIVE_SHARDS	The index creation waits for the write consistency number of shards to be available.		Integer

Name	Description	Default	Type
scrollKeepAliveMs (producer) Constant: PARAM_SCROLL_KEEP_ALIVE_MS	The starting index of the response.		Integer
useScroll (producer) Constant: PARAM_SCROLL	Set to true to enable scroll usage.		Boolean
size (producer) Constant: PARAM_SIZE	The size of the response.		Integer
from (producer) Constant: PARAM_FROM	The starting index of the response.		Integer

22.6. MESSAGE OPERATIONS

The following ElasticSearch operations are currently supported. Simply set an endpoint URI option or exchange header with a key of "operation" and a value set to one of the following. Some operations also require other parameters or the message body to be set.

operation	message body	description

operation	message body	description
Index	**Map**, **String**, **byte[]**, **Reader**, **InputStream** or **IndexRequest.Builder** content to index	Adds content to an index and returns the content's indexId in the body. You can set the name of the target index by setting the message header with the key "indexName". You can set the indexId by setting the message header with the key "indexId".
GetById	**String** or **GetRequest.Builder** index id of content to retrieve	Retrieves the document corresponding to the given index id and returns a GetResponse object in the body. You can set the name of the target index by setting the message header with the key "indexName". You can set the type of document by setting the message header with the key "documentClass".
Delete	**String** or **DeleteRequest.Builder** index id of content to delete	Deletes the specified indexName and returns a Result object in the body. You can set the name of the target index by setting the message header with the key "indexName".

operation	message body	description
DeleteIndex	**String** or **DeleteIndexRequest.Builder** index name of the index to delete	Deletes the specified indexName and returns a status code in the body. You can set the name of the target index by setting the message header with the key "indexName".

operation	message body	description
Bulk	**Iterable** or **BulkRequest.Builder** of any type that is already accepted (DeleteOperation.Builder for delete operation, UpdateOperation.Builder for update operation, CreateOperation.Builder for create operation, byte[], InputStream, String, Reader, Map or any document type for index operation)	Adds/Updates/Deletes content from/to an index and returns a List<BulkResponseItem> object in the body You can set the name of the target index by setting the message header with the key "indexName".

operation	message body	description
Search	**Map**, **String** or **SearchRequest.Builder**	Search the content with the map of query string. You can set the name of the target index by setting the message header with the key "indexName". You can set the number of hits to return by setting the message header with the key "size". You can set the starting document offset by setting the message header with the key "from".
MultiSearch	**MsearchRequest.Builder**	Multiple search in one
MultiGet	**Iterable<String>** or **MgetRequest.Builder** the id of the document to retrieve	Multiple get in one You can set the name of the target index by setting the message header with the key "indexName".
Exists	None	Checks whether the index exists or not and returns a Boolean flag in the body. You must set the name of the target index by setting the message header with the key "indexName".
Update	**byte[]**, **InputStream**, **String**, **Reader**, **Map** or any document type content to update	Updates content to an index and returns the content's indexId in the body. You can set the name of the target index by setting the message header with the key "indexName". You can set the indexId by setting the message header with the key "indexId".
Ping	None	Pings the Elasticsearch cluster and returns true if the ping succeeded, false otherwise

22.7. CONFIGURE THE COMPONENT AND ENABLE BASIC AUTHENTICATION

To use the Elasticsearch component it has to be configured with a minimum configuration.

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
camelContext.addComponent("elasticsearch", elasticsearchComponent);
```

For basic authentication with elasticsearch or using reverse http proxy in front of the elasticsearch cluster, simply setup basic authentication and SSL on the component like the example below

```
ElasticsearchComponent elasticsearchComponent = new ElasticsearchComponent();
elasticsearchComponent.setHostAddresses("myelkhost:9200");
elasticsearchComponent.setUser("elkuser");
elasticsearchComponent.setPassword("secure!!");
elasticsearchComponent.setEnableSSL(true);
elasticsearchComponent.setCertificatePath(certPath);

camelContext.addComponent("elasticsearch", elasticsearchComponent);
```

22.8. INDEX EXAMPLE

Below is a simple INDEX example

```
from("direct:index")
  .to("elasticsearch://elasticsearch?operation=Index&indexName=twitter");
```

```
<route>
    <from uri="direct:index"/>
    <to uri="elasticsearch://elasticsearch?operation=Index&amp;indexName=twitter"/>
</route>
```

For this operation you need to specify a indexId header.

A client would simply need to pass a body message containing a Map to the route. The result body contains the indexId created.

```
Map<String, String> map = new HashMap<String, String>();
map.put("content", "test");
String indexId = template.requestBody("direct:index", map, String.class);
```

22.9. SEARCH EXAMPLE

Searching on specific field(s) and value use the Operation ´Search´. Pass in the query JSON String or the Map

```
from("direct:search")
  .to("elasticsearch://elasticsearch?operation=Search&indexName=twitter");
```

```
<route>
```

```
    <from uri="direct:search"/>
    <to uri="elasticsearch://elasticsearch?operation=Search&amp;indexName=twitter"/>
</route>
```

```
String query = "{\"query\":{\"match\":{\"doc.content\":\"new release of ApacheCamel\"}}}";
HitsMetadata<?> response = template.requestBody("direct:search", query, HitsMetadata.class);
```

Search on specific field(s) using Map.

```
Map<String, Object> actualQuery = new HashMap<>();
actualQuery.put("doc.content", "new release of ApacheCamel");

Map<String, Object> match = new HashMap<>();
match.put("match", actualQuery);

Map<String, Object> query = new HashMap<>();
query.put("query", match);
HitsMetadata<?> response = template.requestBody("direct:search", query, HitsMetadata.class);
```

Search using Elasticsearch scroll api in order to fetch all results.

```
from("direct:search")
  .to("elasticsearch://elasticsearch?
operation=Search&indexName=twitter&useScroll=true&scrollKeepAliveMs=30000");
```

```
<route>
    <from uri="direct:search"/>
    <to uri="elasticsearch://elasticsearch?
operation=Search&amp;indexName=twitter&amp;useScroll=true&amp;scrollKeepAliveMs=30000"/>
</route>
```

```
String query = "{\"query\":{\"match\":{\"doc.content\":\"new release of ApacheCamel\"}}}";
try (ElasticsearchScrollRequestIterator response = template.requestBody("direct:search", query,
ElasticsearchScrollRequestIterator.class)) {
    // do something smart with results
}
```

can also be used.

```
from("direct:search")
  .to("elasticsearch://elasticsearch?
operation=Search&indexName=twitter&useScroll=true&scrollKeepAliveMs=30000")
  .split()
  .body()
  .streaming()
  .to("mock:output")
  .end();
```

22.10. MULTISEARCH EXAMPLE

MultiSearching on specific field(s) and value use the Operation ´MultiSearch´. Pass in the MultiSearchRequest instance

-

```
from("direct:multiSearch")
  .to("elasticsearch://elasticsearch?operation=MultiSearch");
```

```
<route>
    <from uri="direct:multiSearch"/>
    <to uri="elasticsearch://elasticsearch?operation=MultiSearch"/>
</route>
```

MultiSearch on specific field(s)

```
MsearchRequest.Builder builder = new MsearchRequest.Builder().index("twitter").searches(
        new RequestItem.Builder().header(new MultisearchHeader.Builder().build())
                .body(new MultisearchBody.Builder().query(b -> b.matchAll(x -> x)).build()).build(),
        new RequestItem.Builder().header(new MultisearchHeader.Builder().build())
                .body(new MultisearchBody.Builder().query(b -> b.matchAll(x -> x)).build()).build());
List<MultiSearchResponseItem<?>> response = template.requestBody("direct:multiSearch", builder,
List.class);
```

22.11. DOCUMENT TYPE

For all the search operations, it is possible to indicate the type of document to retrieve in order to get the result already unmarshalled with the expected type.

The document type can be set using the header "documentClass" or via the uri parameter of the same name.

22.12. USING CAMEL ELASTICSEARCH WITH SPRING BOOT

When you use **camel-elasticsearch-starter** with Spring Boot v2, then you must declare the following dependency in your own **pom.xml**.

```
<dependency>
  <groupId>jakarta.json</groupId>
  <artifactId>jakarta.json-api</artifactId>
  <version>2.0.2</version>
</dependency>
```

This is needed because Spring Boot v2 provides jakarta.json-api:1.1.6, and Elasticsearch requires to use json-api v2.

22.12.1. Use RestClient provided by Spring Boot

By default Spring Boot will auto configure an Elasticsearch RestClient that will be used by camel, it is possible to customize the client with the following basic properties:

```
spring.elasticsearch.uris=myelkhost:9200
spring.elasticsearch.username=elkuser
spring.elasticsearch.password=secure!!
```

More information can be found in application-properties.data.spring.elasticsearch.connection-timeout.

22.12.2. Disable Sniffer when using Spring Boot

When Spring Boot is on the classpath the Sniffer client for Elasticsearch is enabled by default. This option can be disabled in the Spring Boot Configuration:

```
spring:
  autoconfigure:
    exclude:
org.springframework.boot.autoconfigure.elasticsearch.ElasticsearchRestClientAutoConfiguration
```

22.13. SPRING BOOT AUTO-CONFIGURATION

When using elasticsearch with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-elasticsearch-starter</artifactId>
</dependency>
```

The component supports 15 options, which are listed below.

Name	Description	Default	Type
camel.component.elasticsearch.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.elasticsearch.certificate-path	The path of the self-signed certificate to use to access to Elasticsearch.		String
camel.component.elasticsearch.client	To use an existing configured Elasticsearch client, instead of creating a client per endpoint. This allow to customize the client with specific settings. The option is a org.elasticsearch.client.RestClient type.		RestClient
camel.component.elasticsearch.connection-timeout	The time in ms to wait before connection will timeout.	30000	Integer
camel.component.elasticsearch.enable-s-s-l	Enable SSL.	false	Boolean

Name	Description	Default	Type
camel.component.elasticsearch.enable-sniffer	Enable automatically discover nodes from a running Elasticsearch cluster. If this option is used in conjunction with Spring Boot then it's managed by the Spring Boot configuration (see: Disable Sniffer in Spring Boot).	false	Boolean
camel.component.elasticsearch.enabled	Whether to enable auto configuration of the elasticsearch component. This is enabled by default.		Boolean
camel.component.elasticsearch.host-addresses	Comma separated list with ip:port formatted remote transport addresses to use. The ip and port options must be left blank for hostAddresses to be considered instead.		String
camel.component.elasticsearch.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.elasticsearch.max-retry-timeout	The time in ms before retry.	30000	Integer
camel.component.elasticsearch.password	Password for authenticate.		String
camel.component.elasticsearch.sniff-after-failure-delay	The delay of a sniff execution scheduled after a failure (in milliseconds).	60000	Integer
camel.component.elasticsearch.sniffer-interval	The interval between consecutive ordinary sniff executions in milliseconds. Will be honoured when sniffOnFailure is disabled or when there are no failures between consecutive sniff executions.	300000	Integer

Name	Description	Default	Type
camel.component.elasticsearch.socket-timeout	The timeout in ms to wait before the socket will timeout.	30000	Integer
camel.component.elasticsearch.user	Basic authenticate user.		String

CHAPTER 23. FHIR

Both producer and consumer are supported

The FHIR component integrates with the HAPI-FHIR library which is an open-source implementation of the FHIR (Fast Healthcare Interoperability Resources) specification in Java.

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

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-fhir</artifactId>
    <version>${camel-version}</version>
</dependency>
```

23.1. URI FORMAT

The FHIR Component uses the following URI format:

```
fhir://endpoint-prefix/endpoint?[options]
```

Endpoint prefix can be one of:

- capabilities
- create
- delete
- history
- load-page
- meta
- operation
- patch
- read
- search
- transaction
- update
- validate

23.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level

- endpoint level

23.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

23.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

23.3. COMPONENT OPTIONS

The FHIR component supports 27 options, which are listed below.

Name	Description	Default	Type
encoding (common)	Encoding to use for all request. Enum values: • JSON • XML		String

Name	Description	Default	Type
fhirVersion (common)	The FHIR Version to use. Enum values: • DSTU2 • DSTU2_HL7ORG • DSTU2_1 • DSTU3 • R4 • R5	R4	String
log (common)	Will log every requests and responses.	false	boolean
prettyPrint (common)	Pretty print all request.	false	boolean
serverUrl (common)	The FHIR server base URL.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
client (advanced)	To use the custom client.		IGenericClient
clientFactory (advanced)	To use the custom client factory.		IRestfulClientFactory
compress (advanced)	Compresses outgoing (POST/PUT) contents to the GZIP format.	false	boolean
configuration (advanced)	To use the shared configuration.		FhirConfiguration
connectionTimeout (advanced)	How long to try and establish the initial TCP connection (in ms).	10000	Integer
deferModelScanning (advanced)	When this option is set, model classes will not be scanned for children until the child list for the given type is actually accessed.	false	boolean
fhirContext (advanced)	FhirContext is an expensive object to create. To avoid creating multiple instances, it can be set directly.		FhirContext
forceConformanceCheck (advanced)	Force conformance check.	false	boolean
sessionCookie (advanced)	HTTP session cookie to add to every request.		String
socketTimeout (advanced)	How long to block for individual read/write operations (in ms).	10000	Integer

Name	Description	Default	Type
summary (advanced)	Request that the server modify the response using the _summary param. Enum values: • COUNT • TEXT • DATA • TRUE • FALSE		String
validationMode (advanced)	When should Camel validate the FHIR Server's conformance statement. Enum values: • NEVER • ONCE	ONCE	String
proxyHost (proxy)	The proxy host.		String
proxyPassword (proxy)	The proxy password.		String
proxyPort (proxy)	The proxy port.		Integer
proxyUser (proxy)	The proxy username.		String
accessToken (security)	OAuth access token.		String
password (security)	Username to use for basic authentication.		String
username (security)	Username to use for basic authentication.		String

23.4. ENDPOINT OPTIONS

The FHIR endpoint is configured using URI syntax:

```
fhir:apiName/methodName
```

with the following path and query parameters:

23.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
apiName (common)	**Required** What kind of operation to perform. Enum values: • CAPABILITIES • CREATE • DELETE • HISTORY • LOAD_PAGE • META • OPERATION • PATCH • READ • SEARCH • TRANSACTION • UPDATE • VALIDATE		FhirApiName
methodName (common)	**Required** What sub operation to use for the selected operation.		String

23.4.2. Query Parameters (44 parameters)

Name	Description	Default	Type
encoding (common)	Encoding to use for all request. Enum values: • JSON • XML		String

Name	Description	Default	Type
fhirVersion (common)	The FHIR Version to use. Enum values: • DSTU2 • DSTU2_HL7ORG • DSTU2_1 • DSTU3 • R4 • R5	R4	String
inBody (common)	Sets the name of a parameter to be passed in the exchange In Body.		String
log (common)	Will log every requests and responses.	false	boolean
prettyPrint (common)	Pretty print all request.	false	boolean
serverUrl (common)	The FHIR server base URL.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
client (advanced)	To use the custom client.		IGenericClient
clientFactory (advanced)	To use the custom client factory.		IRestfulClientFactory
compress (advanced)	Compresses outgoing (POST/PUT) contents to the GZIP format.	false	boolean
connectionTimeout (advanced)	How long to try and establish the initial TCP connection (in ms).	10000	Integer
deferModelScanning (advanced)	When this option is set, model classes will not be scanned for children until the child list for the given type is actually accessed.	false	boolean
fhirContext (advanced)	FhirContext is an expensive object to create. To avoid creating multiple instances, it can be set directly.		FhirContext

Name	Description	Default	Type
forceConformanceCheck (advanced)	Force conformance check.	false	boolean
sessionCookie (advanced)	HTTP session cookie to add to every request.		String
socketTimeout (advanced)	How long to block for individual read/write operations (in ms).	10000	Integer
summary (advanced)	Request that the server modify the response using the _summary param. Enum values: • COUNT • TEXT • DATA • TRUE • FALSE		String
validationMode (advanced)	When should Camel validate the FHIR Server's conformance statement. Enum values: • NEVER • ONCE	ONCE	String
proxyHost (proxy)	The proxy host.		String
proxyPassword (proxy)	The proxy password.		String
proxyPort (proxy)	The proxy port.		Integer
proxyUser (proxy)	The proxy username.		String
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object

Name	Description	Default	Type
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessToken (security)	OAuth access token.		String
password (security)	Username to use for basic authentication.		String
username (security)	Username to use for basic authentication.		String

23.5. API PARAMETERS (13 APIS)

The @FHIR endpoint is an API based component and has additional parameters based on which API name and API method is used. The API name and API method is located in the endpoint URI as the **apiName/methodName** path parameters:

```
fhir:apiName/methodName
```

There are 13 API names as listed in the table below:

API Name	Type	Description
capabilities	Both	API to Fetch the capability statement for the server
create	Both	API for the create operation, which creates a new resource instance on the server
delete	Both	API for the delete operation, which performs a logical delete on a server resource
history	Both	API for the history method
load-page	Both	API that Loads the previous/next bundle of resources from a paged set, using the link specified in the link type=next tag within the atom bundle
meta	Both	API for the meta operations, which can be used to get, add and remove tags and other Meta elements from a resource or across the server
operation	Both	API for extended FHIR operations
patch	Both	API for the patch operation, which performs a logical patch on a server resource
read	Both	API method for read operations
search	Both	API to search for resources matching a given set of criteria
transaction	Both	API for sending a transaction (collection of resources) to the server to be executed as a single unit
update	Both	API for the update operation, which performs a logical delete on a server resource
validate	Both	API for validating resources

Each API is documented in the following sections to come.

23.5.1. API: capabilities

Both producer and consumer are supported

The capabilities API is defined in the syntax as follows:

```
fhir:capabilities/methodName?[parameters]
```

The method is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
ofType	Retrieve the conformance statement using the given model type

23.5.1.1. Method ofType

Signatures:

- org.hl7.fhir.instance.model.api.IBaseConformance ofType(Class<org.hl7.fhir.instance.model.api.IBaseConformance> type, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/ofType API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
type	The model type	Class

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.2. API: create

Both producer and consumer are supported

The create API is defined in the syntax as follows:

```
fhir:create/methodName?[parameters]
```

The 1 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
resource	Creates a IBaseResource on the server

23.5.2.1. Method resource

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome resource(String resourceAsString, String url, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- ca.uhn.fhir.rest.api.MethodOutcome resource(org.hl7.fhir.instance.model.api.IBaseResource resource, String url, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resource API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
preferReturn	Add a Prefer header to the request, which requests that the server include or suppress the resource body as a part of the result. If a resource is returned by the server it will be parsed an accessible to the client via MethodOutcome#getResource() , may be null	PreferReturnEnum
resource	The resource to create	IBaseResource
resourceAsString	The resource to create	String
url	The search URL to use. The format of this URL should be of the form ResourceTypeParameters, for example: Patientname=Smith&identifier=13.2.4.11.4%7C847366, may be null	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.3. API: delete

Both producer and consumer are supported

The delete API is defined in the syntax as follows:

```
fhir:delete/methodName?[parameters]
```

The 3 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
resource	Deletes the given resource
resourceById	Deletes the resource by resource type e
resourceConditionalByUrl	Specifies that the delete should be performed as a conditional delete against a given search URL

23.5.3.1. Method resource

Signatures:

- org.hl7.fhir.instance.model.api.IBaseOperationOutcome resource(org.hl7.fhir.instance.model.api.IBaseResource resource, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resource API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
resource	The IBaseResource to delete	IBaseResource

23.5.3.2. Method resourceById

Signatures:

- org.hl7.fhir.instance.model.api.IBaseOperationOutcome resourceById(String type, String stringId, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseOperationOutcome resourceById(org.hl7.fhir.instance.model.api.IIdType id, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resourceById API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The IIdType referencing the resource	IIdType
stringId	It's id	String

Parameter	Description	Type
type	The resource type e.g Patient	String

23.5.3.3. Method resourceConditionalByUrl

Signatures:

- org.hl7.fhir.instance.model.api.IBaseOperationOutcome resourceConditionalByUrl(String url, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resourceConditionalByUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
url	The search URL to use. The format of this URL should be of the form ResourceTypeParameters, for example: Patientname=Smith&identifier=13.2.4.11.4%7C847366	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.4. API: history

Both producer and consumer are supported

The history API is defined in the syntax as follows:

```
fhir:history/methodName?[parameters]
```

The 3 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
onInstance	Perform the operation across all versions of a specific resource (by ID and type) on the server
onServer	Perform the operation across all versions of all resources of all types on the server

Method	Description
onType	Perform the operation across all versions of all resources of the given type on the server

23.5.4.1. Method onInstance

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle onInstance(org.hl7.fhir.instance.model.api.IIdType id, Class<org.hl7.fhir.instance.model.api.IBaseBundle> returnType, Integer count, java.util.Date cutoff, org.hl7.fhir.instance.model.api.IPrimitiveType<java.util.Date> iCutoff, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onInstance API method has the parameters listed in the table below:

Parameter	Description	Type
count	Request that the server return only up to theCount number of resources, may be NULL	Integer
cutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	Date
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
iCutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	IPrimitiveType
id	The IIdType which must be populated with both a resource type and a resource ID at	IIdType
returnType	Request that the method return a Bundle resource (such as ca.uhn.fhir.model.dstu2.resource.Bundle). Use this method if you are accessing a DSTU2 server.	Class

23.5.4.2. Method onServer

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle onServer(Class<org.hl7.fhir.instance.model.api.IBaseBundle> returnType, Integer count, java.util.Date cutoff, org.hl7.fhir.instance.model.api.IPrimitiveType<java.util.Date> iCutoff, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onServer API method has the parameters listed in the table below:

Parameter	Description	Type
count	Request that the server return only up to theCount number of resources, may be NULL	Integer
cutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	Date
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
iCutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	IPrimitiveType
returnType	Request that the method return a Bundle resource (such as ca.uhn.fhir.model.dstu2.resource.Bundle). Use this method if you are accessing a DSTU2 server.	Class

23.5.4.3. Method onType

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle onType(Class<org.hl7.fhir.instance.model.api.IBaseResource> resourceType, Class<org.hl7.fhir.instance.model.api.IBaseBundle> returnType, Integer count, java.util.Date cutoff, org.hl7.fhir.instance.model.api.IPrimitiveType<java.util.Date> iCutoff, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onType API method has the parameters listed in the table below:

Parameter	Description	Type
count	Request that the server return only up to theCount number of resources, may be NULL	Integer
cutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	Date
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
iCutoff	Request that the server return only resource versions that were created at or after the given time (inclusive), may be NULL	IPrimitiveType
resourceType	The resource type to search for	Class

Parameter	Description	Type
returnType	Request that the method return a Bundle resource (such as ca.uhn.fhir.model.dstu2.resource.Bundle). Use this method if you are accessing a DSTU2 server.	Class

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.5. API: load-page

Both producer and consumer are supported

The load-page API is defined in the syntax as follows:

```
fhir:load-page/methodName?[parameters]
```

The 3 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
byUrl	Load a page of results using the given URL and bundle type and return a DSTU1 Atom bundle
next	Load the next page of results using the link with relation next in the bundle
previous	Load the previous page of results using the link with relation prev in the bundle

23.5.5.1. Method byUrl

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle byUrl(String url, Class<org.hl7.fhir.instance.model.api.IBaseBundle> returnType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/byUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map

Parameter	Description	Type
returnType	The return type	Class
url	The search url	String

23.5.5.2. Method next

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle next(org.hl7.fhir.instance.model.api.IBaseBundle bundle, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/next API method has the parameters listed in the table below:

Parameter	Description	Type
bundle	The IBaseBundle	IBaseBundle
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map

23.5.5.3. Method previous

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle previous(org.hl7.fhir.instance.model.api.IBaseBundle bundle, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/previous API method has the parameters listed in the table below:

Parameter	Description	Type
bundle	The IBaseBundle	IBaseBundle
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.6. API: meta

Both producer and consumer are supported

The meta API is defined in the syntax as follows:

```
fhir:meta/methodName?[parameters]
```

The 5 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
add	Add the elements in the given metadata to the already existing set (do not remove any)
delete	Delete the elements in the given metadata from the given id
getFromResource	Fetch the current metadata from a specific resource
getFromServer	Fetch the current metadata from the whole Server
getFromType	Fetch the current metadata from a specific type

23.5.6.1. Method add

Signatures:

- org.hl7.fhir.instance.model.api.IBaseMetaType add(org.hl7.fhir.instance.model.api.IBaseMetaType meta, org.hl7.fhir.instance.model.api.IIdType id, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/add API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The id	IIdType
meta	The IBaseMetaType class	IBaseMetaType

23.5.6.2. Method delete

Signatures:

- org.hl7.fhir.instance.model.api.IBaseMetaType delete(org.hl7.fhir.instance.model.api.IBaseMetaType meta, org.hl7.fhir.instance.model.api.IIdType id, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/delete API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The id	IIdType
meta	The IBaseMetaType class	IBaseMetaType

23.5.6.3. Method getFromResource

Signatures:

- org.hl7.fhir.instance.model.api.IBaseMetaType getFromResource(Class<org.hl7.fhir.instance.model.api.IBaseMetaType> metaType, org.hl7.fhir.instance.model.api.IIdType id, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/getFromResource API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The id	IIdType
metaType	The IBaseMetaType class	Class

23.5.6.4. Method getFromServer

Signatures:

- org.hl7.fhir.instance.model.api.IBaseMetaType getFromServer(Class<org.hl7.fhir.instance.model.api.IBaseMetaType> metaType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/getFromServer API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
metaType	The type of the meta datatype for the given FHIR model version (should be MetaDt.class or MetaType.class)	Class

23.5.6.5. Method getFromType

Signatures:

- org.hl7.fhir.instance.model.api.IBaseMetaType getFromType(Class<org.hl7.fhir.instance.model.api.IBaseMetaType> metaType, String resourceType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/getFromType API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
metaType	The IBaseMetaType class	Class
resourceType	The resource type e.g Patient	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.7. API: operation

Both producer and consumer are supported

The operation API is defined in the syntax as follows:

```
fhir:operation/methodName?[parameters]
```

The 5 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
onInstance	Perform the operation across all versions of a specific resource (by ID and type) on the server
onInstanceVersion	This operation operates on a specific version of a resource
onServer	Perform the operation across all versions of all resources of all types on the server
onType	Perform the operation across all versions of all resources of the given type on the server

Method	Description
processMessage	This operation is called $process-message as defined by the FHIR specification

23.5.7.1. Method onInstance

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource onInstance(org.hl7.fhir.instance.model.api.IIdType id, String name, org.hl7.fhir.instance.model.api.IBaseParameters parameters, Class<org.hl7.fhir.instance.model.api.IBaseParameters> outputParameterType, boolean useHttpGet, Class<org.hl7.fhir.instance.model.api.IBaseResource> returnType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onInstance API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	Resource (version will be stripped)	IIdType
name	Operation name	String
outputParameterType	The type to use for the output parameters (this should be set to Parameters.class drawn from the version of the FHIR structures you are using), may be NULL	Class
parameters	The parameters to use as input. May also be null if the operation does not require any input parameters.	IBaseParameters
returnType	If this operation returns a single resource body as its return type instead of a Parameters resource, use this method to specify that resource type. This is useful for certain operations (e.g. Patient/NNN/$everything) which return a bundle instead of a Parameters resource, may be NULL	Class
useHttpGet	Use HTTP GET verb	Boolean

23.5.7.2. Method onInstanceVersion

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource onInstanceVersion(org.hl7.fhir.instance.model.api.IIdType id, String name,

org.hl7.fhir.instance.model.api.IBaseParameters parameters, Class<org.hl7.fhir.instance.model.api.IBaseParameters> outputParameterType, boolean useHttpGet, Class<org.hl7.fhir.instance.model.api.IBaseResource> returnType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onInstanceVersion API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	Resource version	IIdType
name	Operation name	String
outputParameterType	The type to use for the output parameters (this should be set to Parameters.class drawn from the version of the FHIR structures you are using), may be NULL	Class
parameters	The parameters to use as input. May also be null if the operation does not require any input parameters.	IBaseParameters
returnType	If this operation returns a single resource body as its return type instead of a Parameters resource, use this method to specify that resource type. This is useful for certain operations (e.g. Patient/NNN/$everything) which return a bundle instead of a Parameters resource, may be NULL	Class
useHttpGet	Use HTTP GET verb	Boolean

23.5.7.3. Method onServer

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource onServer(String name, org.hl7.fhir.instance.model.api.IBaseParameters parameters, Class<org.hl7.fhir.instance.model.api.IBaseParameters> outputParameterType, boolean useHttpGet, Class<org.hl7.fhir.instance.model.api.IBaseResource> returnType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onServer API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map

Parameter	Description	Type
name	Operation name	String
outputParameterType	The type to use for the output parameters (this should be set to Parameters.class drawn from the version of the FHIR structures you are using), may be NULL	Class
parameters	The parameters to use as input. May also be null if the operation does not require any input parameters.	IBaseParameters
returnType	If this operation returns a single resource body as its return type instead of a Parameters resource, use this method to specify that resource type. This is useful for certain operations (e.g. Patient/NNN/$everything) which return a bundle instead of a Parameters resource, may be NULL	Class
useHttpGet	Use HTTP GET verb	Boolean

23.5.7.4. Method onType

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource onType(Class<org.hl7.fhir.instance.model.api.IBaseResource> resourceType, String name, org.hl7.fhir.instance.model.api.IBaseParameters parameters, Class<org.hl7.fhir.instance.model.api.IBaseParameters> outputParameterType, boolean useHttpGet, Class<org.hl7.fhir.instance.model.api.IBaseResource> returnType, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/onType API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
name	Operation name	String
outputParameterType	The type to use for the output parameters (this should be set to Parameters.class drawn from the version of the FHIR structures you are using), may be NULL	Class
parameters	The parameters to use as input. May also be null if the operation does not require any input parameters.	IBaseParameters
resourceType	The resource type to operate on	Class

Parameter	Description	Type
returnType	If this operation returns a single resource body as its return type instead of a Parameters resource, use this method to specify that resource type. This is useful for certain operations (e.g. Patient/NNN/$everything) which return a bundle instead of a Parameters resource, may be NULL	Class
useHttpGet	Use HTTP GET verb	Boolean

23.5.7.5. Method processMessage

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle processMessage(String respondToUri, org.hl7.fhir.instance.model.api.IBaseBundle msgBundle, boolean asynchronous, Class<org.hl7.fhir.instance.model.api.IBaseBundle> responseClass, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/processMessage API method has the parameters listed in the table below:

Parameter	Description	Type
asynchronous	Whether to process the message asynchronously or synchronously, defaults to synchronous.	Boolean
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
msgBundle	Set the Message Bundle to POST to the messaging server	IBaseBundle
respondToUri	An optional query parameter indicating that responses from the receiving server should be sent to this URI, may be NULL	String
responseClass	The response class	Class

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.8. API: patch

Both producer and consumer are supported

The patch API is defined in the syntax as follows:

```
fhir:patch/methodName?[parameters]
```

The 2 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
patchById	Applies the patch to the given resource ID
patchByUrl	Specifies that the update should be performed as a conditional create against a given search URL

23.5.8.1. Method patchById

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome patchById(String patchBody, String stringId, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- ca.uhn.fhir.rest.api.MethodOutcome patchById(String patchBody, org.hl7.fhir.instance.model.api.IIdType id, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/patchById API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The resource ID to patch	IIdType
patchBody	The body of the patch document serialized in either XML or JSON which conforms to	String
preferReturn	Add a Prefer header to the request, which requests that the server include or suppress the resource body as a part of the result. If a resource is returned by the server it will be parsed an accessible to the client via MethodOutcome#getResource()	PreferReturnEnum
stringId	The resource ID to patch	String

23.5.8.2. Method patchByUrl

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome patchByUrl(String patchBody, String url, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/patchByUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
patchBody	The body of the patch document serialized in either XML or JSON which conforms to	String
preferReturn	Add a Prefer header to the request, which requests that the server include or suppress the resource body as a part of the result. If a resource is returned by the server it will be parsed an accessible to the client via MethodOutcome#getResource()	PreferReturnEnum
url	The search URL to use. The format of this URL should be of the form ResourceTypeParameters, for example: Patientname=Smith&identifier=13.2.4.11.4%7C847366	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.9. API: read

Both producer and consumer are supported

The read API is defined in the syntax as follows:

```
fhir:read/methodName?[parameters]
```

The 2 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
resourceById	Reads a IBaseResource on the server by id
resourceByUrl	Reads a IBaseResource on the server by url

23.5.9.1. Method resourceById

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource resourceById(Class<org.hl7.fhir.instance.model.api.IBaseResource> resource, Long longId, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceById(Class<org.hl7.fhir.instance.model.api.IBaseResource> resource, String stringId, String version, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceById(Class<org.hl7.fhir.instance.model.api.IBaseResource> resource, org.hl7.fhir.instance.model.api.IIdType id, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceById(String resourceClass, Long longId, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceById(String resourceClass, String stringId, String ifVersionMatches, String version, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceById(String resourceClass, org.hl7.fhir.instance.model.api.IIdType id, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resourceById API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The IIdType referencing the resource	IIdType
ifVersionMatches	A version to match against the newest version on the server	String
longId	The resource ID	Long
resource	The resource to read (e.g. Patient)	Class

Parameter	Description	Type
resourceClass	The resource to read (e.g. Patient)	String
returnNull	Return null if version matches	Boolean
returnResource	Return the resource if version matches	IBaseResource
stringId	The resource ID	String
throwError	Throw error if the version matches	Boolean
version	The resource version	String

23.5.9.2. Method resourceByUrl

Signatures:

- org.hl7.fhir.instance.model.api.IBaseResource resourceByUrl(Class<org.hl7.fhir.instance.model.api.IBaseResource> resource, String url, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceByUrl(Class<org.hl7.fhir.instance.model.api.IBaseResource> resource, org.hl7.fhir.instance.model.api.IIdType iUrl, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceByUrl(String resourceClass, String url, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseResource resourceByUrl(String resourceClass, org.hl7.fhir.instance.model.api.IIdType iUrl, String ifVersionMatches, Boolean returnNull, org.hl7.fhir.instance.model.api.IBaseResource returnResource, Boolean throwError, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resourceByUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
iUrl	The IIdType referencing the resource by absolute url	IIdType

Parameter	Description	Type
ifVersionMatches	A version to match against the newest version on the server	String
resource	The resource to read (e.g. Patient)	Class
resourceClass	The resource to read (e.g. Patient.class)	String
returnNull	Return null if version matches	Boolean
returnResource	Return the resource if version matches	IBaseResource
throwError	Throw error if the version matches	Boolean
url	Referencing the resource by absolute url	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.10. API: search

Both producer and consumer are supported

The search API is defined in the syntax as follows:

```
fhir:search/methodName?[parameters]
```

The 1 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
searchByUrl	Perform a search directly by URL

23.5.10.1. Method searchByUrl

Signatures:

- org.hl7.fhir.instance.model.api.IBaseBundle searchByUrl(String url, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/searchByUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
url	The URL to search for. Note that this URL may be complete (e.g.) in which case the client's base URL will be ignored. Or it can be relative (e.g. Patientname=foo) in which case the client's base URL will be used.	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.11. API: transaction

Both producer and consumer are supported

The transaction API is defined in the syntax as follows:

```
fhir:transaction/methodName?[parameters]
```

The 2 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
withBundle	Use the given raw text (should be a Bundle resource) as the transaction input
withResources	Use a list of resources as the transaction input

23.5.11.1. Method withBundle

Signatures:

- String withBundle(String stringBundle, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- org.hl7.fhir.instance.model.api.IBaseBundle withBundle(org.hl7.fhir.instance.model.api.IBaseBundle bundle, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/withBundle API method has the parameters listed in the table below:

Parameter	Description	Type
bundle	Bundle to use in the transaction	IBaseBundle
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
stringBundle	Bundle to use in the transaction	String

23.5.11.2. Method withResources

Signatures:

- java.util.List<org.hl7.fhir.instance.model.api.IBaseResource> withResources(java.util.List<org.hl7.fhir.instance.model.api.IBaseResource> resources, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/withResources API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
resources	Resources to use in the transaction	List

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.12. API: update

Both producer and consumer are supported

The update API is defined in the syntax as follows:

```
fhir:update/methodName?[parameters]
```

The 2 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
resource	Updates a IBaseResource on the server by id

Method	Description
resourceBySearchUrl	Updates a IBaseResource on the server by search url

23.5.12.1. Method resource

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome resource(String resourceAsString, String stringId, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- ca.uhn.fhir.rest.api.MethodOutcome resource(String resourceAsString, org.hl7.fhir.instance.model.api.IIdType id, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- ca.uhn.fhir.rest.api.MethodOutcome resource(org.hl7.fhir.instance.model.api.IBaseResource resource, String stringId, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);
- ca.uhn.fhir.rest.api.MethodOutcome resource(org.hl7.fhir.instance.model.api.IBaseResource resource, org.hl7.fhir.instance.model.api.IIdType id, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resource API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
id	The IIdType referencing the resource	IIdType
preferReturn	Whether the server include or suppress the resource body as a part of the result	PreferReturnEnum
resource	The resource to update (e.g. Patient)	IBaseResource
resourceAsString	The resource body to update	String
stringId	The ID referencing the resource	String

23.5.12.2. Method resourceBySearchUrl

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome resourceBySearchUrl(String resourceAsString, String url, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

- ca.uhn.fhir.rest.api.MethodOutcome resourceBySearchUrl(org.hl7.fhir.instance.model.api.IBaseResource resource, String url, ca.uhn.fhir.rest.api.PreferReturnEnum preferReturn, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resourceBySearchUrl API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
preferReturn	Whether the server include or suppress the resource body as a part of the result	PreferReturnEnum
resource	The resource to update (e.g. Patient)	IBaseResource
resourceAsString	The resource body to update	String
url	Specifies that the update should be performed as a conditional create against a given search URL	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.5.13. API: validate

Both producer and consumer are supported

The validate API is defined in the syntax as follows:

```
fhir:validate/methodName?[parameters]
```

The 1 method(s) is listed in the table below, followed by detailed syntax for each method. (API methods can have a shorthand alias name which can be used in the syntax instead of the name)

Method	Description
resource	Validates the resource

23.5.13.1. Method resource

Signatures:

- ca.uhn.fhir.rest.api.MethodOutcome resource(String resourceAsString, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

- ca.uhn.fhir.rest.api.MethodOutcome resource(org.hl7.fhir.instance.model.api.IBaseResource resource, java.util.Map<org.apache.camel.component.fhir.api.ExtraParameters, Object> extraParameters);

The fhir/resource API method has the parameters listed in the table below:

Parameter	Description	Type
extraParameters	See ExtraParameters for a full list of parameters that can be passed, may be NULL	Map
resource	The IBaseResource to validate	IBaseResource
resourceAsString	Raw resource to validate	String

In addition to the parameters above, the fhir API can also use any of the Query Parameters.

Any of the parameters can be provided in either the endpoint URI, or dynamically in a message header. The message header name must be of the format **CamelFhir.parameter**. The **inBody** parameter overrides message header, i.e. the endpoint parameter **inBody=myParameterNameHere** would override a **CamelFhir.myParameterNameHere** header.

23.6. SPRING BOOT AUTO-CONFIGURATION

When using fhir with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-fhir-starter</artifactId>
</dependency>
```

The component supports 56 options, which are listed below.

Name	Description	Default	Type
camel.component.fhir.access-token	OAuth access token.		String
camel.component.fhir.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.fhir.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.fhir.client	To use the custom client. The option is a ca.uhn.fhir.rest.client.api.IGenericClient type.		IGenericClient
camel.component.fhir.client-factory	To use the custom client factory. The option is a ca.uhn.fhir.rest.client.api.IRestfulClientFactory type.		IRestfulClientFactory
camel.component.fhir.compress	Compresses outgoing (POST/PUT) contents to the GZIP format.	false	Boolean
camel.component.fhir.configuration	To use the shared configuration. The option is a org.apache.camel.component.fhir.FhirConfiguration type.		FhirConfiguration
camel.component.fhir.connection-timeout	How long to try and establish the initial TCP connection (in ms).	10000	Integer
camel.component.fhir.defer-model-scanning	When this option is set, model classes will not be scanned for children until the child list for the given type is actually accessed.	false	Boolean
camel.component.fhir.enabled	Whether to enable auto configuration of the fhir component. This is enabled by default.		Boolean
camel.component.fhir.encoding	Encoding to use for all request.		String
camel.component.fhir.fhir-context	FhirContext is an expensive object to create. To avoid creating multiple instances, it can be set directly. The option is a ca.uhn.fhir.context.FhirContext type.		FhirContext
camel.component.fhir.fhir-version	The FHIR Version to use.	R4	String

Name	Description	Default	Type
camel.component.fhir.force-conformance-check	Force conformance check.	false	Boolean
camel.component.fhir.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.fhir.log	Will log every requests and responses.	false	Boolean
camel.component.fhir.password	Username to use for basic authentication.		String
camel.component.fhir.pretty-print	Pretty print all request.	false	Boolean
camel.component.fhir.proxy-host	The proxy host.		String
camel.component.fhir.proxy-password	The proxy password.		String
camel.component.fhir.proxy-port	The proxy port.		Integer
camel.component.fhir.proxy-user	The proxy username.		String
camel.component.fhir.server-url	The FHIR server base URL.		String
camel.component.fhir.session-cookie	HTTP session cookie to add to every request.		String

Name	Description	Default	Type
camel.component.fhir.socket-timeout	How long to block for individual read/write operations (in ms).	10000	Integer
camel.component.fhir.summary	Request that the server modify the response using the _summary param.		String
camel.component.fhir.username	Username to use for basic authentication.		String
camel.component.fhir.validation-mode	When should Camel validate the FHIR Server's conformance statement.	ONCE	String
camel.dataformat.fhirjson.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.fhirjson.dont-encode-elements	If provided, specifies the elements which should NOT be encoded. Valid values for this field would include: Patient - Don't encode patient and all its children Patient.name - Don't encode the patient's name Patient.name.family - Don't encode the patient's family name .text - Don't encode the text element on any resource (only the very first position may contain a wildcard) DSTU2 note: Note that values including meta, such as Patient.meta will work for DSTU2 parsers, but values with subelements on meta such as Patient.meta.lastUpdated will only work in DSTU3 mode.		Set
camel.dataformat.fhirjson.dont-strip-versions-from-references-at-paths	If supplied value(s), any resource references at the specified paths will have their resource versions encoded instead of being automatically stripped during the encoding process. This setting has no effect on the parsing process. This method provides a finer-grained level of control than setStripVersionsFromReferences(String) and any paths specified by this method will be encoded even if setStripVersionsFromReferences(String) has been set to true (which is the default).		List
camel.dataformat.fhirjson.enabled	Whether to enable auto configuration of the fhirJson data format. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.dataformat.fhirjson.encode-elements	If provided, specifies the elements which should be encoded, to the exclusion of all others. Valid values for this field would include: Patient - Encode patient and all its children Patient.name - Encode only the patient's name Patient.name.family - Encode only the patient's family name .text - Encode the text element on any resource (only the very first position may contain a wildcard) .(mandatory) - This is a special case which causes any mandatory fields (min 0) to be encoded.		Set
camel.dataformat.fhirjson.encode-elements-applies-to-child-resources-only	If set to true (default is false), the values supplied to setEncodeElements(Set) will not be applied to the root resource (typically a Bundle), but will be applied to any sub-resources contained within it (i.e. search result resources in that bundle).	false	Boolean
camel.dataformat.fhirjson.fhir-version	The version of FHIR to use. Possible values are: DSTU2,DSTU2_HL7ORG,DSTU2_1,DSTU3,R4.	DSTU3	String
camel.dataformat.fhirjson.omit-resource-id	If set to true (default is false) the ID of any resources being encoded will not be included in the output. Note that this does not apply to contained resources, only to root resources. In other words, if this is set to true, contained resources will still have local IDs but the outer/containing ID will not have an ID.	false	Boolean
camel.dataformat.fhirjson.override-resource-id-with-bundle-entry-full-url	If set to true (which is the default), the Bundle.entry.fullUrl will override the Bundle.entry.resource's resource id if the fullUrl is defined. This behavior happens when parsing the source data into a Bundle object. Set this to false if this is not the desired behavior (e.g. the client code wishes to perform additional validation checks between the fullUrl and the resource id).	false	Boolean
camel.dataformat.fhirjson.pretty-print	Sets the pretty print flag, meaning that the parser will encode resources with human-readable spacing and newlines between elements instead of condensing output as much as possible.	false	Boolean
camel.dataformat.fhirjson.server-base-url	Sets the server's base URL used by this parser. If a value is set, resource references will be turned into relative references if they are provided as absolute URLs but have a base matching the given base.		String

Name	Description	Default	Type
camel.dataformat.fhirjson.strip-versions-from-references	If set to true (which is the default), resource references containing a version will have the version removed when the resource is encoded. This is generally good behaviour because in most situations, references from one resource to another should be to the resource by ID, not by ID and version. In some cases though, it may be desirable to preserve the version in resource links. In that case, this value should be set to false. This method provides the ability to globally disable reference encoding. If finer-grained control is needed, use setDontStripVersionsFromReferencesAtPaths(List).	false	Boolean
camel.dataformat.fhirjson.summary-mode	If set to true (default is false) only elements marked by the FHIR specification as being summary elements will be included.	false	Boolean
camel.dataformat.fhirjson.suppress-narratives	If set to true (default is false), narratives will not be included in the encoded values.	false	Boolean
camel.dataformat.fhirxml.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.fhirxml.dont-encode-elements	If provided, specifies the elements which should NOT be encoded. Valid values for this field would include: Patient - Don't encode patient and all its children Patient.name - Don't encode the patient's name Patient.name.family - Don't encode the patient's family name .text - Don't encode the text element on any resource (only the very first position may contain a wildcard) DSTU2 note: Note that values including meta, such as Patient.meta will work for DSTU2 parsers, but values with subelements on meta such as Patient.meta.lastUpdated will only work in DSTU3 mode.		Set
camel.dataformat.fhirxml.dont-strip-versions-from-references-at-paths	If supplied value(s), any resource references at the specified paths will have their resource versions encoded instead of being automatically stripped during the encoding process. This setting has no effect on the parsing process. This method provides a finer-grained level of control than setStripVersionsFromReferences(String) and any paths specified by this method will be encoded even if setStripVersionsFromReferences(String) has been set to true (which is the default).		List

Name	Description	Default	Type
camel.dataformat.fhirxml.enabled	Whether to enable auto configuration of the fhirXml data format. This is enabled by default.		Boolean
camel.dataformat.fhirxml.encode-elements	If provided, specifies the elements which should be encoded, to the exclusion of all others. Valid values for this field would include: Patient - Encode patient and all its children Patient.name - Encode only the patient's name Patient.name.family - Encode only the patient's family name .text - Encode the text element on any resource (only the very first position may contain a wildcard) .(mandatory) - This is a special case which causes any mandatory fields (min 0) to be encoded.		Set
camel.dataformat.fhirxml.encode-elements-applies-to-child-resources-only	If set to true (default is false), the values supplied to setEncodeElements(Set) will not be applied to the root resource (typically a Bundle), but will be applied to any sub-resources contained within it (i.e. search result resources in that bundle).	false	Boolean
camel.dataformat.fhirxml.fhir-version	The version of FHIR to use. Possible values are: DSTU2,DSTU2_HL7ORG,DSTU2_1,DSTU3,R4.	DSTU3	String
camel.dataformat.fhirxml.omit-resource-id	If set to true (default is false) the ID of any resources being encoded will not be included in the output. Note that this does not apply to contained resources, only to root resources. In other words, if this is set to true, contained resources will still have local IDs but the outer/containing ID will not have an ID.	false	Boolean
camel.dataformat.fhirxml.override-resource-id-with-bundle-entry-full-url	If set to true (which is the default), the Bundle.entry.fullUrl will override the Bundle.entry.resource's resource id if the fullUrl is defined. This behavior happens when parsing the source data into a Bundle object. Set this to false if this is not the desired behavior (e.g. the client code wishes to perform additional validation checks between the fullUrl and the resource id).	false	Boolean
camel.dataformat.fhirxml.pretty-print	Sets the pretty print flag, meaning that the parser will encode resources with human-readable spacing and newlines between elements instead of condensing output as much as possible.	false	Boolean

Name	Description	Default	Type
camel.dataformat.fhirxml.server-base-url	Sets the server's base URL used by this parser. If a value is set, resource references will be turned into relative references if they are provided as absolute URLs but have a base matching the given base.		String
camel.dataformat.fhirxml.strip-versions-from-references	If set to true (which is the default), resource references containing a version will have the version removed when the resource is encoded. This is generally good behaviour because in most situations, references from one resource to another should be to the resource by ID, not by ID and version. In some cases though, it may be desirable to preserve the version in resource links. In that case, this value should be set to false. This method provides the ability to globally disable reference encoding. If finer-grained control is needed, use setDontStripVersionsFromReferencesAtPaths(List).	false	Boolean
camel.dataformat.fhirxml.summary-mode	If set to true (default is false) only elements marked by the FHIR specification as being summary elements will be included.	false	Boolean
camel.dataformat.fhirxml.suppress-narratives	If set to true (default is false), narratives will not be included in the encoded values.	false	Boolean

CHAPTER 24. FILE

Both producer and consumer are supported

The File component provides access to file systems, allowing files to be processed by any other Camel Components or messages from other components to be saved to disk.

24.1. URI FORMAT

```
file:directoryName[?options]
```

Where **directoryName** represents the underlying file directory.

Only directories

Camel supports only endpoints configured with a starting directory. So the **directoryName** must be a directory. If you want to consume a single file only, you can use the **fileName** option, e.g. by setting **fileName=thefilename**. Also, the starting directory must not contain dynamic expressions with **${ }** placeholders. Again use the **fileName** option to specify the dynamic part of the filename.

NOTE

Avoid reading files currently being written by another application
Beware the JDK File IO API is a bit limited in detecting whether another application is currently writing/copying a file. And the implementation can be different depending on OS platform as well. This could lead to that Camel thinks the file is not locked by another process and start consuming it. Therefore you have to do you own investigation what suites your environment. To help with this Camel provides different **readLock** options and **doneFileName** option that you can use. See also the section Consuming files from folders where others drop files directly.

24.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

24.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

24.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

24.3. COMPONENT OPTIONS

The File component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

24.4. ENDPOINT OPTIONS

The File endpoint is configured using URI syntax:

```
file:directoryName
```

with the following path and query parameters:

24.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
directoryName (common)	**Required** The starting directory.		File

24.4.2. Query Parameters (94 parameters)

Name	Description	Default	Type
charset (common)	This option is used to specify the encoding of the file. You can use this on the consumer, to specify the encodings of the files, which allow Camel to know the charset it should load the file content in case the file content is being accessed. Likewise when writing a file, you can use this option to specify which charset to write the file as well. Do mind that when writing the file Camel may have to read the message content into memory to be able to convert the data into the configured charset, so do not use this if you have big messages.		String
doneFileName (common)	Producer: If provided, then Camel will write a 2nd done file when the original file has been written. The done file will be empty. This option configures what file name to use. Either you can specify a fixed name. Or you can use dynamic placeholders. The done file will always be written in the same folder as the original file. Consumer: If provided, Camel will only consume files if a done file exists. This option configures what file name to use. Either you can specify a fixed name. Or you can use dynamic placeholders.The done file is always expected in the same folder as the original file. Only $\\{file.name} and $\\{file.name.next} is supported as dynamic placeholders.		String

Name	Description	Default	Type
fileName (common)	Use Expression such as File Language to dynamically set the filename. For consumers, it's used as a filename filter. For producers, it's used to evaluate the filename to write. If an expression is set, it take precedence over the CamelFileName header. (Note: The header itself can also be an Expression). The expression options support both String and Expression types. If the expression is a String type, it is always evaluated using the File Language. If the expression is an Expression type, the specified Expression type is used - this allows you, for instance, to use OGNL expressions. For the consumer, you can use it to filter filenames, so you can for instance consume today's file using the File Language syntax: mydata-$\\{date:now:yyyyMMdd}.txt. The producers support the CamelOverruleFileName header which takes precedence over any existing CamelFileName header; the CamelOverruleFileName is a header that is used only once, and makes it easier as this avoids to temporary store CamelFileName and have to restore it afterwards.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
delete (consumer)	If true, the file will be deleted after it is processed successfully.	false	boolean
moveFailed (consumer)	Sets the move failure expression based on Simple language. For example, to move files into a .error subdirectory use: .error. Note: When moving the files to the fail location Camel will handle the error and will not pick up the file again.		String
noop (consumer)	If true, the file is not moved or deleted in any way. This option is good for readonly data, or for ETL type requirements. If noop=true, Camel will set idempotent=true as well, to avoid consuming the same files over and over again.	false	boolean

Name	Description	Default	Type
preMove (consumer)	Expression (such as File Language) used to dynamically set the filename when moving it before processing. For example to move in-progress files into the order directory set this value to order.		String
preSort (consumer)	When pre-sort is enabled then the consumer will sort the file and directory names during polling, that was retrieved from the file system. You may want to do this in case you need to operate on the files in a sorted order. The pre-sort is executed before the consumer starts to filter, and accept files to process by Camel. This option is default=false meaning disabled.	false	boolean
recursive (consumer)	If a directory, will look for files in all the sub-directories as well.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
directoryMustExist (consumer (advanced))	Similar to the startingDirectoryMustExist option but this applies during polling (after starting the consumer).	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
extendedAttributes (consumer (advanced))	To define which file attributes of interest. Like posix:permissions,posix:owner,basic:lastAccessTime, it supports basic wildcard like posix:, basic:lastAccessTime.		String

Name	Description	Default	Type
inProgressRepository (consumer (advanced))	A pluggable in-progress repository org.apache.camel.spi.IdempotentRepository. The in-progress repository is used to account the current in progress files being consumed. By default a memory based repository is used.		IdempotentRepository
localWorkDirectory (consumer (advanced))	When consuming, a local work directory can be used to store the remote file content directly in local files, to avoid loading the content into memory. This is beneficial, if you consume a very big remote file and thus can conserve memory.		String
onCompletionExceptionHandler (consumer (advanced))	To use a custom org.apache.camel.spi.ExceptionHandler to handle any thrown exceptions that happens during the file on completion process where the consumer does either a commit or rollback. The default implementation will log any exception at WARN level and ignore.		ExceptionHandler
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
probeContentType (consumer (advanced))	Whether to enable probing of the content type. If enable then the consumer uses Files#probeContentType(java.nio.file.Path) to determine the content-type of the file, and store that as a header with key Exchange#FILE_CONTENT_TYPE on the Message.	false	boolean
processStrategy (consumer (advanced))	A pluggable org.apache.camel.component.file.GenericFileProcessStrategy allowing you to implement your own readLock option or similar. Can also be used when special conditions must be met before a file can be consumed, such as a special ready file exists. If this option is set then the readLock option does not apply.		GenericFileProcessStrategy
resumeStrategy (consumer (advanced))	Set a resume strategy for files. This makes it possible to define a strategy for resuming reading files after the last point before stopping the application. See the FileConsumerResumeStrategy for implementation details.		FileConsumerResumeStrategy

Name	Description	Default	Type
startingDirectoryMustExist (consumer (advanced))	Whether the starting directory must exist. Mind that the autoCreate option is default enabled, which means the starting directory is normally auto created if it doesn't exist. You can disable autoCreate and enable this to ensure the starting directory must exist. Will thrown an exception if the directory doesn't exist.	false	boolean
startingDirectoryMustHaveAccess (consumer (advanced))	Whether the starting directory has access permissions. Mind that the startingDirectoryMustExist parameter must be set to true in order to verify that the directory exists. Will thrown an exception if the directory doesn't have read and write permissions.	false	boolean
appendChars (producer)	Used to append characters (text) after writing files. This can for example be used to add new lines or other separators when writing and appending new files or existing files. To specify new-line (slash-n or slash-r) or tab (slash-t) characters then escape with an extra slash, eg slash-slash-n.		String

Name	Description	Default	Type
fileExist (producer)	What to do if a file already exists with the same name. Override, which is the default, replaces the existing file. - Append - adds content to the existing file. - Fail - throws a GenericFileOperationException, indicating that there is already an existing file. - Ignore - silently ignores the problem and does not override the existing file, but assumes everything is okay. - Move - option requires to use the moveExisting option to be configured as well. The option eagerDeleteTargetFile can be used to control what to do if an moving the file, and there exists already an existing file, otherwise causing the move operation to fail. The Move option will move any existing files, before writing the target file. - TryRename is only applicable if tempFileName option is in use. This allows to try renaming the file from the temporary name to the actual name, without doing any exists check. This check may be faster on some file systems and especially FTP servers. Enum values: • Override • Append • Fail • Ignore • Move • TryRename	Override	GenericFileExist
flatten (producer)	Flatten is used to flatten the file name path to strip any leading paths, so it's just the file name. This allows you to consume recursively into sub-directories, but when you eg write the files to another directory they will be written in a single directory. Setting this to true on the producer enforces that any file name in CamelFileName header will be stripped for any leading paths.	false	boolean
jailStartingDirectory (producer)	Used for jailing (restricting) writing files to the starting directory (and sub) only. This is enabled by default to not allow Camel to write files to outside directories (to be more secured out of the box). You can turn this off to allow writing files to directories outside the starting directory, such as parent or root folders.	true	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
moveExisting (producer)	Expression (such as File Language) used to compute file name to use when fileExist=Move is configured. To move files into a backup subdirectory just enter backup. This option only supports the following File Language tokens: file:name, file:name.ext, file:name.noext, file:onlyname, file:onlyname.noext, file:ext, and file:parent. Notice the file:parent is not supported by the FTP component, as the FTP component can only move any existing files to a relative directory based on current dir as base.		String
tempFileName (producer)	The same as tempPrefix option but offering a more fine grained control on the naming of the temporary filename as it uses the File Language. The location for tempFilename is relative to the final file location in the option 'fileName', not the target directory in the base uri. For example if option fileName includes a directory prefix: dir/finalFilename then tempFileName is relative to that subdirectory dir.		String
tempPrefix (producer)	This option is used to write the file using a temporary name and then, after the write is complete, rename it to the real name. Can be used to identify files being written and also avoid consumers (not using exclusive read locks) reading in progress files. Is often used by FTP when uploading big files.		String
allowNullBody (producer (advanced))	Used to specify if a null body is allowed during file writing. If set to true then an empty file will be created, when set to false, and attempting to send a null body to the file component, a GenericFileWriteException of 'Cannot write null body to file.' will be thrown. If the fileExist option is set to 'Override', then the file will be truncated, and if set to append the file will remain unchanged.	false	boolean

Name	Description	Default	Type
chmod (producer (advanced))	Specify the file permissions which is sent by the producer, the chmod value must be between 000 and 777; If there is a leading digit like in 0755 we will ignore it.		String
chmodDirectory (producer (advanced))	Specify the directory permissions used when the producer creates missing directories, the chmod value must be between 000 and 777; If there is a leading digit like in 0755 we will ignore it.		String
eagerDeleteTargetFile (producer (advanced))	Whether or not to eagerly delete any existing target file. This option only applies when you use fileExists=Override and the tempFileName option as well. You can use this to disable (set it to false) deleting the target file before the temp file is written. For example you may write big files and want the target file to exists during the temp file is being written. This ensure the target file is only deleted until the very last moment, just before the temp file is being renamed to the target filename. This option is also used to control whether to delete any existing files when fileExist=Move is enabled, and an existing file exists. If this option copyAndDeleteOnRenameFails false, then an exception will be thrown if an existing file existed, if its true, then the existing file is deleted before the move operation.	true	boolean
forceWrites (producer (advanced))	Whether to force syncing writes to the file system. You can turn this off if you do not want this level of guarantee, for example if writing to logs / audit logs etc; this would yield better performance.	true	boolean
keepLastModified (producer (advanced))	Will keep the last modified timestamp from the source file (if any). Will use the Exchange.FILE_LAST_MODIFIED header to located the timestamp. This header can contain either a java.util.Date or long with the timestamp. If the timestamp exists and the option is enabled it will set this timestamp on the written file. Note: This option only applies to the file producer. You cannot use this option with any of the ftp producers.	false	boolean
moveExistingFileStrategy (producer (advanced))	Strategy (Custom Strategy) used to move file with special naming token to use when fileExist=Move is configured. By default, there is an implementation used if no custom strategy is provided.		FileMoveExistingStrategy

Name	Description	Default	Type
autoCreate (advanced)	Automatically create missing directories in the file's pathname. For the file consumer, that means creating the starting directory. For the file producer, it means the directory the files should be written to.	true	boolean
bufferSize (advanced)	Buffer size in bytes used for writing files (or in case of FTP for downloading and uploading files).	131072	int
copyAndDeleteOnRenameFail (advanced)	Whether to fallback and do a copy and delete file, in case the file could not be renamed directly. This option is not available for the FTP component.	true	boolean
renameUsingCopy (advanced)	Perform rename operations using a copy and delete strategy. This is primarily used in environments where the regular rename operation is unreliable (e.g. across different file systems or networks). This option takes precedence over the copyAndDeleteOnRenameFail parameter that will automatically fall back to the copy and delete strategy, but only after additional delays.	false	boolean
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
antExclude (filter)	Ant style filter exclusion. If both antInclude and antExclude are used, antExclude takes precedence over antInclude. Multiple exclusions may be specified in comma-delimited format.		String
antFilterCaseSensitive (filter)	Sets case sensitive flag on ant filter.	true	boolean
antInclude (filter)	Ant style filter inclusion. Multiple inclusions may be specified in comma-delimited format.		String
eagerMaxMessagesPerPoll (filter)	Allows for controlling whether the limit from maxMessagesPerPoll is eager or not. If eager then the limit is during the scanning of files. Where as false would scan all files, and then perform sorting. Setting this option to false allows for sorting all files first, and then limit the poll. Mind that this requires a higher memory usage as all file details are in memory to perform the sorting.	true	boolean

Name	Description	Default	Type
exclude (filter)	Is used to exclude files, if filename matches the regex pattern (matching is case in-sensitive). Notice if you use symbols such as plus sign and others you would need to configure this using the RAW() syntax if configuring this as an endpoint uri. See more details at configuring endpoint uris.		String
excludeExt (filter)	Is used to exclude files matching file extension name (case insensitive). For example to exclude bak files, then use excludeExt=bak. Multiple extensions can be separated by comma, for example to exclude bak and dat files, use excludeExt=bak,dat. Note that the file extension includes all parts, for example having a file named mydata.tar.gz will have extension as tar.gz. For more flexibility then use the include/exclude options.		String
filter (filter)	Pluggable filter as a org.apache.camel.component.file.GenericFileFilter class. Will skip files if filter returns false in its accept() method.		GenericFileFilter
filterDirectory (filter)	Filters the directory based on Simple language. For example to filter on current date, you can use a simple date pattern such as $\\{date:now:yyyMMdd}.		String
filterFile (filter)	Filters the file based on Simple language. For example to filter on file size, you can use $\\{file:size} 5000.		String
idempotent (filter)	Option to use the Idempotent Consumer EIP pattern to let Camel skip already processed files. Will by default use a memory based LRUCache that holds 1000 entries. If noop=true then idempotent will be enabled as well to avoid consuming the same files over and over again.	false	Boolean
idempotentKey (filter)	To use a custom idempotent key. By default the absolute path of the file is used. You can use the File Language, for example to use the file name and file size, you can do: idempotentKey=$\\{file:name}-$\\{file:size}.		String
idempotentRepository (filter)	A pluggable repository org.apache.camel.spi.IdempotentRepository which by default use MemoryIdempotentRepository if none is specified and idempotent is true.		IdempotentRepository

Name	Description	Default	Type
include (filter)	Is used to include files, if filename matches the regex pattern (matching is case in-sensitive). Notice if you use symbols such as plus sign and others you would need to configure this using the RAW() syntax if configuring this as an endpoint uri. See more details at configuring endpoint uris.		String
includeExt (filter)	Is used to include files matching file extension name (case insensitive). For example to include txt files, then use includeExt=txt. Multiple extensions can be separated by comma, for example to include txt and xml files, use includeExt=txt,xml. Note that the file extension includes all parts, for example having a file named mydata.tar.gz will have extension as tar.gz. For more flexibility then use the include/exclude options.		String
maxDepth (filter)	The maximum depth to traverse when recursively processing a directory.	2147483647	int
maxMessagesPerPoll (filter)	To define a maximum messages to gather per poll. By default no maximum is set. Can be used to set a limit of e.g. 1000 to avoid when starting up the server that there are thousands of files. Set a value of 0 or negative to disabled it. Notice: If this option is in use then the File and FTP components will limit before any sorting. For example if you have 100000 files and use maxMessagesPerPoll=500, then only the first 500 files will be picked up, and then sorted. You can use the eagerMaxMessagesPerPoll option and set this to false to allow to scan all files first and then sort afterwards.		int
minDepth (filter)	The minimum depth to start processing when recursively processing a directory. Using minDepth=1 means the base directory. Using minDepth=2 means the first sub directory.		int
move (filter)	Expression (such as Simple Language) used to dynamically set the filename when moving it after processing. To move files into a .done subdirectory just enter .done.		String
exclusiveReadLockStrategy (lock)	Pluggable read-lock as a org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy implementation.		GenericFileExclusiveReadLockStrategy
readLock (lock)	Used by consumer, to only poll the files if it has	none	String

Name	Description	Default	Type
	exclusive read-lock on the file (i.e. the file is not in-progress or being written). Camel will wait until the file lock is granted. This option provides the build in strategies: - none - No read lock is in use - markerFile - Camel creates a marker file (fileName.camelLock) and then holds a lock on it. This option is not available for the FTP component - changed - Changed is using file length/modification timestamp to detect whether the file is currently being copied or not. Will at least use 1 sec to determine this, so this option cannot consume files as fast as the others, but can be more reliable as the JDK IO API cannot always determine whether a file is currently being used by another process. The option readLockCheckInterval can be used to set the check frequency. - fileLock - is for using java.nio.channels.FileLock. This option is not avail for Windows OS and the FTP component. This approach should be avoided when accessing a remote file system via a mount/share unless that file system supports distributed file locks. - rename - rename is for using a try to rename the file as a test if we can get exclusive read-lock. - idempotent - (only for file component) idempotent is for using a idempotentRepository as the read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that. - idempotent-changed - (only for file component) idempotent-changed is for using a idempotentRepository and changed as the combined read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that. - idempotent-rename - (only for file component) idempotent-rename is for using a idempotentRepository and rename as the combined read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that.Notice: The various read locks is not all suited to work in clustered mode, where concurrent consumers on different nodes is competing for the same files on a shared file system. The markerFile using a close to atomic operation to create the empty marker file, but its not guaranteed to work in a cluster. The fileLock may work better but then the file system need to support distributed file locks, and so on. Using the idempotent read lock can support clustering if the idempotent repository supports clustering, such as Hazelcast Component or Infinispan. Enum values: - none - markerFile - fileLock		

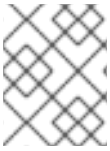
Name	Description	Default	Type
	• rename • changed • idempotent • idempotent-changed • idempotent-rename		
readLockCheckInterval (lock)	Interval in millis for the read-lock, if supported by the read lock. This interval is used for sleeping between attempts to acquire the read lock. For example when using the changed read lock, you can set a higher interval period to cater for slow writes. The default of 1 sec. may be too fast if the producer is very slow writing the file. Notice: For FTP the default readLockCheckInterval is 5000. The readLockTimeout value must be higher than readLockCheckInterval, but a rule of thumb is to have a timeout that is at least 2 or more times higher than the readLockCheckInterval. This is needed to ensure that amble time is allowed for the read lock process to try to grab the lock before the timeout was hit.	1000	long
readLockDeleteOrphanLockFiles (lock)	Whether or not read lock with marker files should upon startup delete any orphan read lock files, which may have been left on the file system, if Camel was not properly shutdown (such as a JVM crash). If turning this option to false then any orphaned lock file will cause Camel to not attempt to pickup that file, this could also be due another node is concurrently reading files from the same shared directory.	true	boolean
readLockIdempotentReleaseAsync (lock)	Whether the delayed release task should be synchronous or asynchronous. See more details at the readLockIdempotentReleaseDelay option.	false	boolean
readLockIdempotentReleaseAsyncPoolSize (lock)	The number of threads in the scheduled thread pool when using asynchronous release tasks. Using a default of 1 core threads should be sufficient in almost all use-cases, only set this to a higher value if either updating the idempotent repository is slow, or there are a lot of files to process. This option is not in-use if you use a shared thread pool by configuring the readLockIdempotentReleaseExecutorService option. See more details at the readLockIdempotentReleaseDelay option.		int

Name	Description	Default	Type
readLockIdempotentReleaseDelay (lock)	Whether to delay the release task for a period of millis. This can be used to delay the release tasks to expand the window when a file is regarded as read-locked, in an active/active cluster scenario with a shared idempotent repository, to ensure other nodes cannot potentially scan and acquire the same file, due to race-conditions. By expanding the time-window of the release tasks helps prevents these situations. Note delaying is only needed if you have configured readLockRemoveOnCommit to true.		int
readLockIdempotentReleaseExecutorService (lock)	To use a custom and shared thread pool for asynchronous release tasks. See more details at the readLockIdempotentReleaseDelay option.		ScheduledExecutorService
readLockLoggingLevel (lock)	Logging level used when a read lock could not be acquired. By default a DEBUG is logged. You can change this level, for example to OFF to not have any logging. This option is only applicable for readLock of types: changed, fileLock, idempotent, idempotent-changed, idempotent-rename, rename. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	DEBUG	LoggingLevel
readLockMarkerFile (lock)	Whether to use marker file with the changed, rename, or exclusive read lock types. By default a marker file is used as well to guard against other processes picking up the same files. This behavior can be turned off by setting this option to false. For example if you do not want to write marker files to the file systems by the Camel application.	true	boolean
readLockMinAge (lock)	This option is applied only for readLock=changed. It allows to specify a minimum age the file must be before attempting to acquire the read lock. For example use readLockMinAge=300s to require the file is at last 5 minutes old. This can speedup the changed read lock as it will only attempt to acquire files which are at least that given age.	0	long

Name	Description	Default	Type
readLockMinLength (lock)	This option is applied only for readLock=changed. It allows you to configure a minimum file length. By default Camel expects the file to contain data, and thus the default value is 1. You can set this option to zero, to allow consuming zero-length files.	1	long
readLockRemoveOnCommit (lock)	This option is applied only for readLock=idempotent. It allows to specify whether to remove the file name entry from the idempotent repository when processing the file is succeeded and a commit happens. By default the file is not removed which ensures that any race-condition do not occur so another active node may attempt to grab the file. Instead the idempotent repository may support eviction strategies that you can configure to evict the file name entry after X minutes - this ensures no problems with race conditions. See more details at the readLockIdempotentReleaseDelay option.	false	boolean
readLockRemoveOnRollback (lock)	This option is applied only for readLock=idempotent. It allows to specify whether to remove the file name entry from the idempotent repository when processing the file failed and a rollback happens. If this option is false, then the file name entry is confirmed (as if the file did a commit).	true	boolean
readLockTimeout (lock)	Optional timeout in millis for the read-lock, if supported by the read-lock. If the read-lock could not be granted and the timeout triggered, then Camel will skip the file. At next poll Camel, will try the file again, and this time maybe the read-lock could be granted. Use a value of 0 or lower to indicate forever. Currently fileLock, changed and rename support the timeout. Notice: For FTP the default readLockTimeout value is 20000 instead of 10000. The readLockTimeout value must be higher than readLockCheckInterval, but a rule of thumb is to have a timeout that is at least 2 or more times higher than the readLockCheckInterval. This is needed to ensure that amble time is allowed for the read lock process to try to grab the lock before the timeout was hit.	10000	long
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object

Name	Description	Default	Type
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
shuffle (sort)	To shuffle the list of files (sort in random order).	false	boolean
sortBy (sort)	Built-in sort by using the File Language. Supports nested sorts, so you can have a sort by file name and as a 2nd group sort by modified date.		String
sorter (sort)	Pluggable sorter as a java.util.Comparator class.		Comparator



NOTE

Default behavior for file producer
By default it will override any existing file, if one exist with the same name.

24.5. MOVE AND DELETE OPERATIONS

Any move or delete operations is executed after (post command) the routing has completed; so during processing of the **Exchange** the file is still located in the inbox folder.

Lets illustrate this with an example:

```
from("file://inbox?move=.done").to("bean:handleOrder");
```

When a file is dropped in the **inbox** folder, the file consumer notices this and creates a new **FileExchange** that is routed to the **handleOrder** bean. The bean then processes the **File** object. At this point in time the file is still located in the **inbox** folder. After the bean completes, and thus the route is completed, the file consumer will perform the move operation and move the file to the **.done** sub-folder.

The **move** and the **preMove** options are considered as a directory name (though if you use an expression such as File Language, or Simple then the result of the expression evaluation is the file name to be used. For example, if you set:

```
move=../backup/copy-of-${file:name}
```

then that's using the File language which we use return the file name to be used), which can be either relative or absolute. If relative, the directory is created as a sub-folder from within the folder where the file was consumed.

By default, Camel will move consumed files to the **.camel** sub-folder relative to the directory where the file was consumed.

If you want to delete the file after processing, the route should be:

```
from("file://inbox?delete=true").to("bean:handleOrder");
```

We have introduced a **pre** move operation to move files **before** they are processed. This allows you to mark which files have been scanned as they are moved to this sub folder before being processed.

```
from("file://inbox?preMove=inprogress").to("bean:handleOrder");
```

You can combine the **pre** move and the regular move:

```
from("file://inbox?preMove=inprogress&move=.done").to("bean:handleOrder");
```

So in this situation, the file is in the **inprogress** folder when being processed and after it's processed, it's moved to the **.done** folder.

24.6. FINE GRAINED CONTROL OVER MOVE AND PREMOVE OPTION

The **move** and **preMove** options are Expression-based, so we have the full power of the File Language to do advanced configuration of the directory and name pattern.
Camel will, in fact, internally convert the directory name you enter into a File Language expression. So when we enter **move=.done** Camel will convert this into: **${file:parent}/.done/${file:onlyname}**. This is only done if Camel detects that you have not provided a $\{ } in the option value yourself. So when you enter a $\{ } Camel will **not** convert it and thus you have the full power.

So if we want to move the file into a backup folder with today's date as the pattern, we can do:

```
move=backup/${date:now:yyyyMMdd}/${file:name}
```

24.7. ABOUT MOVEFAILED

The **moveFailed** option allows you to move files that **could not** be processed successfully to another location such as an error folder of your choice. For example to move the files in an error folder with a

timestamp you can use **moveFailed=/error/${file:name.noext}-${date:now:yyyyMMddHHmmssSSS}.${\'file:ext}**.

See more examples at

24.8. MESSAGE HEADERS

The following headers are supported by this component:

24.8.1. File producer only

Header	Description
CamelFileName	Specifies the name of the file to write (relative to the endpoint directory). This name can be a **String**; a **String** with a File Language or Simple language expression; or an Expression object. If it's **null** then Camel will auto-generate a filename based on the message unique ID.
CamelFileNameProduced	The actual absolute filepath (path + name) for the output file that was written. This header is set by Camel and its purpose is providing end-users with the name of the file that was written.
CamelOverruleFileName	Is used for overruling **CamelFileName** header and use the value instead (but only once, as the producer will remove this header after writing the file). The value can be only be a String. Notice that if the option **fileName** has been configured, then this is still being evaluated.

24.8.2. File consumer only

Header	Description
CamelFileName	Name of the consumed file as a relative file path with offset from the starting directory configured on the endpoint.
CamelFileNameOnly	Only the file name (the name with no leading paths).
CamelFileAbsolute	A **boolean** option specifying whether the consumed file denotes an absolute path or not. Should normally be **false** for relative paths. Absolute paths should normally not be used but we added to the move option to allow moving files to absolute paths. But can be used elsewhere as well.
CamelFileAbsolutePath	The absolute path to the file. For relative files this path holds the relative path instead.
CamelFilePath	The file path. For relative files this is the starting directory + the relative filename. For absolute files this is the absolute path.
CamelFileRelativePath	The relative path.

Header	Description
CamelFileParent	The parent path.
CamelFileLength	A **long** value containing the file size.
CamelFileLastModified	A **Long** value containing the last modified timestamp of the file.

24.9. BATCH CONSUMER

This component implements the Batch Consumer.

24.10. EXCHANGE PROPERTIES, FILE CONSUMER ONLY

As the file consumer implements the **BatchConsumer** it supports batching the files it polls. By batching we mean that Camel will add the following additional properties to the Exchange, so you know the number of files polled, the current index, and whether the batch is already completed.

Property	Description
CamelBatchSize	The total number of files that was polled in this batch.
CamelBatchIndex	The current index of the batch. Starts from 0.
CamelBatchComplete	A **boolean** value indicating the last Exchange in the batch. Is only **true** for the last entry.

This allows you for instance to know how many files exist in this batch and for instance let the Aggregator2 aggregate this number of files.

24.11. USING CHARSET

The charset option allows for configuring an encoding of the files on both the consumer and producer endpoints. For example if you read utf-8 files, and want to convert the files to iso-8859-1, you can do:

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

You can also use the **convertBodyTo** in the route. In the example below we have still input files in utf-8 format, but we want to convert the file content to a byte array in iso-8859-1 format. And then let a bean process the data. Before writing the content to the outbox folder using the current charset.

```
from("file:inbox?charset=utf-8")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

If you omit the charset on the consumer endpoint, then Camel does not know the charset of the file, and would by default use "UTF-8". However you can configure a JVM system property to override and use a different default encoding with the key **org.apache.camel.default.charset**.

In the example below this could be a problem if the files is not in UTF-8 encoding, which would be the default encoding for read the files.
In this example when writing the files, the content has already been converted to a byte array, and thus would write the content directly as is (without any further encodings).

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .to("file:outbox");
```

You can also override and control the encoding dynamic when writing files, by setting a property on the exchange with the key **Exchange.CHARSET_NAME**. For example in the route below we set the property with a value from a message header.

```
from("file:inbox")
  .convertBodyTo(byte[].class, "iso-8859-1")
  .to("bean:myBean")
  .setProperty(Exchange.CHARSET_NAME, header("someCharsetHeader"))
  .to("file:outbox");
```

We suggest to keep things simpler, so if you pickup files with the same encoding, and want to write the files in a specific encoding, then favor to use the **charset** option on the endpoints.

Notice that if you have explicit configured a **charset** option on the endpoint, then that configuration is used, regardless of the **Exchange.CHARSET_NAME** property.

If you have some issues then you can enable DEBUG logging on **org.apache.camel.component.file**, and Camel logs when it reads/write a file using a specific charset.
For example the route below will log the following:

```
from("file:inbox?charset=utf-8")
  .to("file:outbox?charset=iso-8859-1")
```

And the logs:

```
DEBUG GenericFileConverter          - Read file /Users/davsclaus/workspace/camel/camel-
core/target/charset/input/input.txt with charset utf-8
DEBUG FileOperations                - Using Reader to write file: target/charset/output.txt with charset:
iso-8859-1
```

24.12. COMMON GOTCHAS WITH FOLDER AND FILENAMES

When Camel is producing files (writing files) there are a few gotchas affecting how to set a filename of your choice. By default, Camel will use the message ID as the filename, and since the message ID is normally a unique generated ID, you will end up with filenames such as: **ID-MACHINENAME-2443-1211718892437-1-0**. If such a filename is not desired, then you must provide a filename in the **CamelFileName** message header. The constant, **Exchange.FILE_NAME**, can also be used.

The sample code below produces files using the message ID as the filename:

```
from("direct:report").to("file:target/reports");
```

To use **report.txt** as the filename you have to do:

```
from("direct:report").setHeader(Exchange.FILE_NAME, constant("report.txt")).to( "file:target/reports");
```

- the same as above, but with **CamelFileName**:

```
from("direct:report").setHeader("CamelFileName", constant("report.txt")).to( "file:target/reports");
```

And a syntax where we set the filename on the endpoint with the **fileName** URI option.

```
from("direct:report").to("file:target/reports/?fileName=report.txt");
```

24.13. FILENAME EXPRESSION

Filename can be set either using the **expression** option or as a string-based File language expression in the **CamelFileName** header. See the File language for syntax and samples.

24.14. CONSUMING FILES FROM FOLDERS WHERE OTHERS DROP FILES DIRECTLY

Beware if you consume files from a folder where other applications write files to directly. Take a look at the different readLock options to see what suits your use cases. The best approach is however to write to another folder and after the write move the file in the drop folder. However if you write files directly to the drop folder then the option changed could better detect whether a file is currently being written/copied as it uses a file changed algorithm to see whether the file size / modification changes over a period of time. The other readLock options rely on Java File API that sadly is not always very good at detecting this. You may also want to look at the doneFileName option, which uses a marker file (done file) to signal when a file is done and ready to be consumed.

24.15. USING DONE FILES

See also section *writing done files* below.

If you want only to consume files when a done file exists, then you can use the **doneFileName** option on the endpoint.

```
from("file:bar?doneFileName=done");
```

Will only consume files from the bar folder, if a done *file* exists in the same directory as the target files. Camel will automatically delete the *done file* when it's done consuming the files. Camel does not delete automatically the *done file* if **noop=true** is configured.

However it is more common to have one *done file* per target file. This means there is a 1:1 correlation. To do this you must use dynamic placeholders in the **doneFileName** option. Currently Camel supports the following two dynamic tokens: **file:name** and **file:name.noext** which must be enclosed in $\{ }. The consumer only supports the static part of the *done file* name as either prefix or suffix (not both).

```
from("file:bar?doneFileName=${file:name}.done");
```

In this example only files will be polled if there exists a done file with the name *file name*.done. For example

- **hello.txt** - is the file to be consumed
- **hello.txt.done** - is the associated done file

You can also use a prefix for the done file, such as:

```
from("file:bar?doneFileName=ready-${file:name}");
```

- **hello.txt** - is the file to be consumed
- **ready-hello.txt** - is the associated done file

24.16. WRITING DONE FILES

After you have written a file you may want to write an additional *donefile* as a kind of marker, to indicate to others that the file is finished and has been written. To do that you can use the **doneFileName** option on the file producer endpoint.

```
.to("file:bar?doneFileName=done");
```

Will simply create a file named **done** in the same directory as the target file.

However it is more common to have one done file per target file. This means there is a 1:1 correlation. To do this you must use dynamic placeholders in the **doneFileName** option. Currently Camel supports the following two dynamic tokens: **file:name** and **file:name.noext** which must be enclosed in $\{ }.

```
.to("file:bar?doneFileName=done-${file:name}");
```

Will for example create a file named **done-foo.txt** if the target file was **foo.txt** in the same directory as the target file.

```
.to("file:bar?doneFileName=${file:name}.done");
```

Will for example create a file named **foo.txt.done** if the target file was **foo.txt** in the same directory as the target file.

```
.to("file:bar?doneFileName=${file:name.noext}.done");
```

Will for example create a file named **foo.done** if the target file was **foo.txt** in the same directory as the target file.

24.17. SAMPLES

24.17.1. Read from a directory and write to another directory

```
from("file://inputdir/?delete=true").to("file://outputdir")
```

24.17.2. Read from a directory and write to another directory using a overrule dynamic name

```
from("file://inputdir/?delete=true").to("file://outputdir?overruleFile=copy-of-${file:name}")
```

Listen on a directory and create a message for each file dropped there. Copy the contents to the **outputdir** and delete the file in the **inputdir**.

24.17.3. Reading recursively from a directory and writing to another

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

Listen on a directory and create a message for each file dropped there. Copy the contents to the **outputdir** and delete the file in the **inputdir**. Will scan recursively into sub-directories. Will lay out the files in the same directory structure in the **outputdir** as the **inputdir**, including any sub-directories.

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

Will result in the following output layout:

```
outputdir/foo.txt
outputdir/sub/bar.txt
```

24.18. USING FLATTEN

If you want to store the files in the outputdir directory in the same directory, disregarding the source directory layout (e.g. to flatten out the path), you just add the **flatten=true** option on the file producer side:

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir?flatten=true")
```

Will result in the following output layout:

```
outputdir/foo.txt
outputdir/bar.txt
```

24.19. READING FROM A DIRECTORY AND THE DEFAULT MOVE OPERATION

Camel will by default move any processed file into a **.camel** subdirectory in the directory the file was consumed from.

```
from("file://inputdir/?recursive=true&delete=true").to("file://outputdir")
```

Affects the layout as follows:
before

```
inputdir/foo.txt
inputdir/sub/bar.txt
```

after

```
inputdir/.camel/foo.txt
inputdir/sub/.camel/bar.txt
outputdir/foo.txt
outputdir/sub/bar.txt
```

24.20. READ FROM A DIRECTORY AND PROCESS THE MESSAGE IN JAVA

```
from("file://inputdir/").process(new Processor() {
  public void process(Exchange exchange) throws Exception {
    Object body = exchange.getIn().getBody();
    // do some business logic with the input body
  }
});
```

The body will be a **File** object that points to the file that was just dropped into the **inputdir** directory.

24.21. WRITING TO FILES

Camel is of course also able to write files, i.e. produce files. In the sample below we receive some reports on the SEDA queue that we process before they are being written to a directory.

24.21.1. Write to subdirectory using Exchange.FILE_NAME

Using a single route, it is possible to write a file to any number of subdirectories. If you have a route setup as such:

```
<route>
  <from uri="bean:myBean"/>
  <to uri="file:/rootDirectory"/>
</route>
```

You can have **myBean** set the header **Exchange.FILE_NAME** to values such as:

```
Exchange.FILE_NAME = hello.txt => /rootDirectory/hello.txt
Exchange.FILE_NAME = foo/bye.txt => /rootDirectory/foo/bye.txt
```

This allows you to have a single route to write files to multiple destinations.

24.21.2. Writing file through the temporary directory relative to the final destination

Sometime you need to temporarily write the files to some directory relative to the destination directory. Such situation usually happens when some external process with limited filtering capabilities is reading from the directory you are writing to. In the example below files will be written to the **/var/myapp/filesInProgress** directory and after data transfer is done, they will be atomically moved to the` /var/myapp/finalDirectory `directory.

```
from("direct:start").
  to("file:///var/myapp/finalDirectory?tempPrefix=/../filesInProgress/");
```

24.22. USING EXPRESSION FOR FILENAMES

In this sample we want to move consumed files to a backup folder using today's date as a sub-folder name:

```
from("file://inbox?move=backup/${date:now:yyyyMMdd}/${file:name}").to("...");
```

See File language for more samples.

24.23. AVOIDING READING THE SAME FILE MORE THAN ONCE (IDEMPOTENT CONSUMER)

Camel supports Idempotent Consumer directly within the component so it will skip already processed files. This feature can be enabled by setting the **idempotent=true** option.

```
from("file://inbox?idempotent=true").to("...");
```

Camel uses the absolute file name as the idempotent key, to detect duplicate files. You can customize this key by using an expression in the idempotentKey option. For example to use both the name and the file size as the key

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentKey=${file:name}-${file:size}"/>
  <to uri="bean:processInbox"/>
</route>
```

By default Camel uses a in memory based store for keeping track of consumed files, it uses a least recently used cache holding up to 1000 entries. You can plugin your own implementation of this store by using the **idempotentRepository** option using the # sign in the value to indicate it's a referring to a bean in the Registry with the specified **id**.

```
 <!-- define our store as a plain spring bean -->
 <bean id="myStore" class="com.mycompany.MyIdempotentStore"/>

<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#myStore"/>
  <to uri="bean:processInbox"/>
</route>
```

Camel will log at **DEBUG** level if it skips a file because it has been consumed before:

```
DEBUG FileConsumer is idempotent and the file has been consumed before. Will skip this file:
target\idempotent\report.txt
```

24.24. USING A FILE BASED IDEMPOTENT REPOSITORY

In this section we will use the file based idempotent repository **org.apache.camel.processor.idempotent.FileIdempotentRepository** instead of the in-memory based that is used as default.
This repository uses a 1st level cache to avoid reading the file repository. It will only use the file repository to store the content of the 1st level cache. Thereby the repository can survive server restarts. It will load the content of the file into the 1st level cache upon startup. The file structure is very simple as

it stores the key in separate lines in the file. By default, the file store has a size limit of 1mb. When the file grows larger Camel will truncate the file store, rebuilding the content by flushing the 1st level cache into a fresh empty file.

We configure our repository using Spring XML creating our file idempotent repository and define our file consumer to use our repository with the **idempotentRepository** using # sign to indicate Registry lookup:

24.25. USING A JPA BASED IDEMPOTENT REPOSITORY

In this section we will use the JPA based idempotent repository instead of the in-memory based that is used as default.

First we need a persistence-unit in **META-INF/persistence.xml** where we need to use the class **org.apache.camel.processor.idempotent.jpa.MessageProcessed** as model.

```
<persistence-unit name="idempotentDb" transaction-type="RESOURCE_LOCAL">
  <class>org.apache.camel.processor.idempotent.jpa.MessageProcessed</class>

  <properties>
    <property name="openjpa.ConnectionURL" value="jdbc:derby:target/idempotentTest;create=true"/>
    <property name="openjpa.ConnectionDriverName"
value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema"/>
    <property name="openjpa.Log" value="DefaultLevel=WARN, Tool=INFO"/>
    <property name="openjpa.Multithreaded" value="true"/>
  </properties>
</persistence-unit>
```

Next, we can create our JPA idempotent repository in the spring XML file as well:

```
<!-- we define our jpa based idempotent repository we want to use in the file consumer -->
<bean id="jpaStore" class="org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository">
    <!-- Here we refer to the entityManagerFactory -->
    <constructor-arg index="0" ref="entityManagerFactory"/>
    <!-- This 2nd parameter is the name  (= a category name).
         You can have different repositories with different names -->
    <constructor-arg index="1" value="FileConsumer"/>
</bean>
```

And yes then we just need to refer to the **jpaStore** bean in the file consumer endpoint using the **idempotentRepository** using the # syntax option:

```
<route>
  <from uri="file://inbox?idempotent=true&amp;idempotentRepository=#jpaStore"/>
  <to uri="bean:processInbox"/>
</route>
```

24.26. FILTER USING ORG.APACHE.CAMEL.COMPONENT.FILE.GENERICFILEFILTER

Camel supports pluggable filtering strategies. You can then configure the endpoint with such a filter to skip certain files being processed.

In the sample we have built our own filter that skips files starting with **skip** in the filename:

And then we can configure our route using the **filter** attribute to reference our filter (using # notation) that we have defined in the spring XML file:

```
<!-- define our filter as a plain spring bean -->
<bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="file://inbox?filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

24.27. FILTERING USING ANT PATH MATCHER

The ANT path matcher is based on AntPathMatcher.

The file paths is matched with the following rules:

- **?** matches one character
- ***** matches zero or more characters
- ****** matches zero or more directories in a path

The **antInclude** and **antExclude** options make it easy to specify ANT style include/exclude without having to define the filter. See the URI options above for more information.

The sample below demonstrates how to use it:

24.27.1. Sorting using Comparator

Camel supports pluggable sorting strategies. This strategy it to use the build in **java.util.Comparator** in Java. You can then configure the endpoint with such a comparator and have Camel sort the files before being processed.

In the sample we have built our own comparator that just sorts by file name:

And then we can configure our route using the **sorter** option to reference to our sorter (**mySorter**) we have defined in the spring XML file:

```
<!-- define our sorter as a plain spring bean -->
<bean id="mySorter" class="com.mycompany.MyFileSorter"/>

<route>
  <from uri="file://inbox?sorter=#mySorter"/>
  <to uri="bean:processInbox"/>
</route>
```

NOTE

URI options can reference beans using the # syntax
In the Spring DSL route above notice that we can refer to beans in the Registry by prefixing the id with #. So writing **sorter=#mySorter**, will instruct Camel to go look in the Registry for a bean with the ID, **mySorter**.

24.27.2. Sorting using sortBy

Camel supports pluggable sorting strategies. This strategy it to use the File language to configure the sorting. The **sortBy** option is configured as follows:

```
sortBy=group 1;group 2;group 3;...
```

Where each group is separated with semi colon. In the simple situations you just use one group, so a simple example could be:

```
sortBy=file:name
```

This will sort by file name, you can reverse the order by prefixing **reverse:** to the group, so the sorting is now Z..A:

```
sortBy=reverse:file:name
```

As we have the full power of File language we can use some of the other parameters, so if we want to sort by file size we do:

```
sortBy=file:length
```

You can configure to ignore the case, using **ignoreCase:** for string comparison, so if you want to use file name sorting but to ignore the case then we do:

```
sortBy=ignoreCase:file:name
```

You can combine ignore case and reverse, however reverse must be specified first:

```
sortBy=reverse:ignoreCase:file:name
```

In the sample below we want to sort by last modified file, so we do:

```
sortBy=file:modified
```

And then we want to group by name as a 2nd option so files with same modifcation is sorted by name:

```
sortBy=file:modified;file:name
```

Now there is an issue here, can you spot it? Well the modified timestamp of the file is too fine as it will be in milliseconds, but what if we want to sort by date only and then subgroup by name?
Well as we have the true power of File language we can use its date command that supports patterns. So this can be solved as:

```
sortBy=date:file:yyyyMMdd;file:name
```

Yeah, that is pretty powerful, oh by the way you can also use reverse per group, so we could reverse the file names:

```
sortBy=date:file:yyyyMMdd;reverse:file:name
```

24.28. USING GENERICFILEPROCESSSTRATEGY

The option **processStrategy** can be used to use a custom **GenericFileProcessStrategy** that allows you to implement your own *begin*, *commit* and *rollback* logic.
For instance lets assume a system writes a file in a folder you should consume. But you should not start consuming the file before another *ready* file has been written as well.

So by implementing our own **GenericFileProcessStrategy** we can implement this as:

- In the **begin()** method we can test whether the special *ready* file exists. The begin method returns a **boolean** to indicate if we can consume the file or not.
- In the **abort()** method special logic can be executed in case the **begin** operation returned **false**, for example to cleanup resources etc.
- in the **commit()** method we can move the actual file and also delete the *ready* file.

24.29. USING FILTER

The **filter** option allows you to implement a custom filter in Java code by implementing the **org.apache.camel.component.file.GenericFileFilter** interface. This interface has an **accept** method that returns a boolean. Return **true** to include the file, and **false** to skip the file. There is a **isDirectory** method on **GenericFile** whether the file is a directory. This allows you to filter unwanted directories, to avoid traversing down unwanted directories.

For example to skip any directories which starts with **"skip"** in the name, can be implemented as follows:

24.30. USING BRIDGEERRORHANDLER

If you want to use the Camel Error Handler to deal with any exception occurring in the file consumer, then you can enable the **bridgeErrorHandler** option as shown below:

```
// to handle any IOException being thrown
onException(IOException.class)
    .handled(true)
    .log("IOException occurred due: ${exception.message}")
    .transform().simple("Error ${exception.message}")
    .to("mock:error");

// this is the file route that pickup files, notice how we bridge the consumer to use the Camel routing
error handler
// the exclusiveReadLockStrategy is only configured because this is from an unit test, so we use that
to simulate exceptions
from("file:target/nospace?bridgeErrorHandler=true")
    .convertBodyTo(String.class)
    .to("mock:result");
```

So all you have to do is to enable this option, and the error handler in the route will take it from there.

IMPORTANT

When using **bridgeErrorHandler**
When using bridgeErrorHandler, then interceptors, OnCompletions does **not** apply. The Exchange is processed directly by the Camel Error Handler, and does not allow prior actions such as interceptors, onCompletion to take action.

24.31. DEBUG LOGGING

This component has log level **TRACE** that can be helpful if you have problems.

24.32. SPRING BOOT AUTO-CONFIGURATION

When using file with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-file-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.cluster.file.acquire-lock-delay	The time to wait before starting to try to acquire lock.		String
camel.cluster.file.acquire-lock-interval	The time to wait between attempts to try to acquire lock.		String
camel.cluster.file.attributes	Custom service attributes.		Map
camel.cluster.file.enabled	Sets if the file cluster service should be enabled or not, default is false.	false	Boolean
camel.cluster.file.id	Cluster Service ID.		String
camel.cluster.file.order	Service lookup order/priority.		Integer
camel.cluster.file.root	The root path.		String

Name	Description	Default	Type
camel.component.file.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.file.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.file.enabled	Whether to enable auto configuration of the file component. This is enabled by default.		Boolean
camel.component.file.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 25. FTP

Both producer and consumer are supported

This component provides access to remote file systems over the FTP and SFTP protocols.

When consuming from remote FTP server make sure you read the section titled *Default when consuming files* further below for details related to consuming files.

Absolute path is **not** supported. Camel translates absolute path to relative by trimming all leading slashes from **directoryname**. There'll be WARN message printed in the logs.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-ftp</artifactId>
  <version>{CamelSBVersion}</version>See the documentation of the Apache Commons
  <!-- use the same version as your Camel core version -->
</dependency>
```

25.1. URI FORMAT

```
ftp://[username@]hostname[:port]/directoryname[?options]
sftp://[username@]hostname[:port]/directoryname[?options]
ftps://[username@]hostname[:port]/directoryname[?options]
```

Where **directoryname** represents the underlying directory. The directory name is a relative path. Absolute path's is **not** supported. The relative path can contain nested folders, such as /inbox/us.

The **autoCreate** option is supported. When consumer starts, before polling is scheduled, there's additional FTP operation performed to create the directory configured for endpoint. The default value for **autoCreate** is **true**.

If no **username** is provided, then **anonymous** login is attempted using no password.
If no **port** number is provided, Camel will provide default values according to the protocol (ftp = 21, sftp = 22, ftps = 2222).

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

This component uses two different libraries for the actual FTP work. FTP and FTPS uses Apache Commons Net while SFTP uses JCraft JSCH.

FTPS (also known as FTP Secure) is an extension to FTP that adds support for the Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) cryptographic protocols.

25.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

25.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

25.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

25.3. COMPONENT OPTIONS

The FTP component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

25.4. ENDPOINT OPTIONS

The FTP endpoint is configured using URI syntax:

```
ftp:host:port/directoryName
```

with the following path and query parameters:

25.4.1. Path Parameters (3 parameters)

Name	Description	Default	Type
host (common)	**Required** Hostname of the FTP server.		String
port (common)	Port of the FTP server.		int
directoryName (common)	The starting directory.		String

25.4.2. Query Parameters (111 parameters)

Name	Description	Default	Type
binary (common)	Specifies the file transfer mode, BINARY or ASCII. Default is ASCII (false).	false	boolean
charset (common)	This option is used to specify the encoding of the file. You can use this on the consumer, to specify the encodings of the files, which allow Camel to know the charset it should load the file content in case the file content is being accessed. Likewise when writing a file, you can use this option to specify which charset to write the file as well. Do mind that when writing the file Camel may have to read the message content into memory to be able to convert the data into the configured charset, so do not use this if you have big messages.		String
disconnect (common)	Whether or not to disconnect from remote FTP server right after use. Disconnect will only disconnect the current connection to the FTP server. If you have a consumer which you want to stop, then you need to stop the consumer/route instead.	false	boolean
doneFileName (common)	Producer: If provided, then Camel will write a 2nd done file when the original file has been written. The done file will be empty. This option configures what file name to use. Either you can specify a fixed name. Or you can use dynamic placeholders. The done file will always be written in the same folder as the original file. Consumer: If provided, Camel will only consume files if a done file exists. This option configures what file name to use. Either you can specify a fixed name. Or you can use dynamic placeholders.The done file is always expected in the same folder as the original file. Only $\\{file.name} and $\\{file.name.next} is supported as dynamic placeholders.		String

Name	Description	Default	Type
fileName (common)	Use Expression such as File Language to dynamically set the filename. For consumers, it's used as a filename filter. For producers, it's used to evaluate the filename to write. If an expression is set, it take precedence over the CamelFileName header. (Note: The header itself can also be an Expression). The expression options support both String and Expression types. If the expression is a String type, it is always evaluated using the File Language. If the expression is an Expression type, the specified Expression type is used - this allows you, for instance, to use OGNL expressions. For the consumer, you can use it to filter filenames, so you can for instance consume today's file using the File Language syntax: mydata-$\\{date:now:yyyyMMdd}.txt. The producers support the CamelOverruleFileName header which takes precedence over any existing CamelFileName header; the CamelOverruleFileName is a header that is used only once, and makes it easier as this avoids to temporary store CamelFileName and have to restore it afterwards.		String
passiveMode (common)	Sets passive mode connections. Default is active mode connections.	false	boolean
separator (common)	Sets the path separator to be used. UNIX = Uses unix style path separator Windows = Uses windows style path separator Auto = (is default) Use existing path separator in file name. Enum values: • UNIX • Windows • Auto	UNIX	PathSeparator
transferLoggingIntervalSeconds (common)	Configures the interval in seconds to use when logging the progress of upload and download operations that are in-flight. This is used for logging progress when operations takes longer time.	5	int

Name	Description	Default	Type
transferLoggingLevel (common)	Configure the logging level to use when logging the progress of upload and download operations. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	DEBUG	LoggingLevel
transferLoggingVerbose (common)	Configures whether the perform verbose (fine grained) logging of the progress of upload and download operations.	false	boolean
fastExistsCheck (common (advanced))	If set this option to be true, camel-ftp will use the list file directly to check if the file exists. Since some FTP server may not support to list the file directly, if the option is false, camel-ftp will use the old way to list the directory and check if the file exists. This option also influences readLock=changed to control whether it performs a fast check to update file information or not. This can be used to speed up the process if the FTP server has a lot of files.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
delete (consumer)	If true, the file will be deleted after it is processed successfully.	false	boolean
moveFailed (consumer)	Sets the move failure expression based on Simple language. For example, to move files into a .error subdirectory use: .error. Note: When moving the files to the fail location Camel will handle the error and will not pick up the file again.		String

Name	Description	Default	Type
noop (consumer)	If true, the file is not moved or deleted in any way. This option is good for readonly data, or for ETL type requirements. If noop=true, Camel will set idempotent=true as well, to avoid consuming the same files over and over again.	false	boolean
preMove (consumer)	Expression (such as File Language) used to dynamically set the filename when moving it before processing. For example to move in-progress files into the order directory set this value to order.		String
preSort (consumer)	When pre-sort is enabled then the consumer will sort the file and directory names during polling, that was retrieved from the file system. You may want to do this in case you need to operate on the files in a sorted order. The pre-sort is executed before the consumer starts to filter, and accept files to process by Camel. This option is default=false meaning disabled.	false	boolean
recursive (consumer)	If a directory, will look for files in all the sub-directories as well.	false	boolean
resumeDownload (consumer)	Configures whether resume download is enabled. This must be supported by the FTP server (almost all FTP servers support it). In addition the options localWorkDirectory must be configured so downloaded files are stored in a local directory, and the option binary must be enabled, which is required to support resuming of downloads.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
streamDownload (consumer)	Sets the download method to use when not using a local working directory. If set to true, the remote files are streamed to the route as they are read. When set to false, the remote files are loaded into memory before being sent into the route. If enabling this option then you must set stepwise=false as both cannot be enabled at the same time.	false	boolean
download (consumer (advanced))	Whether the FTP consumer should download the file. If this option is set to false, then the message body will be null, but the consumer will still trigger a Camel Exchange that has details about the file such as file name, file size, etc. It's just that the file will not be downloaded.	false	boolean

Name	Description	Default	Type
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern
handleDirectoryParserAbsoluteResult (consumer (advanced))	Allows you to set how the consumer will handle subfolders and files in the path if the directory parser results in with absolute paths The reason for this is that some FTP servers may return file names with absolute paths, and if so then the FTP component needs to handle this by converting the returned path into a relative path.	false	boolean
ignoreFileNotFoundOrPermissionError (consumer (advanced))	Whether to ignore when (trying to list files in directories or when downloading a file), which does not exist or due to permission error. By default when a directory or file does not exists or insufficient permission, then an exception is thrown. Setting this option to true allows to ignore that instead.	false	boolean
inProgressRepository (consumer (advanced))	A pluggable in-progress repository org.apache.camel.spi.IdempotentRepository. The in-progress repository is used to account the current in progress files being consumed. By default a memory based repository is used.		IdempotentRepository
localWorkDirectory (consumer (advanced))	When consuming, a local work directory can be used to store the remote file content directly in local files, to avoid loading the content into memory. This is beneficial, if you consume a very big remote file and thus can conserve memory.		String

Name	Description	Default	Type
onCompletionExceptionHandler (consumer (advanced))	To use a custom org.apache.camel.spi.ExceptionHandler to handle any thrown exceptions that happens during the file on completion process where the consumer does either a commit or rollback. The default implementation will log any exception at WARN level and ignore.		ExceptionHandler
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
processStrategy (consumer (advanced))	A pluggable org.apache.camel.component.file.GenericFileProcessStrategy allowing you to implement your own readLock option or similar. Can also be used when special conditions must be met before a file can be consumed, such as a special ready file exists. If this option is set then the readLock option does not apply.		GenericFileProcessStrategy
useList (consumer (advanced))	Whether to allow using LIST command when downloading a file. Default is true. In some use cases you may want to download a specific file and are not allowed to use the LIST command, and therefore you can set this option to false. Notice when using this option, then the specific file to download does not include meta-data information such as file size, timestamp, permissions etc, because those information is only possible to retrieve when LIST command is in use.	true	boolean

Name	Description	Default	Type
fileExist (producer)	What to do if a file already exists with the same name. Override, which is the default, replaces the existing file. - Append - adds content to the existing file. - Fail - throws a GenericFileOperationException, indicating that there is already an existing file. - Ignore - silently ignores the problem and does not override the existing file, but assumes everything is okay. - Move - option requires to use the moveExisting option to be configured as well. The option eagerDeleteTargetFile can be used to control what to do if an moving the file, and there exists already an existing file, otherwise causing the move operation to fail. The Move option will move any existing files, before writing the target file. - TryRename is only applicable if tempFileName option is in use. This allows to try renaming the file from the temporary name to the actual name, without doing any exists check. This check may be faster on some file systems and especially FTP servers. Enum values: • Override • Append • Fail • Ignore • Move • TryRename	Override	GenericFileExist
flatten (producer)	Flatten is used to flatten the file name path to strip any leading paths, so it's just the file name. This allows you to consume recursively into sub-directories, but when you eg write the files to another directory they will be written in a single directory. Setting this to true on the producer enforces that any file name in CamelFileName header will be stripped for any leading paths.	false	boolean
jailStartingDirectory (producer)	Used for jailing (restricting) writing files to the starting directory (and sub) only. This is enabled by default to not allow Camel to write files to outside directories (to be more secured out of the box). You can turn this off to allow writing files to directories outside the starting directory, such as parent or root folders.	true	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
moveExisting (producer)	Expression (such as File Language) used to compute file name to use when fileExist=Move is configured. To move files into a backup subdirectory just enter backup. This option only supports the following File Language tokens: file:name, file:name.ext, file:name.noext, file:onlyname, file:onlyname.noext, file:ext, and file:parent. Notice the file:parent is not supported by the FTP component, as the FTP component can only move any existing files to a relative directory based on current dir as base.		String
tempFileName (producer)	The same as tempPrefix option but offering a more fine grained control on the naming of the temporary filename as it uses the File Language. The location for tempFilename is relative to the final file location in the option 'fileName', not the target directory in the base uri. For example if option fileName includes a directory prefix: dir/finalFilename then tempFileName is relative to that subdirectory dir.		String
tempPrefix (producer)	This option is used to write the file using a temporary name and then, after the write is complete, rename it to the real name. Can be used to identify files being written and also avoid consumers (not using exclusive read locks) reading in progress files. Is often used by FTP when uploading big files.		String
allowNullBody (producer (advanced))	Used to specify if a null body is allowed during file writing. If set to true then an empty file will be created, when set to false, and attempting to send a null body to the file component, a GenericFileWriteException of 'Cannot write null body to file.' will be thrown. If the fileExist option is set to 'Override', then the file will be truncated, and if set to append the file will remain unchanged.	false	boolean
chmod (producer (advanced))	Allows you to set chmod on the stored file. For example chmod=640.		String

Name	Description	Default	Type
disconnectOnBatchComplete (producer (advanced))	Whether or not to disconnect from remote FTP server right after a Batch upload is complete. disconnectOnBatchComplete will only disconnect the current connection to the FTP server.	false	boolean
eagerDeleteTargetFile (producer (advanced))	Whether or not to eagerly delete any existing target file. This option only applies when you use fileExists=Override and the tempFileName option as well. You can use this to disable (set it to false) deleting the target file before the temp file is written. For example you may write big files and want the target file to exists during the temp file is being written. This ensure the target file is only deleted until the very last moment, just before the temp file is being renamed to the target filename. This option is also used to control whether to delete any existing files when fileExist=Move is enabled, and an existing file exists. If this option copyAndDeleteOnRenameFails false, then an exception will be thrown if an existing file existed, if its true, then the existing file is deleted before the move operation.	true	boolean
keepLastModified (producer (advanced))	Will keep the last modified timestamp from the source file (if any). Will use the Exchange.FILE_LAST_MODIFIED header to located the timestamp. This header can contain either a java.util.Date or long with the timestamp. If the timestamp exists and the option is enabled it will set this timestamp on the written file. Note: This option only applies to the file producer. You cannot use this option with any of the ftp producers.	false	boolean
moveExistingFileStrategy (producer (advanced))	Strategy (Custom Strategy) used to move file with special naming token to use when fileExist=Move is configured. By default, there is an implementation used if no custom strategy is provided.		FileMoveExistingStrategy
sendNoop (producer (advanced))	Whether to send a noop command as a pre-write check before uploading files to the FTP server. This is enabled by default as a validation of the connection is still valid, which allows to silently re-connect to be able to upload the file. However if this causes problems, you can turn this option off.	true	boolean

Name	Description	Default	Type
activePortRange (advanced)	Set the client side port range in active mode. The syntax is: minPort-maxPort Both port numbers are inclusive, eg 10000-19999 to include all 1xxxx ports.		String
autoCreate (advanced)	Automatically create missing directories in the file's pathname. For the file consumer, that means creating the starting directory. For the file producer, it means the directory the files should be written to.	true	boolean
bufferSize (advanced)	Buffer size in bytes used for writing files (or in case of FTP for downloading and uploading files).	131072	int
connectTimeout (advanced)	Sets the connect timeout for waiting for a connection to be established Used by both FTPClient and JSCH.	10000	int
ftpClient (advanced)	To use a custom instance of FTPClient.		FTPClient
ftpClientConfig (advanced)	To use a custom instance of FTPClientConfig to configure the FTP client the endpoint should use.		FTPClientConfig
ftpClientConfigParameters (advanced)	Used by FtpComponent to provide additional parameters for the FTPClientConfig.		Map
ftpClientParameters (advanced)	Used by FtpComponent to provide additional parameters for the FTPClient.		Map
maximumReconnectAttempts (advanced)	Specifies the maximum reconnect attempts Camel performs when it tries to connect to the remote FTP server. Use 0 to disable this behavior.		int
reconnectDelay (advanced)	Delay in millis Camel will wait before performing a reconnect attempt.	1000	long
siteCommand (advanced)	Sets optional site command(s) to be executed after successful login. Multiple site commands can be separated using a new line character.		String
soTimeout (advanced)	Sets the so timeout FTP and FTPS Is the SocketOptions.SO_TIMEOUT value in millis. Recommended option is to set this to 300000 so as not have a hanged connection. On SFTP this option is set as timeout on the JSCH Session instance.	300000	int

Name	Description	Default	Type
stepwise (advanced)	Sets whether we should stepwise change directories while traversing file structures when downloading files, or as well when uploading a file to a directory. You can disable this if you for example are in a situation where you cannot change directory on the FTP server due security reasons. Stepwise cannot be used together with streamDownload.	true	boolean
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
throwExceptionOnConnectFailed (advanced)	Should an exception be thrown if connection failed (exhausted)By default exception is not thrown and a WARN is logged. You can use this to enable exception being thrown and handle the thrown exception from the org.apache.camel.spi.PollingConsumerPollStrategy rollback method.	false	boolean
timeout (advanced)	Sets the data timeout for waiting for reply Used only by FTPClient.	30000	int
antExclude (filter)	Ant style filter exclusion. If both antInclude and antExclude are used, antExclude takes precedence over antInclude. Multiple exclusions may be specified in comma-delimited format.		String
antFilterCaseSensitive (filter)	Sets case sensitive flag on ant filter.	true	boolean
antInclude (filter)	Ant style filter inclusion. Multiple inclusions may be specified in comma-delimited format.		String
eagerMaxMessagesPerPoll (filter)	Allows for controlling whether the limit from maxMessagesPerPoll is eager or not. If eager then the limit is during the scanning of files. Where as false would scan all files, and then perform sorting. Setting this option to false allows for sorting all files first, and then limit the poll. Mind that this requires a higher memory usage as all file details are in memory to perform the sorting.	true	boolean

Name	Description	Default	Type
exclude (filter)	Is used to exclude files, if filename matches the regex pattern (matching is case in-sensitive). Notice if you use symbols such as plus sign and others you would need to configure this using the RAW() syntax if configuring this as an endpoint uri. See more details at configuring endpoint uris.		String
excludeExt (filter)	Is used to exclude files matching file extension name (case insensitive). For example to exclude bak files, then use excludeExt=bak. Multiple extensions can be separated by comma, for example to exclude bak and dat files, use excludeExt=bak,dat. Note that the file extension includes all parts, for example having a file named mydata.tar.gz will have extension as tar.gz. For more flexibility then use the include/exclude options.		String
filter (filter)	Pluggable filter as a org.apache.camel.component.file.GenericFileFilter class. Will skip files if filter returns false in its accept() method.		GenericFileFilter
filterDirectory (filter)	Filters the directory based on Simple language. For example to filter on current date, you can use a simple date pattern such as $\\{date:now:yyyMMdd}.		String
filterFile (filter)	Filters the file based on Simple language. For example to filter on file size, you can use $\\{file:size} 5000.		String
idempotent (filter)	Option to use the Idempotent Consumer EIP pattern to let Camel skip already processed files. Will by default use a memory based LRUCache that holds 1000 entries. If noop=true then idempotent will be enabled as well to avoid consuming the same files over and over again.	false	Boolean
idempotentKey (filter)	To use a custom idempotent key. By default the absolute path of the file is used. You can use the File Language, for example to use the file name and file size, you can do: idempotentKey=$\\{file:name}-$\\{file:size}.		String
idempotentRepository (filter)	A pluggable repository org.apache.camel.spi.IdempotentRepository which by default use MemoryIdempotentRepository if none is specified and idempotent is true.		IdempotentRepository

Name	Description	Default	Type
include (filter)	Is used to include files, if filename matches the regex pattern (matching is case in-sensitive). Notice if you use symbols such as plus sign and others you would need to configure this using the RAW() syntax if configuring this as an endpoint uri. See more details at configuring endpoint uris.		String
includeExt (filter)	Is used to include files matching file extension name (case insensitive). For example to include txt files, then use includeExt=txt. Multiple extensions can be separated by comma, for example to include txt and xml files, use includeExt=txt,xml. Note that the file extension includes all parts, for example having a file named mydata.tar.gz will have extension as tar.gz. For more flexibility then use the include/exclude options.		String
maxDepth (filter)	The maximum depth to traverse when recursively processing a directory.	2147483647	int
maxMessagesPerPoll (filter)	To define a maximum messages to gather per poll. By default no maximum is set. Can be used to set a limit of e.g. 1000 to avoid when starting up the server that there are thousands of files. Set a value of 0 or negative to disabled it. Notice: If this option is in use then the File and FTP components will limit before any sorting. For example if you have 100000 files and use maxMessagesPerPoll=500, then only the first 500 files will be picked up, and then sorted. You can use the eagerMaxMessagesPerPoll option and set this to false to allow to scan all files first and then sort afterwards.		int
minDepth (filter)	The minimum depth to start processing when recursively processing a directory. Using minDepth=1 means the base directory. Using minDepth=2 means the first sub directory.		int
move (filter)	Expression (such as Simple Language) used to dynamically set the filename when moving it after processing. To move files into a .done subdirectory just enter .done.		String
exclusiveReadLockStrategy (lock)	Pluggable read-lock as a org.apache.camel.component.file.GenericFileExclusiveReadLockStrategy implementation.		GenericFileExclusiveReadLockStrategy
readLock (lock)	Used by consumer, to only poll the files if it has	none	String

Name	Description	Default	Type
	exclusive read-lock on the file (i.e. the file is not in-progress or being written). Camel will wait until the file lock is granted. This option provides the build in strategies: - none - No read lock is in use - markerFile - Camel creates a marker file (fileName.camelLock) and then holds a lock on it. This option is not available for the FTP component - changed - Changed is using file length/modification timestamp to detect whether the file is currently being copied or not. Will at least use 1 sec to determine this, so this option cannot consume files as fast as the others, but can be more reliable as the JDK IO API cannot always determine whether a file is currently being used by another process. The option readLockCheckInterval can be used to set the check frequency. - fileLock - is for using java.nio.channels.FileLock. This option is not avail for Windows OS and the FTP component. This approach should be avoided when accessing a remote file system via a mount/share unless that file system supports distributed file locks. - rename - rename is for using a try to rename the file as a test if we can get exclusive read-lock. - idempotent - (only for file component) idempotent is for using a idempotentRepository as the read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that. - idempotent-changed - (only for file component) idempotent-changed is for using a idempotentRepository and changed as the combined read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that. - idempotent-rename - (only for file component) idempotent-rename is for using a idempotentRepository and rename as the combined read-lock. This allows to use read locks that supports clustering if the idempotent repository implementation supports that.Notice: The various read locks is not all suited to work in clustered mode, where concurrent consumers on different nodes is competing for the same files on a shared file system. The markerFile using a close to atomic operation to create the empty marker file, but its not guaranteed to work in a cluster. The fileLock may work better but then the file system need to support distributed file locks, and so on. Using the idempotent read lock can support clustering if the idempotent repository supports clustering, such as Hazelcast Component or Infinispan. Enum values: ● none ● markerFile ● fileLock		

Name	Description	Default	Type
	• rename • changed • idempotent • idempotent-changed • idempotent-rename		
readLockCheckInterval (lock)	Interval in millis for the read-lock, if supported by the read lock. This interval is used for sleeping between attempts to acquire the read lock. For example when using the changed read lock, you can set a higher interval period to cater for slow writes. The default of 1 sec. may be too fast if the producer is very slow writing the file. Notice: For FTP the default readLockCheckInterval is 5000. The readLockTimeout value must be higher than readLockCheckInterval, but a rule of thumb is to have a timeout that is at least 2 or more times higher than the readLockCheckInterval. This is needed to ensure that amble time is allowed for the read lock process to try to grab the lock before the timeout was hit.	1000	long
readLockDeleteOrphanLockFiles (lock)	Whether or not read lock with marker files should upon startup delete any orphan read lock files, which may have been left on the file system, if Camel was not properly shutdown (such as a JVM crash). If turning this option to false then any orphaned lock file will cause Camel to not attempt to pickup that file, this could also be due another node is concurrently reading files from the same shared directory.	true	boolean
readLockLoggingLevel (lock)	Logging level used when a read lock could not be acquired. By default a DEBUG is logged. You can change this level, for example to OFF to not have any logging. This option is only applicable for readLock of types: changed, fileLock, idempotent, idempotent-changed, idempotent-rename, rename. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	DEBUG	LoggingLevel

Name	Description	Default	Type
readLockMarkerFile (lock)	Whether to use marker file with the changed, rename, or exclusive read lock types. By default a marker file is used as well to guard against other processes picking up the same files. This behavior can be turned off by setting this option to false. For example if you do not want to write marker files to the file systems by the Camel application.	true	boolean
readLockMinAge (lock)	This option is applied only for readLock=changed. It allows to specify a minimum age the file must be before attempting to acquire the read lock. For example use readLockMinAge=300s to require the file is at last 5 minutes old. This can speedup the changed read lock as it will only attempt to acquire files which are at least that given age.	0	long
readLockMinLength (lock)	This option is applied only for readLock=changed. It allows you to configure a minimum file length. By default Camel expects the file to contain data, and thus the default value is 1. You can set this option to zero, to allow consuming zero-length files.	1	long
readLockRemoveOnCommit (lock)	This option is applied only for readLock=idempotent. It allows to specify whether to remove the file name entry from the idempotent repository when processing the file is succeeded and a commit happens. By default the file is not removed which ensures that any race-condition do not occur so another active node may attempt to grab the file. Instead the idempotent repository may support eviction strategies that you can configure to evict the file name entry after X minutes - this ensures no problems with race conditions. See more details at the readLockIdempotentReleaseDelay option.	false	boolean
readLockRemoveOnRollback (lock)	This option is applied only for readLock=idempotent. It allows to specify whether to remove the file name entry from the idempotent repository when processing the file failed and a rollback happens. If this option is false, then the file name entry is confirmed (as if the file did a commit).	true	boolean

Name	Description	Default	Type
readLockTimeout (lock)	Optional timeout in millis for the read-lock, if supported by the read-lock. If the read-lock could not be granted and the timeout triggered, then Camel will skip the file. At next poll Camel, will try the file again, and this time maybe the read-lock could be granted. Use a value of 0 or lower to indicate forever. Currently fileLock, changed and rename support the timeout. Notice: For FTP the default readLockTimeout value is 20000 instead of 10000. The readLockTimeout value must be higher than readLockCheckInterval, but a rule of thumb is to have a timeout that is at least 2 or more times higher than the readLockCheckInterval. This is needed to ensure that amble time is allowed for the read lock process to try to grab the lock before the timeout was hit.	10000	long
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
account (security)	Account to use for login.		String
password (security)	Password to use for login.		String
username (security)	Username to use for login.		String
shuffle (sort)	To shuffle the list of files (sort in random order).	false	boolean
sortBy (sort)	Built-in sort by using the File Language. Supports nested sorts, so you can have a sort by file name and as a 2nd group sort by modified date.		String
sorter (sort)	Pluggable sorter as a java.util.Comparator class.		Comparator

25.5. FTPS COMPONENT DEFAULT TRUST STORE

When using the **ftpClient.** properties related to SSL with the FTPS component, the trust store accept all certificates. If you only want trust selective certificates, you have to configure the trust store with the **ftpClient.trustStore.xxx** options or by configuring a custom **ftpClient**.

When using **sslContextParameters**, the trust store is managed by the configuration of the provided SSLContextParameters instance.

You can configure additional options on the **ftpClient** and **ftpClientConfig** from the URI directly by using the **ftpClient.** or **ftpClientConfig.** prefix.

For example to set the **setDataTimeout** on the **FTPClient** to 30 seconds you can do:

```
from("ftp://foo@myserver?password=secret&ftpClient.dataTimeout=30000").to("bean:foo");
```

You can mix and match and have use both prefixes, for example to configure date format or timezones.

```
from("ftp://foo@myserver?
password=secret&ftpClient.dataTimeout=30000&ftpClientConfig.serverLanguageCode=fr").to("bean:f
oo");
```

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

See the documentation of the Apache Commons FTP FTPClientConfig for possible options and more details. And as well for Apache Commons FTP FTPClient.

If you do not like having many and long configuration in the url you can refer to the **ftpClient** or **ftpClientConfig** to use by letting Camel lookup in the Registry for it.

For example:

```
<bean id="myConfig" class="org.apache.commons.net.ftp.FTPClientConfig">
    <property name="lenientFutureDates" value="true"/>
    <property name="serverLanguageCode" value="fr"/>
</bean>
```

And then let Camel lookup this bean when you use the # notation in the url.

```
from("ftp://foo@myserver?password=secret&ftpClientConfig=#myConfig").to("bean:foo");
```

25.6. EXAMPLES

```
ftp://someone@someftpserver.com/public/upload/images/holiday2008?password=secret&binary=true

ftp://someoneelse@someotherftpserver.co.uk:12049/reports/2008/password=secret&binary=false

ftp://publicftpserver.com/download
```

25.7. CONCURRENCY

FTP Consumer does not support concurrency

The FTP consumer (with the same endpoint) does not support concurrency (the backing FTP client is not thread safe).
You can use multiple FTP consumers to poll from different endpoints. It is only a single endpoint that does not support concurrent consumers.

The FTP producer does **not** have this issue, it supports concurrency.

25.8. MORE INFORMATION

This component is an extension of the File component. So there are more samples and details on the File component page.

25.9. DEFAULT WHEN CONSUMING FILES

The FTP consumer will by default leave the consumed files untouched on the remote FTP server. You have to configure it explicitly if you want it to delete the files or move them to another location. For example you can use **delete=true** to delete the files, or use **move=.done** to move the files into a hidden done sub directory.

The regular File consumer is different as it will by default move files to a **.camel** sub directory. The reason Camel does **not** do this by default for the FTP consumer is that it may lack permissions by default to be able to move or delete files.

25.9.1. limitations

The option **readLock** can be used to force Camel **not** to consume files that is currently in the progress of being written. However, this option is turned off by default, as it requires that the user has write access. See the options table at File2 for more details about read locks.

There are other solutions to avoid consuming files that are currently being written over FTP; for instance, you can write to a temporary destination and move the file after it has been written.

When moving files using **move** or **preMove** option the files are restricted to the FTP_ROOT folder. That prevents you from moving files outside the FTP area. If you want to move files to another area you can use soft links and move files into a soft linked folder.

25.10. MESSAGE HEADERS

The following message headers can be used to affect the behavior of the component

Header	Description
CamelFileName	Specifies the output file name (relative to the endpoint directory) to be used for the output message when sending to the endpoint. If this is not present and no expression either, then a generated message ID is used as the filename instead.
CamelFileNameProduced	The actual filepath (path + name) for the output file that was written. This header is set by Camel and its purpose is providing end-users the name of the file that was written.
CamelFileNameConsumed	The file name of the file consumed
CamelFileHost	The remote hostname.
CamelFileLocalWorkPath	Path to the local work file, if local work directory is used.

In addition the FTP/FTPS consumer and producer will enrich the Camel **Message** with the following headers

Header	Description
CamelFtpReplyCode	The FTP client reply code (the type is a integer)
CamelFtpReplyString	The FTP client reply string

25.10.1. Exchange Properties

Camel sets the following exchange properties

Header	Description
CamelBatchIndex	Current index out of total number of files being consumed in this batch.
CamelBatchSize	Total number of files being consumed in this batch.

Header	Description
CamelBatchComplete	True if there are no more files in this batch.

25.11. ABOUT TIMEOUTS

The two set of libraries (see top) has different API for setting timeout. You can use the **connectTimeout** option for both of them to set a timeout in millis to establish a network connection. An individual **soTimeout** can also be set on the FTP/FTPS, which corresponds to using **ftpClient.soTimeout**. Notice SFTP will automatically use **connectTimeout** as its **soTimeout**. The **timeout** option only applies for FTP/FTPS as the data timeout, which corresponds to the **ftpClient.dataTimeout** value. All timeout values are in millis.

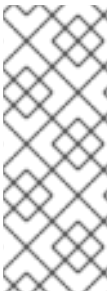
25.12. USING LOCAL WORK DIRECTORY

Camel supports consuming from remote FTP servers and downloading the files directly into a local work directory. This avoids reading the entire remote file content into memory as it is streamed directly into the local file using **FileOutputStream**.

Camel will store to a local file with the same name as the remote file, though with **.inprogress** as extension while the file is being downloaded. Afterwards, the file is renamed to remove the **.inprogress** suffix. And finally, when the Exchange is complete the local file is deleted.

So if you want to download files from a remote FTP server and store it as files then you need to route to a file endpoint such as:

```
from("ftp://someone@someserver.com?password=secret&localWorkDirectory=/tmp").to("file://inbox");
```



NOTE

The route above is ultra efficient as it avoids reading the entire file content into memory. It will download the remote file directly to a local file stream. The **java.io.File** handle is then used as the Exchange body. The file producer leverages this fact and can work directly on the work file **java.io.File** handle and perform a **java.io.File.rename** to the target filename. As Camel knows it's a local work file, it can optimize and use a rename instead of a file copy, as the work file is meant to be deleted anyway.

25.13. STEPWISE CHANGING DIRECTORIES

Camel FTP can operate in two modes in terms of traversing directories when consuming files (eg downloading) or producing files (eg uploading)

- stepwise
- not stepwise

You may want to pick either one depending on your situation and security issues. Some Camel end users can only download files if they use stepwise, while others can only download if they do not.

You can use the **stepwise** option to control the behavior.

Note that stepwise changing of directory will in most cases only work when the user is confined to it's home directory and when the home directory is reported as **"/"**.

The difference between the two of them is best illustrated with an example. Suppose we have the following directory structure on the remote FTP server we need to traverse and download files:

```
/
/one
/one/two
/one/two/sub-a
/one/two/sub-b
```

And that we have a file in each of sub-a (a.txt) and sub-b (b.txt) folder.

25.14. USING STEPWISE=TRUE (DEFAULT MODE)

```
TYPE A
200 Type set to A
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,17,94
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,95
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,96
200 Port command successful
LIST
150 Opening data channel for directory list.
226 Transfer OK
CDUP
200 CDUP successful. "/one/two" is current directory.
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
```

```
CWD two
250 CWD successful. "/one/two" is current directory.
PORT 127,0,0,1,17,97
200 Port command successful
RETR foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-a
250 CWD successful. "/one/two/sub-a" is current directory.
PORT 127,0,0,1,17,98
200 Port command successful
RETR a.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
PWD
257 "/" is current directory.
CWD one
250 CWD successful. "/one" is current directory.
CWD two
250 CWD successful. "/one/two" is current directory.
CWD sub-b
250 CWD successful. "/one/two/sub-b" is current directory.
PORT 127,0,0,1,17,99
200 Port command successful
RETR b.txt
150 Opening data channel for file transfer.
226 Transfer OK
CWD /
250 CWD successful. "/" is current directory.
QUIT
221 Goodbye
disconnected.
```

As you can see when stepwise is enabled, it will traverse the directory structure using CD xxx.

25.15. USING STEPWISE=FALSE

```
230 Logged on
TYPE A
200 Type set to A
SYST
215 UNIX emulated by FileZilla
PORT 127,0,0,1,4,122
200 Port command successful
LIST one/two
```

```
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,123
200 Port command successful
LIST one/two/sub-a
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,124
200 Port command successful
LIST one/two/sub-b
150 Opening data channel for directory list
226 Transfer OK
PORT 127,0,0,1,4,125
200 Port command successful
RETR one/two/foo.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,126
200 Port command successful
RETR one/two/sub-a/a.txt
150 Opening data channel for file transfer.
226 Transfer OK
PORT 127,0,0,1,4,127
200 Port command successful
RETR one/two/sub-b/b.txt
150 Opening data channel for file transfer.
226 Transfer OK
QUIT
221 Goodbye
disconnected.
```

As you can see when not using stepwise, there are no CD operation invoked at all.

25.16. SAMPLES

In the sample below we set up Camel to download all the reports from the FTP server once every hour (60 min) as BINARY content and store it as files on the local file system.

And the route using XML DSL:

```
  <route>
    <from uri="ftp://scott@localhost/public/reports?
password=tiger&amp;binary=true&amp;delay=60000"/>
    <to uri="file://target/test-reports"/>
  </route>
```

25.16.1. Consuming a remote FTPS server (implicit SSL) and client authentication

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&securityProtocol=SSL&implicit=true
     &ftpClient.keyStore.file=./src/test/resources/server.jks
     &ftpClient.keyStore.password=password&ftpClient.keyStore.keyPassword=password")
  .to("bean:foo");
```

25.16.2. Consuming a remote FTPS server (explicit TLS) and a custom trust store configuration

```
from("ftps://admin@localhost:2222/public/camel?
password=admin&ftpClient.trustStore.file=./src/test/resources/server.jks&ftpClient.trustStore.password=
password")
  .to("bean:foo");
```

25.17. CUSTOM FILTERING

Camel supports pluggable filtering strategies. This strategy it to use the build in **org.apache.camel.component.file.GenericFileFilter** in Java. You can then configure the endpoint with such a filter to skip certain filters before being processed.

In the sample we have built our own filter that only accepts files starting with report in the filename.

And then we can configure our route using the **filter** attribute to reference our filter (using # notation) that we have defined in the spring XML file:

```
 <!-- define our sorter as a plain spring bean -->
 <bean id="myFilter" class="com.mycompany.MyFileFilter"/>

<route>
  <from uri="ftp://someuser@someftpserver.com?password=secret&amp;filter=#myFilter"/>
  <to uri="bean:processInbox"/>
</route>
```

25.18. FILTERING USING ANT PATH MATCHER

The ANT path matcher is a filter that is shipped out-of-the-box in the **camel-spring** jar. So you need to depend on **camel-spring** if you are using Maven.
The reason is that we leverage Spring's AntPathMatcher to do the actual matching.

The file paths are matched with the following rules:

- **?** matches one character
- ***** matches zero or more characters
- ****** matches zero or more directories in a path

The sample below demonstrates how to use it:

25.19. USING A PROXY WITH SFTP

To use an HTTP proxy to connect to your remote host, you can configure your route in the following way:

```
<!-- define our sorter as a plain spring bean -->
<bean id="proxy" class="com.jcraft.jsch.ProxyHTTP">
  <constructor-arg value="localhost"/>
  <constructor-arg value="7777"/>
</bean>
```

```
<route>
  <from uri="sftp://localhost:9999/root?username=admin&password=admin&proxy=#proxy"/>
  <to uri="bean:processFile"/>
</route>
```

You can also assign a user name and password to the proxy, if necessary. Please consult the documentation for **com.jcraft.jsch.Proxy** to discover all options.

25.20. SETTING PREFERRED SFTP AUTHENTICATION METHOD

If you want to explicitly specify the list of authentication methods that should be used by **sftp** component, use **preferredAuthentications** option. If for example you would like Camel to attempt to authenticate with private/public SSH key and fallback to user/password authentication in the case when no public key is available, use the following route configuration:

```
from("sftp://localhost:9999/root?
username=admin&password=admin&preferredAuthentications=publickey,password").
  to("bean:processFile");
```

25.21. CONSUMING A SINGLE FILE USING A FIXED NAME

When you want to download a single file and knows the file name, you can use **fileName=myFileName.txt** to tell Camel the name of the file to download. By default the consumer will still do a FTP LIST command to do a directory listing and then filter these files based on the **fileName** option. Though in this use-case it may be desirable to turn off the directory listing by setting **useList=false**. For example the user account used to login to the FTP server may not have permission to do a FTP LIST command. So you can turn off this with **useList=false**, and then provide the fixed name of the file to download with **fileName=myFileName.txt**, then the FTP consumer can still download the file. If the file for some reason does not exist, then Camel will by default throw an exception, you can turn this off and ignore this by setting **ignoreFileNotFoundOrPermissionError=true**.

For example to have a Camel route that pickup a single file, and delete it after use you can do

```
from("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true&fileName
=report.txt&delete=true")
  .to("activemq:queue:report");
```

Notice that we have used all the options we talked above.

You can also use this with **ConsumerTemplate**. For example to download a single file (if it exists) and grab the file content as a String type:

```
String data = template.retrieveBodyNoWait("ftp://admin@localhost:21/nolist/?
password=admin&stepwise=false&useList=false&ignoreFileNotFoundOrPermissionError=true&fileName
=report.txt&delete=true", String.class);
```

25.22. DEBUG LOGGING

This component has log level **TRACE** that can be helpful if you have problems.

25.23. SPRING BOOT AUTO-CONFIGURATION

When using ftp with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-ftp-starter</artifactId>
</dependency>
```

The component supports 13 options, which are listed below.

Name	Description	Default	Type
camel.component.ftp.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.ftp.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.ftp.enabled	Whether to enable auto configuration of the ftp component. This is enabled by default.		Boolean
camel.component.ftp.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.ftps.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.ftps.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.ftps.enabled	Whether to enable auto configuration of the ftps component. This is enabled by default.		Boolean
camel.component.ftps.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.ftps.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.sftp.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.sftp.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.sftp.enabled	Whether to enable auto configuration of the sftp component. This is enabled by default.		Boolean
camel.component.sftp.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 26. GOOGLE BIGQUERY

Since Camel 2.20

Only producer is supported.

The Google Bigquery component provides access to the Cloud BigQuery Infrastructure via the link:https://developers.google.com/api-client-library/java/apis/bigquery/v2 [Google Client Services API].

The current implementation does not use gRPC.

The current implementation does not support querying BigQuery, it is only a producer.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-bigquery</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

26.1. AUTHENTICATION CONFIGURATION

Google BigQuery component authentication is targeted for use with the GCP Service Accounts. For more information please refer to Google Cloud Platform Auth Guide .

Google security credentials can be set explicitly by providing the path to the GCP credentials file location.

Or they are set implicitly, where the connection factory falls back on Application Default Credentials.

When you have the **service account key** you can provide authentication credentials to your application code. Google security credentials can be set through the component endpoint:

```
String endpoint = "google-bigquery://project-id:datasetId[:tableId]?
serviceAccountKey=/home/user/Downloads/my-key.json";
```

You can also use the base64 encoded content of the authentication credentials file if you don't want to set a file system path.

```
String endpoint = "google-bigquery://project-id:datasetId[:tableId]?serviceAccountKey=base64:
<base64 encoded>";
```

Or by setting the environment variable **GOOGLE_APPLICATION_CREDENTIALS** :

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/user/Downloads/my-key.json"
```

26.2. URI FORMAT

```
google-bigquery://project-id:datasetId[:tableId]?[options]
```

26.3. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

26.3.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

26.3.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

26.4. COMPONENT OPTIONS

The Google BigQuery component supports 5 options, which are listed below.

Name	Description	Default	Type
connectionFactory (producer)	**Autowired** ConnectionFactory to obtain connection to Bigquery Service. If not provided the default one will be used.		GoogleBigQueryConnectionFactory
datasetId (producer)	BigQuery Dataset Id.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
projectId (producer)	Google Cloud Project Id.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

26.5. ENDPOINT OPTIONS

The Google BigQuery endpoint is configured using URI syntax:

```
google-bigquery:projectId:datasetId:tableId
```

with the following path and query parameters:

26.5.1. Path Parameters (3 parameters)

Name	Description	Default	Type
projectId (common)	**Required** Google Cloud Project Id.		String
datasetId (common)	**Required** BigQuery Dataset Id.		String
tableId (common)	BigQuery table id.		String

26.5.2. Query Parameters (4 parameters)

Name	Description	Default	Type
connectionFactory (producer)	**Autowired** ConnectionFactory to obtain connection to Bigquery Service. If not provided the default one will be used.		GoogleBigQueryConnectionFactory
useAsInsertId (producer)	Field name to use as insert id.		String
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
serviceAccountKey (security)	Service account key in json format to authenticate an application as a service account to google cloud platform.		String

26.6. MESSAGE HEADERS

The Google BigQuery component supports 4 message header(s), which is/are listed below:

Name	Description	Default	Type
CamelGoogleBigQueryTableSuffix (producer) Constant: TABLE_SUFFIX	Table suffix to use when inserting data.		String
CamelGoogleBigQueryTableId (producer) Constant: TABLE_ID	Table id where data will be submitted. If specified will override endpoint configuration.		String

Name	Description	Default	Type
CamelGoogleBigQueryInsertId (producer) Constant: INSERT_ID	InsertId to use when inserting data.		String
CamelGoogleBigQueryPartitionDecorator (producer) Constant: PARTITION_DECORATOR	Partition decorator to indicate partition to use when inserting data.		String

26.7. PRODUCER ENDPOINTS

Producer endpoints can accept and deliver to BigQuery individual and grouped exchanges alike. Grouped exchanges have **Exchange.GROUPED_EXCHANGE** property set.

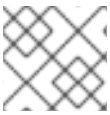
Google BigQuery producer will send a grouped exchange in a single api call unless different table suffix or partition decorators are specified in which case it will break it down to ensure data is written with the correct suffix or partition decorator.

Google BigQuery endpoint expects the payload to be either a map or list of maps. A payload containing a map will insert a single row and a payload containing a list of map's will insert a row for each entry in the list.

26.8. TEMPLATE TABLES

Templated tables can be specified using the **GoogleBigQueryConstants.TABLE_SUFFIX** header. For example, the following route will create tables and insert records sharded on a per day basis:

```
from("direct:start")
  .header(GoogleBigQueryConstants.TABLE_SUFFIX, "_${date:now:yyyyMMdd}")
  .to("google-bigquery:sampleDataset:sampleTable")
```



NOTE

It is recommended to use partitioning for this use case.

For more information about Template table, see Template Tables.

26.9. PARTITIONING

Partitioning is specified when creating a table and if set data will be automatically partitioned into separate tables. When inserting data a specific partition can be specified by setting the **GoogleBigQueryConstants.PARTITION_DECORATOR** header on the exchange.

For more information about partitioning, see Creating partitioned tables.

26.10. ENSURING DATA CONSISTENCY

An insert id can be set on the exchange with the header **GoogleBigQueryConstants.INSERT_ID** or by specifying query parameter **useAsInsertId**. As an insert id need to be specified per row, the inserted the exchange header can't be used when the payload is a list. If the payload is a list then the **GoogleBigQueryConstants.INSERT_ID** will be ignored. In that case use the query parameter **useAsInsertId**.

For more information, see Data consistency

26.11. SPRING BOOT AUTO-CONFIGURATION

When using google-bigquery with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-google-bigquery-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.component.google-bigquery-sql.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.google-bigquery-sql.connection-factory	ConnectionFactory to obtain connection to Bigquery Service. If not provided the default one will be used. The option is a org.apache.camel.component.google.bigquery.GoogleBigQueryConnectionFactory type.		GoogleBigQueryConnectionFactory
camel.component.google-bigquery-sql.enabled	Whether to enable auto configuration of the google-bigquery-sql component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.google-bigquery-sql.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.google-bigquery-sql.project-id	Google Cloud Project Id.		String
camel.component.google-bigquery.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.google-bigquery.connection-factory	ConnectionFactory to obtain connection to Bigquery Service. If not provided the default one will be used. The option is a org.apache.camel.component.google.bigquery.GoogleBigQueryConnectionFactory type.		GoogleBigQueryConnectionFactory
camel.component.google-bigquery.dataset-id	BigQuery Dataset Id.		String
camel.component.google-bigquery.enabled	Whether to enable auto configuration of the google-bigquery component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.google-bigquery.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.google-bigquery.project-id	Google Cloud Project Id.		String

CHAPTER 27. GOOGLE PUBSUB

Since Camel 2.19

Both producer and consumer are supported.

The Google Pubsub component provides access to the Cloud Pub/Sub Infrastructure via the Google Cloud Java Client for Google Cloud Pub/Sub.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-google-pubsub</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

27.1. URI FORMAT

The Google Pubsub Component uses the following URI format:

```
google-pubsub://project-id:destinationName?[options]
```

Destination Name can be either a topic or a subscription name.

27.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

27.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

27.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

27.3. COMPONENT OPTIONS

The Google Pubsub component supports 10 options, which are listed below.

Name	Description	Default	Type
authenticate (common)	Use Credentials when interacting with PubSub service (no authentication is required when using emulator).	true	boolean
endpoint (common)	Endpoint to use with local Pub/Sub emulator.		String
serviceAccountKey (common)	The Service account key that can be used as credentials for the PubSub publisher/subscriber. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
synchronousPullRetryableCodes (consumer)	Comma-separated list of additional retryable error codes for synchronous pull. By default the PubSub client library retries ABORTED, UNAVAILABLE, UNKNOWN.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
publisherCacheSize (producer)	Maximum number of producers to cache. This could be increased if you have producers for lots of different topics.		int
publisherCacheTimeout (producer)	How many milliseconds should each producer stay alive in the cache.		int
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
publisherTerminationTimeout (advanced)	How many milliseconds should a producer be allowed to terminate.		int

27.4. ENDPOINT OPTIONS

The Google Pubsub endpoint is configured using URI syntax:

```
google-pubsub:projectId:destinationName
```

with the following path and query parameters:

27.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
projectId (common)	**Required** The Google Cloud PubSub Project Id.		String
destinationName (common)	**Required** The Destination Name. For the consumer this will be the subscription name, while for the producer this will be the topic name.		String

27.4.2. Query Parameters (15 parameters)

Name	Description	Default	Type
authenticate (common)	Use Credentials when interacting with PubSub service (no authentication is required when using emulator).	true	boolean
loggerId (common)	Logger ID to use when a match to the parent route required.		String
serviceAccountKey (common)	The Service account key that can be used as credentials for the PubSub publisher/subscriber. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
ackMode (consumer)	AUTO = exchange gets ack'ed/nack'ed on completion. NONE = downstream process has to ack/nack explicitly. Enum values: • AUTO • NONE	AUTO	AckMode
concurrentConsumers (consumer)	The number of parallel streams consuming from the subscription.	1	Integer
maxAckExtensionPeriod (consumer)	Set the maximum period a message ack deadline will be extended. Value in seconds.	3600	int
maxMessagesPerPoll (consumer)	The max number of messages to receive from the server in a single API call.	1	Integer
synchronousPull (consumer)	Synchronously pull batches of messages.	false	boolean

Name	Description	Default	Type
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
messageOrderingEnabled (producer (advanced))	Should message ordering be enabled.	false	boolean
pubsubEndpoint (producer (advanced))	Pub/Sub endpoint to use. Required when using message ordering, and ensures that messages are received in order even when multiple publishers are used.		String

Name	Description	Default	Type
serializer (producer (advanced))	**Autowired** A custom GooglePubsubSerializer to use for serializing message payloads in the producer.		GooglePubsubSerializer

27.5. MESSAGE HEADERS

The Google Pubsub component supports 5 message header(s), which are listed below:

Name	Description	Default	Type
CamelGooglePubsubMessageId (common) Constant: MESSAGE_ID	The ID of the message, assigned by the server when the message is published.		String
CamelGooglePubsubMsgAckId (consumer) Constant: ACK_ID	The ID used to acknowledge the received message.		String
CamelGooglePubsubPublishTime (consumer) Constant: PUBLISH_TIME	The time at which the message was published.		Timestamp
CamelGooglePubsubAttributes (common) Constant: ATTRIBUTES	The attributes of the message.		Map
CamelGooglePubsubOrderingKey (producer) Constant: ORDERING_KEY	If non-empty, identifies related messages for which publish order should be respected.		String

27.6. PRODUCER ENDPOINTS

Producer endpoints can accept and deliver to PubSub individual and grouped exchanges alike. Grouped exchanges have **Exchange.GROUPED_EXCHANGE** property set.

Google PubSub expects the payload to be byte[] array, Producer endpoints will send:

- String body as byte[] encoded as UTF-8
- byte[] body as is
- Everything else will be serialised into byte[] array

A Map set as message header **GooglePubsubConstants.ATTRIBUTES** will be sent as PubSub attributes.

Google PubSub supports ordered message delivery.

To enable this set set the options messageOrderingEnabled to true, and the pubsubEndpoint to a GCP region.

When producing messages set the message header **GooglePubsubConstants.ORDERING_KEY** . This will be set as the PubSub orderingKey for the message.

More information in Ordering messages.

Once exchange has been delivered to PubSub the PubSub Message ID will be assigned to the header **GooglePubsubConstants.MESSAGE_ID**.

27.7. CONSUMER ENDPOINTS

Google PubSub will redeliver the message if it has not been acknowledged within the time period set as a configuration option on the subscription.

The component will acknowledge the message once exchange processing has been completed.

If the route throws an exception, the exchange is marked as failed and the component will NACK the message - it will be redelivered immediately.

To ack/nack the message the component uses Acknowledgement ID stored as header **GooglePubsubConstants.ACK_ID**. If the header is removed or tampered with, the ack will fail and the message will be redelivered again after the ack deadline.

27.8. MESSAGE BODY

The consumer endpoint returns the content of the message as byte[] - exactly as the underlying system sends it. It is up for the route to convert/unmarshall the contents.

27.9. AUTHENTICATION CONFIGURATION

By default this component aquires credentials using **GoogleCredentials.getApplicationDefault()**. This behavior can be disabled by setting *authenticate* option to **false**, in which case requests to Google API will be made without authentication details. This is only desirable when developing against an emulator. This behavior can be altered by supplying a path to a service account key file.

27.10. ROLLBACK AND REDELIVERY

The rollback for Google PubSub relies on the idea of the Acknowledgement Deadline - the time period where Google PubSub expects to receive the acknowledgement. If the acknowledgement has not been received, the message is redelivered.

Google provides an API to extend the deadline for a message.

More information in Google PubSub Documentation .

So, rollback is effectively a deadline extension API call with zero value - i.e. deadline is reached now and message can be redelivered to the next consumer.

It is possible to delay the message redelivery by setting the acknowledgement deadline explicitly for the rollback by setting the message header **GooglePubsubConstants.ACK_DEADLINE** to the value in seconds.

27.11. SPRING BOOT AUTO-CONFIGURATION

When using google-pubsub with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-google-pubsub-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.component.google-pubsub.authenticate	Use Credentials when interacting with PubSub service (no authentication is required when using emulator).	true	Boolean
camel.component.google-pubsub.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.google-pubsub.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.google-pubsub.enabled	Whether to enable auto configuration of the google-pubsub component. This is enabled by default.		Boolean
camel.component.google-pubsub.endpoint	Endpoint to use with local Pub/Sub emulator.		String
camel.component.google-pubsub.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.google-pubsub.publisher-cache-size	Maximum number of producers to cache. This could be increased if you have producers for lots of different topics.		Integer
camel.component.google-pubsub.publisher-cache-timeout	How many milliseconds should each producer stay alive in the cache.		Integer
camel.component.google-pubsub.publisher-termination-timeout	How many milliseconds should a producer be allowed to terminate.		Integer
camel.component.google-pubsub.service-account-key	The Service account key that can be used as credentials for the PubSub publisher/subscriber. It can be loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
camel.component.google-pubsub.synchronous-pull-retryable-codes	Comma-separated list of additional retryable error codes for synchronous pull. By default the PubSub client library retries ABORTED, UNAVAILABLE, UNKNOWN.		String

CHAPTER 28. HTTP

Only producer is supported

The HTTP component provides HTTP based endpoints for calling external HTTP resources (as a client to call external servers using HTTP).

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-http</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

28.1. URI FORMAT

```
http:hostname[:port][/resourceUri][?options]
```

Will by default use port 80 for HTTP and 443 for HTTPS.

28.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

28.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

28.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

28.3. COMPONENT OPTIONS

The HTTP component supports 37 options, which are listed below.

Name	Description	Default	Type
cookieStore (producer)	To use a custom org.apache.http.client.CookieStore. By default the org.apache.http.impl.client.BasicCookieStore is used which is an in-memory only cookie store. Notice if bridgeEndpoint=true then the cookie store is forced to be a noop cookie store as cookie shouldn't be stored as we are just bridging (eg acting as a proxy).		CookieStore
copyHeaders (producer)	If this option is true then IN exchange headers will be copied to OUT exchange headers according to copy strategy. Setting this to false, allows to only include the headers from the HTTP response (not propagating IN headers).	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
responsePayloadStreamingThreshold (producer)	This threshold in bytes controls whether the response payload should be stored in memory as a byte array or be streaming based. Set this to -1 to always use streaming mode.	8192	int
skipRequestHeaders (producer (advanced))	Whether to skip mapping all the Camel headers as HTTP request headers. If there are no data from Camel headers needed to be included in the HTTP request then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	boolean

Name	Description	Default	Type
skipResponseHeaders (producer (advanced))	Whether to skip mapping all the HTTP response headers to Camel headers. If there are no data needed from HTTP headers then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	boolean
allowJavaSerializedObject (advanced)	Whether to allow java serialization when a request uses context-type=application/x-java-serialized-object. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	boolean
authCachingDisabled (advanced)	Disables authentication scheme caching.	false	boolean
automaticRetriesDisabled (advanced)	Disables automatic request recovery and re-execution.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
clientConnectionManager (advanced)	To use a custom and shared HttpClientConnectionManager to manage connections. If this has been configured then this is always used for all endpoints created by this component.		HttpClientConnectionManager
connectionsPerRoute (advanced)	The maximum number of connections per route.	20	int
connectionStateDisabled (advanced)	Disables connection state tracking.	false	boolean
connectionTimeToLive (advanced)	The time for connection to live, the time unit is millisecond, the default value is always keep alive.		long
contentCompressionDisabled (advanced)	Disables automatic content decompression.	false	boolean

Name	Description	Default	Type
cookieManagementDisabled (advanced)	Disables state (cookie) management.	false	boolean
defaultUserAgentDisabled (advanced)	Disables the default user agent set by this builder if none has been provided by the user.	false	boolean
httpBinding (advanced)	To use a custom HttpBinding to control the mapping between Camel message and HttpClient.		HttpBinding
httpClientConfigurer (advanced)	To use the custom HttpClientConfigurer to perform configuration of the HttpClient that will be used.		HttpClientConfigurer
httpConfiguration (advanced)	To use the shared HttpConfiguration as base configuration.		HttpConfiguration
httpContext (advanced)	To use a custom org.apache.http.protocol.HttpContext when executing requests.		HttpContext
maxTotalConnections (advanced)	The maximum number of connections.	200	int
redirectHandlingDisabled (advanced)	Disables automatic redirect handling.	false	boolean
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
proxyAuthDomain (proxy)	Proxy authentication domain to use.		String
proxyAuthHost (proxy)	Proxy authentication host.		String
proxyAuthMethod (proxy)	Proxy authentication method to use. Enum values: • Basic • Digest • NTLM		String

Name	Description	Default	Type
proxyAuthNtHost (proxy)	Proxy authentication domain (workstation name) to use with NTML.		String
proxyAuthPassword (proxy)	Proxy authentication password.		String
proxyAuthPort (proxy)	Proxy authentication port.		Integer
proxyAuthUsername (proxy)	Proxy authentication username.		String
sslContextParameters (security)	To configure security using SSLContextParameters. Important: Only one instance of org.apache.camel.support.jsse.SSLContextParameters is supported per HttpComponent. If you need to use 2 or more different instances, you need to define a new HttpComponent per instance you need.		SSLContextParameters
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean
x509HostnameVerifier (security)	To use a custom X509HostnameVerifier such as DefaultHostnameVerifier or NoopHostnameVerifier.		HostnameVerifier
connectionRequestTimeout (timeout)	The timeout in milliseconds used when requesting a connection from the connection manager. A timeout value of zero is interpreted as an infinite timeout. A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	int
connectTimeout (timeout)	Determines the timeout in milliseconds until a connection is established. A timeout value of zero is interpreted as an infinite timeout. A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	int
socketTimeout (timeout)	Defines the socket timeout in milliseconds, which is the timeout for waiting for data or, put differently, a maximum period inactivity between two consecutive data packets). A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	int

28.4. ENDPOINT OPTIONS

The HTTP endpoint is configured using URI syntax:

```
http://httpUri
```

with the following path and query parameters:

28.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
httpUri (common)	**Required** The url of the HTTP endpoint to call.		URI

28.4.2. Query Parameters (51 parameters)

Name	Description	Default	Type
chunked (producer)	If this option is false the Servlet will disable the HTTP streaming and set the content-length header on the response.	true	boolean
disableStreamCache (common)	Determines whether or not the raw input stream from Servlet is cached or not (Camel will read the stream into a in memory/overflow to file, Stream caching) cache. By default Camel will cache the Servlet input stream to support reading it multiple times to ensure it Camel can retrieve all data from the stream. However you can set this option to true when you for example need to access the raw stream, such as streaming it directly to a file or other persistent store. DefaultHttpBinding will copy the request input stream into a stream cache and put it into message body if this option is false to support reading the stream multiple times. If you use Servlet to bridge/proxy an endpoint then consider enabling this option to improve performance, in case you do not need to read the message payload multiple times. The http producer will by default cache the response body stream. If setting this option to true, then the producers will not cache the response body stream but use the response stream as-is as the message body.	false	boolean
headerFilterStrategy (common)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
httpBinding (common (advanced))	To use a custom HttpBinding to control the mapping between Camel message and HttpClient.		HttpBinding

Name	Description	Default	Type
bridgeEndpoint (producer)	If the option is true, HttpProducer will ignore the Exchange.HTTP_URI header, and use the endpoint's URI for request. You may also set the option throwExceptionOnFailure to be false to let the HttpProducer send all the fault response back.	false	boolean
clearExpiredCookies (producer)	Whether to clear expired cookies before sending the HTTP request. This ensures the cookies store does not keep growing by adding new cookies which is newer removed when they are expired. If the component has disabled cookie management then this option is disabled too.	true	boolean
connectionClose (producer)	Specifies whether a Connection Close header must be added to HTTP Request. By default connectionClose is false.	false	boolean
copyHeaders (producer)	If this option is true then IN exchange headers will be copied to OUT exchange headers according to copy strategy. Setting this to false, allows to only include the headers from the HTTP response (not propagating IN headers).	true	boolean
customHostHeader (producer)	To use custom host header for producer. When not set in query will be ignored. When set will override host header derived from url.		String
httpMethod (producer)	Configure the HTTP method to use. The HttpMethod header cannot override this option if set. Enum values: • GET • POST • PUT • DELETE • HEAD • OPTIONS • TRACE • PATCH		HttpMethods
ignoreResponseBody (producer)	If this option is true, The http producer won't read response body and cache the input stream.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
preserveHostHeader (producer)	If the option is true, HttpProducer will set the Host header to the value contained in the current exchange Host header, useful in reverse proxy applications where you want the Host header received by the downstream server to reflect the URL called by the upstream client, this allows applications which use the Host header to generate accurate URL's for a proxied service.	false	boolean
throwExceptionOnFailure (producer)	Option to disable throwing the HttpOperationFailedException in case of failed responses from the remote server. This allows you to get all responses regardless of the HTTP status code.	true	boolean
transferException (producer)	If enabled and an Exchange failed processing on the consumer side, and if the caused Exception was send back serialized in the response as a application/x-java-serialized-object content type. On the producer side the exception will be deserialized and thrown as is, instead of the HttpOperationFailedException. The caused exception is required to be serialized. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	boolean
cookieHandler (producer (advanced))	Configure a cookie handler to maintain a HTTP session.		CookieHandler

Name	Description	Default	Type
cookieStore (producer (advanced))	To use a custom CookieStore. By default the BasicCookieStore is used which is an in-memory only cookie store. Notice if bridgeEndpoint=true then the cookie store is forced to be a noop cookie store as cookie shouldn't be stored as we are just bridging (eg acting as a proxy). If a cookieHandler is set then the cookie store is also forced to be a noop cookie store as cookie handling is then performed by the cookieHandler.		CookieStore
deleteWithBody (producer (advanced))	Whether the HTTP DELETE should include the message body or not. By default HTTP DELETE do not include any HTTP body. However in some rare cases users may need to be able to include the message body.	false	boolean
getWithBody (producer (advanced))	Whether the HTTP GET should include the message body or not. By default HTTP GET do not include any HTTP body. However in some rare cases users may need to be able to include the message body.	false	boolean
okStatusCodeRange (producer (advanced))	The status codes which are considered a success response. The values are inclusive. Multiple ranges can be defined, separated by comma, e.g. 200-204,209,301-304. Each range must be a single number or from-to with the dash included.	200-299	String
skipRequestHeaders (producer (advanced))	Whether to skip mapping all the Camel headers as HTTP request headers. If there are no data from Camel headers needed to be included in the HTTP request then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	boolean
skipResponseHeaders (producer (advanced))	Whether to skip mapping all the HTTP response headers to Camel headers. If there are no data needed from HTTP headers then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	boolean
userAgent (producer (advanced))	To set a custom HTTP User-Agent request header.		String
clientBuilder (advanced)	Provide access to the http client request parameters used on new RequestConfig instances used by producers or consumers of this endpoint.		HttpClientBuilder

Name	Description	Default	Type
clientConnectionManager (advanced)	To use a custom HttpClientConnectionManager to manage connections.		HttpClientConnectionManager
connectionsPerRoute (advanced)	The maximum number of connections per route.	20	int
httpClient (advanced)	Sets a custom HttpClient to be used by the producer.		HttpClient
httpClientConfigurer (advanced)	Register a custom configuration strategy for new HttpClient instances created by producers or consumers such as to configure authentication mechanisms etc.		HttpClientConfigurer
httpClientOptions (advanced)	To configure the HttpClient using the key/values from the Map.		Map
httpContext (advanced)	To use a custom HttpContext instance.		HttpContext
maxTotalConnections (advanced)	The maximum number of connections.	200	int
useSystemProperties (advanced)	To use System Properties as fallback for configuration.	false	boolean
proxyAuthDomain (proxy)	Proxy authentication domain to use with NTML.		String
proxyAuthHost (proxy)	Proxy authentication host.		String
proxyAuthMethod (proxy)	Proxy authentication method to use. Enum values: • Basic • Digest • NTLM		String
proxyAuthNtHost (proxy)	Proxy authentication domain (workstation name) to use with NTML.		String

Name	Description	Default	Type
proxyAuthPassword (proxy)	Proxy authentication password.		String
proxyAuthPort (proxy)	Proxy authentication port.		int
proxyAuthScheme (proxy)	Proxy authentication scheme to use. Enum values: • http • https		String
proxyAuthUsername (proxy)	Proxy authentication username.		String
proxyHost (proxy)	Proxy hostname to use.		String
proxyPort (proxy)	Proxy port to use.		int
authDomain (security)	Authentication domain to use with NTML.		String
authenticationPreemptive (security)	If this option is true, camel-http sends preemptive basic authentication to the server.	false	boolean
authHost (security)	Authentication host to use with NTML.		String
authMethod (security)	Authentication methods allowed to use as a comma separated list of values Basic, Digest or NTLM.		String
authMethodPriority (security)	Which authentication method to prioritize to use, either as Basic, Digest or NTLM. Enum values: • Basic • Digest • NTLM		String
authPassword (security)	Authentication password.		String

Name	Description	Default	Type
authUsername (security)	Authentication username.		String
sslContextParameters (security)	To configure security using SSLContextParameters. Important: Only one instance of org.apache.camel.util.jsse.SSLContextParameters is supported per HttpComponent. If you need to use 2 or more different instances, you need to define a new HttpComponent per instance you need.		SSLContextParameters
x509HostnameVerifier (security)	To use a custom X509HostnameVerifier such as DefaultHostnameVerifier or NoopHostnameVerifier.		HostnameVerifier

28.5. MESSAGE HEADERS

Name	Type	Description
Exchange.HTTP_URI	**String**	URI to call. Will override existing URI set directly on the endpoint. This uri is the uri of the http server to call. Its not the same as the Camel endpoint uri, where you can configure endpoint options such as security etc. This header does not support that, its only the uri of the http server.
Exchange.HTTP_PATH	**String**	Request URI's path, the header will be used to build the request URI with the HTTP_URI.
Exchange.HTTP_QUERY	**String**	URI parameters. Will override existing URI parameters set directly on the endpoint.
Exchange.HTTP_RESPONSE_CODE	**int**	The HTTP response code from the external server. Is 200 for OK.
Exchange.HTTP_RESPONSE_TEXT	**String**	The HTTP response text from the external server.
Exchange.HTTP_CHARACTER_ENCODING	**String**	Character encoding.
Exchange.CONTENT_TYPE	**String**	The HTTP content type. Is set on both the IN and OUT message to provide a content type, such as **text/html**.
Exchange.CONTENT_ENCODING	**String**	The HTTP content encoding. Is set on both the IN and OUT message to provide a content encoding, such as **gzip**.

28.6. MESSAGE BODY

Camel will store the HTTP response from the external server on the OUT body. All headers from the IN message will be copied to the OUT message, so headers are preserved during routing. Additionally Camel will add the HTTP response headers as well to the OUT message headers.

28.7. USING SYSTEM PROPERTIES

When setting useSystemProperties to true, the HTTP Client will look for the following System Properties and it will use it:

- ssl.TrustManagerFactory.algorithm
- javax.net.ssl.trustStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStoreProvider
- javax.net.ssl.trustStorePassword
- java.home
- ssl.KeyManagerFactory.algorithm
- javax.net.ssl.keyStoreType
- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreProvider
- javax.net.ssl.keyStorePassword
- http.proxyHost
- http.proxyPort
- http.nonProxyHosts
- http.keepAlive
- http.maxConnections

28.8. RESPONSE CODE

Camel will handle according to the HTTP response code:

- Response code is in the range 100..299, Camel regards it as a success response.
- Response code is in the range 300..399, Camel regards it as a redirection response and will throw a **HttpOperationFailedException** with the information.
- Response code is 400+, Camel regards it as an external server failure and will throw a **HttpOperationFailedException** with the information.

throwExceptionOnFailure The option, **throwExceptionOnFailure**, can be set to **false** to prevent the

HttpOperationFailedException from being thrown for failed response codes. This allows you to get any response from the remote server.
There is a sample below demonstrating this.

28.9. EXCEPTIONS

HttpOperationFailedException exception contains the following information:

- The HTTP status code
- The HTTP status line (text of the status code)
- Redirect location, if server returned a redirect
- Response body as a **java.lang.String**, if server provided a body as response

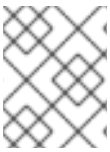
28.10. WHICH HTTP METHOD WILL BE USED

The following algorithm is used to determine what HTTP method should be used:
1. Use method provided as endpoint configuration (**httpMethod**).
2. Use method provided in header (**Exchange.HTTP_METHOD**).
3. **GET** if query string is provided in header.
4. **GET** if endpoint is configured with a query string.
5. **POST** if there is data to send (body is not **null**).
6. **GET** otherwise.

28.11. HOW TO GET ACCESS TO HTTPSERVLETREQUEST AND HTTPSERVLETRESPONSE

You can get access to these two using the Camel type converter system using

```
HttpServletRequest request = exchange.getIn().getBody(HttpServletRequest.class);
HttpServletResponse response = exchange.getIn().getBody(HttpServletResponse.class);
```



NOTE

You can get the request and response not just from the processor after the camel-jetty or camel-cxf endpoint.

28.12. CONFIGURING URI TO CALL

You can set the HTTP producer's URI directly form the endpoint URI. In the route below, Camel will call out to the external server, **oldhost**, using HTTP.

```
from("direct:start")
        .to("http://oldhost");
```

And the equivalent Spring sample:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="direct:start"/>
```

```
    <to uri="http://oldhost"/>
  </route>
</camelContext>
```

You can override the HTTP endpoint URI by adding a header with the key, **Exchange.HTTP_URI**, on the message.

```
from("direct:start")
  .setHeader(Exchange.HTTP_URI, constant("http://newhost"))
  .to("http://oldhost");
```

In the sample above Camel will call the http://newhost/ despite the endpoint is configured with http://oldhost/.
If the http endpoint is working in bridge mode, it will ignore the message header of **Exchange.HTTP_URI**.

28.13. CONFIGURING URI PARAMETERS

The **http** producer supports URI parameters to be sent to the HTTP server. The URI parameters can either be set directly on the endpoint URI or as a header with the key **Exchange.HTTP_QUERY** on the message.

```
from("direct:start")
  .to("http://oldhost?order=123&detail=short");
```

Or options provided in a header:

```
from("direct:start")
  .setHeader(Exchange.HTTP_QUERY, constant("order=123&detail=short"))
  .to("http://oldhost");
```

28.14. HOW TO SET THE HTTP METHOD (GET/PATCH/POST/PUT/DELETE/HEAD/OPTIONS/TRACE) TO THE HTTP PRODUCER

The HTTP component provides a way to set the HTTP request method by setting the message header. Here is an example:

```
from("direct:start")
  .setHeader(Exchange.HTTP_METHOD,
constant(org.apache.camel.component.http.HttpMethods.POST))
  .to("http://www.google.com")
  .to("mock:results");
```

The method can be written a bit shorter using the string constants:

```
.setHeader("CamelHttpMethod", constant("POST"))
```

And the equivalent Spring sample:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
```

```
    <from uri="direct:start"/>
    <setHeader name="CamelHttpMethod">
      <constant>POST</constant>
    </setHeader>
    <to uri="http://www.google.com"/>
    <to uri="mock:results"/>
  </route>
</camelContext>
```

28.15. USING CLIENT TIMEOUT - SO_TIMEOUT

See the HttpSOTimeoutTest unit test.

28.16. CONFIGURING A PROXY

The HTTP component provides a way to configure a proxy.

```
from("direct:start")
  .to("http://oldhost?proxyAuthHost=www.myproxy.com&proxyAuthPort=80");
```

There is also support for proxy authentication via the **proxyAuthUsername** and **proxyAuthPassword** options.

28.16.1. Using proxy settings outside of URI

To avoid System properties conflicts, you can set proxy configuration only from the CamelContext or URI.
Java DSL :

```
context.getGlobalOptions().put("http.proxyHost", "172.168.18.9");
context.getGlobalOptions().put("http.proxyPort", "8080");
```

Spring XML

```
<camelContext>
  <properties>
    <property key="http.proxyHost" value="172.168.18.9"/>
    <property key="http.proxyPort" value="8080"/>
  </properties>
</camelContext>
```

Camel will first set the settings from Java System or CamelContext Properties and then the endpoint proxy options if provided.
So you can override the system properties with the endpoint options.

There is also a **http.proxyScheme** property you can set to explicit configure the scheme to use.

28.17. CONFIGURING CHARSET

If you are using **POST** to send data you can configure the **charset** using the **Exchange** property:

```
exchange.setProperty(Exchange.CHARSET_NAME, "ISO-8859-1");
```

28.17.1. Sample with scheduled poll

This sample polls the Google homepage every 10 seconds and write the page to the file **message.html**:

```
from("timer://foo?fixedRate=true&delay=0&period=10000")
  .to("http://www.google.com")
  .setHeader(FileComponent.HEADER_FILE_NAME, "message.html")
  .to("file:target/google");
```

28.17.2. URI Parameters from the endpoint URI

In this sample we have the complete URI endpoint that is just what you would have typed in a web browser. Multiple URI parameters can of course be set using the **&** character as separator, just as you would in the web browser. Camel does no tricks here.

```
// we query for Camel at the Google page
template.sendBody("http://www.google.com/search?q=Camel", null);
```

28.17.3. URI Parameters from the Message

```
Map headers = new HashMap();
headers.put(Exchange.HTTP_QUERY, "q=Camel&lr=lang_en");
// we query for Camel and English language at Google
template.sendBody("http://www.google.com/search", null, headers);
```

In the header value above notice that it should **not** be prefixed with **?** and you can separate parameters as usual with the **&** char.

28.17.4. Getting the Response Code

You can get the HTTP response code from the HTTP component by getting the value from the Out message header with **Exchange.HTTP_RESPONSE_CODE**.

```
Exchange exchange = template.send("http://www.google.com/search", new Processor() {
  public void process(Exchange exchange) throws Exception {
    exchange.getIn().setHeader(Exchange.HTTP_QUERY, constant("hl=en&q=activemq"));
  }
});
Message out = exchange.getOut();
int responseCode = out.getHeader(Exchange.HTTP_RESPONSE_CODE, Integer.class);
```

28.18. DISABLING COOKIES

To disable cookies you can set the HTTP Client to ignore cookies by adding the following URI option:

```
httpClient.cookieSpec=ignore
```

28.19. BASIC AUTH WITH THE STREAMING MESSAGE BODY

In order to avoid the **NonRepeatableRequestException**, you need to do the Preemptive Basic Authentication by adding the option:
authenticationPreemptive=true

28.20. ADVANCED USAGE

If you need more control over the HTTP producer you should use the **HttpComponent** where you can set various classes to give you custom behavior.

28.20.1. Setting up SSL for HTTP Client

Using the JSSE Configuration Utility

The HTTP component supports SSL/TLS configuration through the Camel JSSE Configuration Utility. This utility greatly decreases the amount of component specific code you need to write and is configurable at the endpoint and component levels. The following examples demonstrate how to use the utility with the HTTP component.

Programmatic configuration of the component

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

HttpComponent httpComponent = getContext().getComponent("https", HttpComponent.class);
httpComponent.setSslContextParameters(scp);
```

Spring DSL based configuration of endpoint

```
<camel:sslContextParameters
   id="sslContextParameters">
 <camel:keyManagers
    keyPassword="keyPassword">
   <camel:keyStore
      resource="/users/home/server/keystore.jks"
      password="keystorePassword"/>
 </camel:keyManagers>
</camel:sslContextParameters>

<to uri="https://127.0.0.1/mail/?sslContextParameters=#sslContextParameters"/>
```

Configuring Apache HTTP Client Directly

Basically camel-http component is built on the top of Apache HttpClient. Please refer to SSL/TLS customization for details or have a look into the **org.apache.camel.component.http.HttpsServerTestSupport** unit test base class.
You can also implement a custom **org.apache.camel.component.http.HttpClientConfigurer** to do some configuration on the http client if you need full control of it.

However if you *just* want to specify the keystore and truststore you can do this with Apache HTTP **HttpClientConfigurer**, for example:

```
KeyStore keystore = ...;
KeyStore truststore = ...;

SchemeRegistry registry = new SchemeRegistry();
registry.register(new Scheme("https", 443, new SSLSocketFactory(keystore, "mypassword",
truststore)));
```

And then you need to create a class that implements **HttpClientConfigurer**, and registers https protocol providing a keystore or truststore per example above. Then, from your camel route builder class you can hook it up like so:

```
HttpComponent httpComponent = getContext().getComponent("http", HttpComponent.class);
httpComponent.setHttpClientConfigurer(new MyHttpClientConfigurer());
```

If you are doing this using the Spring DSL, you can specify your **HttpClientConfigurer** using the URI. For example:

```
<bean id="myHttpClientConfigurer"
 class="my.https.HttpClientConfigurer">
</bean>

<to uri="https://myhostname.com:443/myURL?httpClientConfigurer=myHttpClientConfigurer"/>
```

As long as you implement the HttpClientConfigurer and configure your keystore and truststore as described above, it will work fine.

Using HTTPS to authenticate gotchas

An end user reported that he had problem with authenticating with HTTPS. The problem was eventually resolved by providing a custom configured **org.apache.http.protocol.HttpContext**:

- 1. Create a (Spring) factory for HttpContexts:

```
public class HttpContextFactory {

  private String httpHost = "localhost";
  private String httpPort = 9001;

  private BasicHttpContext httpContext = new BasicHttpContext();
  private BasicAuthCache authCache = new BasicAuthCache();
  private BasicScheme basicAuth = new BasicScheme();

  public HttpContext getObject() {
    authCache.put(new HttpHost(httpHost, httpPort), basicAuth);

    httpContext.setAttribute(ClientContext.AUTH_CACHE, authCache);

    return httpContext;
  }

  // getter and setter
}
```

- 2. Declare an HttpContext in the Spring application context file:

```
<bean id="myHttpContext" factory-bean="httpContextFactory" factory-method="getObject"/>
```

- 3. Reference the context in the http URL:

```
<to uri="https://myhostname.com:443/myURL?httpContext=myHttpContext"/>
```

Using different SSLContextParameters

The HTTP component only support one instance of **org.apache.camel.support.jsse.SSLContextParameters** per component. If you need to use 2 or more different instances, then you need to setup multiple HTTP components as shown below. Where we have 2 components, each using their own instance of **sslContextParameters** property.

```
<bean id="http-foo" class="org.apache.camel.component.http.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams1"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>

<bean id="http-bar" class="org.apache.camel.component.http.HttpComponent">
  <property name="sslContextParameters" ref="sslContextParams2"/>
  <property name="x509HostnameVerifier" ref="hostnameVerifier"/>
</bean>
```

28.21. SPRING BOOT AUTO-CONFIGURATION

When using http with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-http-starter</artifactId>
</dependency>
```

The component supports 38 options, which are listed below.

Name	Description	Default	Type
camel.component.http.allow-java-serialized-object	Whether to allow java serialization when a request uses context-type=application/x-java-serialized-object. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	Boolean
camel.component.http.auth-caching-disabled	Disables authentication scheme caching.	false	Boolean

Name	Description	Default	Type
camel.component.http.automatic-retries-disabled	Disables automatic request recovery and re-execution.	false	Boolean
camel.component.http.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.http.client-connection-manager	To use a custom and shared HttpClientConnectionManager to manage connections. If this has been configured then this is always used for all endpoints created by this component. The option is a org.apache.http.conn.HttpClientConnectionManager type.		HttpClientConnectionManager
camel.component.http.connect-timeout	Determines the timeout in milliseconds until a connection is established. A timeout value of zero is interpreted as an infinite timeout. A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	Integer
camel.component.http.connection-request-timeout	The timeout in milliseconds used when requesting a connection from the connection manager. A timeout value of zero is interpreted as an infinite timeout. A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	Integer
camel.component.http.connection-state-disabled	Disables connection state tracking.	false	Boolean
camel.component.http.connection-time-to-live	The time for connection to live, the time unit is millisecond, the default value is always keep alive.		Long
camel.component.http.connections-per-route	The maximum number of connections per route.	20	Integer

Name	Description	Default	Type
camel.component.http.content-compression-disabled	Disables automatic content decompression.	false	Boolean
camel.component.http.cookie-management-disabled	Disables state (cookie) management.	false	Boolean
camel.component.http.cookie-store	To use a custom org.apache.http.client.CookieStore. By default the org.apache.http.impl.client.BasicCookieStore is used which is an in-memory only cookie store. Notice if bridgeEndpoint=true then the cookie store is forced to be a noop cookie store as cookie shouldn't be stored as we are just bridging (eg acting as a proxy). The option is a org.apache.http.client.CookieStore type.		CookieStore
camel.component.http.copy-headers	If this option is true then IN exchange headers will be copied to OUT exchange headers according to copy strategy. Setting this to false, allows to only include the headers from the HTTP response (not propagating IN headers).	true	Boolean
camel.component.http.default-user-agent-disabled	Disables the default user agent set by this builder if none has been provided by the user.	false	Boolean
camel.component.http.enabled	Whether to enable auto configuration of the http component. This is enabled by default.		Boolean
camel.component.http.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.http.http-binding	To use a custom HttpBinding to control the mapping between Camel message and HttpClient. The option is a org.apache.camel.http.common.HttpBinding type.		HttpBinding

Name	Description	Default	Type
camel.component.http.http-client-configurer	To use the custom HttpClientConfigurer to perform configuration of the HttpClient that will be used. The option is a org.apache.camel.component.http.HttpClientConfigurer type.		HttpClientConfigurer
camel.component.http.http-configuration	To use the shared HttpConfiguration as base configuration. The option is a org.apache.camel.http.common.HttpConfiguration type.		HttpConfiguration
camel.component.http.http-context	To use a custom org.apache.http.protocol.HttpContext when executing requests. The option is a org.apache.http.protocol.HttpContext type.		HttpContext
camel.component.http.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.http.max-total-connections	The maximum number of connections.	200	Integer
camel.component.http.proxy-auth-domain	Proxy authentication domain to use.		String
camel.component.http.proxy-auth-host	Proxy authentication host.		String
camel.component.http.proxy-auth-method	Proxy authentication method to use.		String
camel.component.http.proxy-auth-nt-host	Proxy authentication domain (workstation name) to use with NTML.		String

Name	Description	Default	Type
camel.component.http.proxy-auth-password	Proxy authentication password.		String
camel.component.http.proxy-auth-port	Proxy authentication port.		Integer
camel.component.http.proxy-auth-username	Proxy authentication username.		String
camel.component.http.redirect-handling-disabled	Disables automatic redirect handling.	false	Boolean
camel.component.http.response-payload-streaming-threshold	This threshold in bytes controls whether the response payload should be stored in memory as a byte array or be streaming based. Set this to -1 to always use streaming mode.	8192	Integer
camel.component.http.skip-request-headers	Whether to skip mapping all the Camel headers as HTTP request headers. If there are no data from Camel headers needed to be included in the HTTP request then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	Boolean
camel.component.http.skip-response-headers	Whether to skip mapping all the HTTP response headers to Camel headers. If there are no data needed from HTTP headers then this can avoid parsing overhead with many object allocations for the JVM garbage collector.	false	Boolean
camel.component.http.socket-timeout	Defines the socket timeout in milliseconds, which is the timeout for waiting for data or, put differently, a maximum period inactivity between two consecutive data packets). A timeout value of zero is interpreted as an infinite timeout. A negative value is interpreted as undefined (system default).	-1	Integer

Name	Description	Default	Type
camel.component.http.ssl-context-parameters	To configure security using SSLContextParameters. Important: Only one instance of org.apache.camel.support.jsse.SSLContextParameters is supported per HttpComponent. If you need to use 2 or more different instances, you need to define a new HttpComponent per instance you need. The option is a org.apache.camel.support.jsse.SSLContextParameters type.		SSLContextParameters
camel.component.http.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.http.x509-hostname-verifier	To use a custom X509HostnameVerifier such as DefaultHostnameVerifier or NoopHostnameVerifier. The option is a javax.net.ssl.HostnameVerifier type.		HostnameVerifier

CHAPTER 29. INFINISPAN

Both producer and consumer are supported

This component allows you to interact with Infinispan distributed data grid / cache using the Hot Rod procol. Infinispan is an extremely scalable, highly available key/value data store and data grid platform written in Java.

If you use Maven, you must add the following dependency to your **pom.xml**:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-infinispan</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

29.1. URI FORMAT

```
infinispan://cacheName?[options]
```

The producer allows sending messages to a remote cache using the HotRod protocol. The consumer allows listening for events from a remote cache using the HotRod protocol.

29.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

29.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

29.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

29.3. COMPONENT OPTIONS

The Infinispan component supports 26 options, which are listed below.

Name	Description	Default	Type
configuration (common)	Component configuration.		InfinispanRemoteConfiguration
hosts (common)	Specifies the host of the cache on Infinispan instance.		String
queryBuilder (common)	Specifies the query builder.		InfinispanQueryBuilder
secure (common)	Define if we are connecting to a secured Infinispan instance.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
customListener (consumer)	Returns the custom listener in use, if provided.		InfinispanRemoteCustomListener
eventTypes (consumer)	Specifies the set of event types to register by the consumer.Multiple event can be separated by comma. The possible event types are: CLIENT_CACHE_ENTRY_CREATED, CLIENT_CACHE_ENTRY_MODIFIED, CLIENT_CACHE_ENTRY_REMOVED, CLIENT_CACHE_ENTRY_EXPIRED, CLIENT_CACHE_FAILOVER.		String
defaultValue (producer)	Set a specific default value for some producer operations.		Object
key (producer)	Set a specific key for producer operations.		Object

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
oldValue (producer)	Set a specific old value for some producer operations.		Object

Name	Description	Default	Type
operation (producer)	The operation to perform. Enum values: • PUT • PUTASYNC • PUTALL • PUTALLASYNC • PUTIFABSENT • PUTIFABSENTASYNC • GET • GETORDEFAULT • CONTAINSKEY • CONTAINSVALUE • REMOVE • REMOVEASYNC • REPLACE • REPLACEASYNC • SIZE • CLEAR • CLEARASYNC • QUERY • STATS • COMPUTE • COMPUTEASYNC	PUT	InfinispanOperation
value (producer)	Set a specific value for producer operations.		Object
password (security)	Define the password to access the infinispan instance.		String
saslMechanism (security)	Define the SASL Mechanism to access the infinispan instance.		String

Name	Description	Default	Type
securityRealm (security)	Define the security realm to access the infinispan instance.		String
securityServerName (security)	Define the security server name to access the infinispan instance.		String
username (security)	Define the username to access the infinispan instance.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
cacheContainer (advanced)	**Autowired** Specifies the cache Container to connect.		RemoteCacheManager
cacheContainerConfiguration (advanced)	**Autowired** The CacheContainer configuration. Used if the cacheContainer is not defined.		Configuration
configurationProperties (advanced)	Implementation specific properties for the CacheManager.		Map
configurationUri (advanced)	An implementation specific URI for the CacheManager.		String
flags (advanced)	A comma separated list of org.infinispan.client.hotrod.Flag to be applied by default on each cache invocation.		String
remappingFunction (advanced)	Set a specific remappingFunction to use in a compute operation.		BiFunction
resultHeader (advanced)	Store the operation result in a header instead of the message body. By default, resultHeader == null and the query result is stored in the message body, any existing content in the message body is discarded. If resultHeader is set, the value is used as the name of the header to store the query result and the original message body is preserved. This value can be overridden by an in message header named: CamelInfinispanOperationResultHeader.		String

29.4. ENDPOINT OPTIONS

The Infinispan endpoint is configured using URI syntax:

```
infinispan:cacheName
```

with the following path and query parameters:

29.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
cacheName (common)	**Required** The name of the cache to use. Use current to use the existing cache name from the currently configured cached manager. Or use default for the default cache manager name.		String

29.4.2. Query Parameters (26 parameters)

Name	Description	Default	Type
hosts (common)	Specifies the host of the cache on Infinispan instance.		String
queryBuilder (common)	Specifies the query builder.		InfinispanQueryBuilder
secure (common)	Define if we are connecting to a secured Infinispan instance.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
customListener (consumer)	Returns the custom listener in use, if provided.		InfinispanRemoteCustomListener

Name	Description	Default	Type
eventTypes (consumer)	Specifies the set of event types to register by the consumer.Multiple event can be separated by comma. The possible event types are: CLIENT_CACHE_ENTRY_CREATED, CLIENT_CACHE_ENTRY_MODIFIED, CLIENT_CACHE_ENTRY_REMOVED, CLIENT_CACHE_ENTRY_EXPIRED, CLIENT_CACHE_FAILOVER.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
defaultValue (producer)	Set a specific default value for some producer operations.		Object
key (producer)	Set a specific key for producer operations.		Object
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
oldValue (producer)	Set a specific old value for some producer operations.		Object

Name	Description	Default	Type
operation (producer)	The operation to perform. Enum values: • PUT • PUTASYNC • PUTALL • PUTALLASYNC • PUTIFABSENT • PUTIFABSENTASYNC • GET • GETORDEFAULT • CONTAINSKEY • CONTAINSVALUE • REMOVE • REMOVEASYNC • REPLACE • REPLACEASYNC • SIZE • CLEAR • CLEARASYNC • QUERY • STATS • COMPUTE • COMPUTEASYNC	PUT	InfinispanOperation
value (producer)	Set a specific value for producer operations.		Object
password (security)	Define the password to access the infinispan instance.		String
saslMechanism (security)	Define the SASL Mechanism to access the infinispan instance.		String

Name	Description	Default	Type
securityRealm (security)	Define the security realm to access the infinispan instance.		String
securityServerName (security)	Define the security server name to access the infinispan instance.		String
username (security)	Define the username to access the infinispan instance.		String
cacheContainer (advanced)	**Autowired** Specifies the cache Container to connect.		RemoteCacheManager
cacheContainerConfiguration (advanced)	**Autowired** The CacheContainer configuration. Used if the cacheContainer is not defined.		Configuration
configurationProperties (advanced)	Implementation specific properties for the CacheManager.		Map
configurationUri (advanced)	An implementation specific URI for the CacheManager.		String
flags (advanced)	A comma separated list of org.infinispan.client.hotrod.Flag to be applied by default on each cache invocation.		String
remappingFunction (advanced)	Set a specific remappingFunction to use in a compute operation.		BiFunction
resultHeader (advanced)	Store the operation result in a header instead of the message body. By default, resultHeader == null and the query result is stored in the message body, any existing content in the message body is discarded. If resultHeader is set, the value is used as the name of the header to store the query result and the original message body is preserved. This value can be overridden by an in message header named: CamelInfinispanOperationResultHeader.		String

29.5. CAMEL OPERATIONS

This section lists all available operations, along with their header information.

Table 29.1. Table 1. Put Operations

Operation Name	Description
InfinispanOperation.PUT	Puts a key/value pair in the cache, optionally with expiration
InfinispanOperation.PUTASYNC	Asynchronously puts a key/value pair in the cache, optionally with expiration
InfinispanOperation.PUTIFABSENT	Puts a key/value pair in the cache if it did not exist, optionally with expiration
InfinispanOperation.PUTIFABSENTASYNC	Asynchronously puts a key/value pair in the cache if it did not exist, optionally with expiration

- **Required Headers**:
  - CamelInfinispanKey
  - CamelInfinispanValue
- **Optional Headers**:
  - CamelInfinispanLifespanTime
  - CamelInfinispanLifespanTimeUnit
  - CamelInfinispanMaxIdleTime
  - CamelInfinispanMaxIdleTimeUnit
- **Result Header**:
  - CamelInfinispanOperationResult

Table 29.2. Table 2. Put All Operations

Operation Name	Description
InfinispanOperation.PUTALL	Adds multiple entries to a cache, optionally with expiration
CamelInfinispanOperation.PUTALLASYNC	Asynchronously adds multiple entries to a cache, optionally with expiration

- **Required Headers**:
  - CamelInfinispanMap
- **Optional Headers**:
  - CamelInfinispanLifespanTime
  - CamelInfinispanLifespanTimeUnit
  - CamelInfinispanMaxIdleTime

- CamelInfinispanMaxIdleTimeUnit

Table 29.3. Table 3. Get Operations

Operation Name	Description
InfinispanOperation.GET	Retrieves the value associated with a specific key from the cache
InfinispanOperation.GETORDEFAULT	Retrieves the value, or default value, associated with a specific key from the cache

- **Required Headers**:
  - CamelInfinispanKey

Table 29.4. Table 4. Contains Key Operation

Operation Name	Description
InfinispanOperation.CONTAINSKEY	Determines whether a cache contains a specific key

- **Required Headers**
  - CamelInfinispanKey
- **Result Header**
  - CamelInfinispanOperationResult

Table 29.5. Table 5. Contains Value Operation

Operation Name	Description
InfinispanOperation.CONTAINSVALUE	Determines whether a cache contains a specific value

- **Required Headers**:
  - CamelInfinispanKey

Table 29.6. Table 6. Remove Operations

Operation Name	Description
InfinispanOperation.REMOVE	Removes an entry from a cache, optionally only if the value matches a given one
InfinispanOperation.REMOVEASYNC	Asynchronously removes an entry from a cache, optionally only if the value matches a given one

- **Required Headers**:

- CamelInfinispanKey

- **Optional Headers**:
  - CamelInfinispanValue
- **Result Header**:
  - CamelInfinispanOperationResult

Table 29.7. Table 7. Replace Operations

Operation Name	Description
InfinispanOperation.REPLACE	Conditionally replaces an entry in the cache, optionally with expiration
InfinispanOperation.REPLACEASYNC	Asynchronously conditionally replaces an entry in the cache, optionally with expiration

- **Required Headers**:
  - CamelInfinispanKey
  - CamelInfinispanValue
  - CamelInfinispanOldValue
- **Optional Headers**:
  - CamelInfinispanLifespanTime
  - CamelInfinispanLifespanTimeUnit
  - CamelInfinispanMaxIdleTime
  - CamelInfinispanMaxIdleTimeUnit
- **Result Header**:
  - CamelInfinispanOperationResult

Table 29.8. Table 8. Clear Operations

Operation Name	Description
InfinispanOperation.CLEAR	Clears the cache
InfinispanOperation.CLEARASYNC	Asynchronously clears the cache

Table 29.9. Table 9. Size Operation

Operation Name	Description
InfinispanOperation.SIZE	Returns the number of entries in the cache

- **Result Header**
  - CamelInfinispanOperationResult

Table 29.10. Table 10. Stats Operation

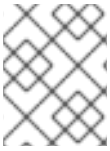
Operation Name	Description
InfinispanOperation.STATS	Returns statistics about the cache

- **Result Header**:
  - CamelInfinispanOperationResult

Table 29.11. Table 11. Query Operation

Operation Name	Description
InfinispanOperation.QUERY	Executes a query on the cache

- **Required Headers**:
  - CamelInfinispanQueryBuilder
- **Result Header**:
  - CamelInfinispanOperationResult



NOTE

Write methods like put(key, value) and remove(key) do not return the previous value by default.

29.6. MESSAGE HEADERS

Name	Default Value	Type	Context	Description
CamelInfinispanCacheName	**null**	String	Shared	The cache participating in the operation or event.
CamelInfinispanOperation	**PUT**	InfinispanOperation	Producer	The operation to perform.

Name	Default Value	Type	Context	Description
CamelInfinispanMap	**null**	Map	Producer	A Map to use in case of CamelInfinispanOperationPutAll operation
CamelInfinispanKey	**null**	Object	Shared	The key to perform the operation to or the key generating the event.
CamelInfinispanValue	**null**	Object	Producer	The value to use for the operation.
CamelInfinispanEventType	**null**	String	Consumer	The type of the received event.
CamelInfinispanLifespanTime	**null**	long	Producer	The Lifespan time of a value inside the cache. Negative values are interpreted as infinity.
CamelInfinispanTimeUnit	**null**	String	Producer	The Time Unit of an entry Lifespan Time.
CamelInfinispanMaxIdleTime	**null**	long	Producer	The maximum amount of time an entry is allowed to be idle for before it is considered as expired.
CamelInfinispanMaxIdleTimeUnit	**null**	String	Producer	The Time Unit of an entry Max Idle Time.
CamelInfinispanQueryBuilder	null	InfinispanQueryBuilder	Producer	The QueryBuilde to use for QUERY command, if not present the command defaults to InifinispanConfiguration's one
CamelInfinispanOperationResultHeader	null	String	Producer	Store the operation result in a header instead of the message body

29.7. EXAMPLES

- Put a key/value into a named cache:

```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION).constant(InfinispanOperation.PUT) (1)
    .setHeader(InfinispanConstants.KEY).constant("123") (2)
    .to("infinispan:myCacheName&cacheContainer=#cacheContainer"); (3)
```

Where,

- 1 - Set the operation to perform
- 2 - Set the key used to identify the element in the cache
- 3 - Use the configured cache manager **cacheContainer** from the registry to put an element to the cache named **myCacheName**
  It is possible to configure the lifetime and/or the idle time before the entry expires and gets evicted from the cache, as example:

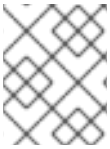
```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION).constant(InfinispanOperation.GET)
    .setHeader(InfinispanConstants.KEY).constant("123")
    .setHeader(InfinispanConstants.LIFESPAN_TIME).constant(100L) (1)

.setHeader(InfinispanConstants.LIFESPAN_TIME_UNIT.constant(TimeUnit.MILLISECONDS.t
oString()) (2)
    .to("infinispan:myCacheName");
```

where,

- 1 - Set the lifespan of the entry
- 2 - Set the time unit for the lifespan

Queries

```
from("direct:start")
   .setHeader(InfinispanConstants.OPERATION, InfinispanConstants.QUERY)
   .setHeader(InfinispanConstants.QUERY_BUILDER, new InfinispanQueryBuilder() {
      @Override
      public Query build(QueryFactory<Query> qf) {
         return qf.from(User.class).having("name").like("%abc%").build();
      }
   })
   .to("infinispan:myCacheName?cacheContainer=#cacheManager") ;
```



NOTE

The .proto descriptors for domain objects must be registered with the remote Data Grid server, see Remote Query Example in the official Infinispan documentation.

Custom Listeners

```
from("infinispan://?cacheContainer=#cacheManager&customListener=#myCustomListener")
  .to("mock:result");
```

The instance of **myCustomListener** must exist and Camel should be able to look it up from the **Registry**. Users are encouraged to extend the **org.apache.camel.component.infinispan.remote.InfinispanRemoteCustomListener** class and annotate the resulting class with **@ClientListener** which can be found found in package **org.infinispan.client.hotrod.annotation**.

29.8. USING THE INFINISPAN BASED IDEMPOTENT REPOSITORY

In this section we will use the Infinispan based idempotent repository.

Java Example

```
InfinispanRemoteConfiguration conf = new InfinispanRemoteConfiguration(); (1)
conf.setHosts("localhost:1122")

InfinispanRemoteIdempotentRepository repo = new
InfinispanRemoteIdempotentRepository("idempotent");  (2)
repo.setConfiguration(conf);

context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() {
        from("direct:start")
            .idempotentConsumer(header("MessageID"), repo) (3)
            .to("mock:result");
    }
});
```

where,

- 1 - Configure the cache
- 2 - Configure the repository bean
- 3 - Set the repository to the route

XML Example

```
<bean id="infinispanRepo"
class="org.apache.camel.component.infinispan.remote.InfinispanRemoteIdempotentRepository"
destroy-method="stop">
  <constructor-arg value="idempotent"/> (1)
  <property name="configuration"> (2)
    <bean class="org.apache.camel.component.infinispan.remote.InfinispanRemoteConfiguration">
      <property name="hosts" value="localhost:11222"/>
    </bean>
  </property>
</bean>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:start" />
        <idempotentConsumer messageIdRepositoryRef="infinispanRepo"> (3)
            <header>MessageID</header>
            <to uri="mock:result" />
        </idempotentConsumer>
    </route>
</camelContext>
```

where,

- 1 - Set the name of the cache that will be used by the repository
- 2 - Configure the repository bean

- 3 - Set the repository to the route

29.9. USING THE INFINISPAN BASED AGGREGATION REPOSITORY

In this section we will use the Infinispan based aggregation repository.

Java Example

```
InfinispanRemoteConfiguration conf = new InfinispanRemoteConfiguration(); (1)
conf.setHosts("localhost:1122")

InfinispanRemoteAggregationRepository repo = new InfinispanRemoteAggregationRepository();  (2)
repo.setCacheName("aggregation");
repo.setConfiguration(conf);

context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() {
        from("direct:start")
                .aggregate(header("MessageID"))
                .completionSize(3)
                .aggregationRepository(repo) (3)
                .aggregationStrategyRef("myStrategy")
                .to("mock:result");
    }
});
```

where,

- 1 - Configure the cache
- 2 - Create the repository bean
- 3 - Set the repository to the route

XML Example

```
<bean id="infinispanRepo"
class="org.apache.camel.component.infinispan.remote.InfinispanRemoteAggregationRepository"
destroy-method="stop">
  <constructor-arg value="aggregation"/> (1)
  <property name="configuration"> (2)
    <bean class="org.apache.camel.component.infinispan.remote.InfinispanRemoteConfiguration">
      <property name="hosts" value="localhost:11222"/>
    </bean>
  </property>
</bean>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:start" />
        <aggregate strategyRef="myStrategy"
                completionSize="3"
                aggregationRepositoryRef="infinispanRepo"> (3)
            <correlationExpression>
```

```
            <header>MessageID</header>
          </correlationExpression>
          <to uri="mock:result"/>
        </aggregate>
      </route>
    </camelContext>
```

where,

- 1 - Set the name of the cache that will be used by the repository
- 2 - Configure the repository bean
- 3 - Set the repository to the route

NOTE

With the release of Infinispan 11, it is required to set the encoding configuration on any cache created. This is critical for consuming events too. For more information have a look at Data Encoding and MediaTypes in the official Infinispan documentation.

29.10. SPRING BOOT AUTO-CONFIGURATION

When using infinispan with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-infinispan-starter</artifactId>
</dependency>
```

The component supports 23 options, which are listed below.

Name	Description	Default	Type
camel.component.infinispan.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.infinispan.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.infinispan.cache-container	Specifies the cache Container to connect. The option is a org.infinispan.client.hotrod.RemoteCacheManager type.		RemoteCacheManager
camel.component.infinispan.cache-container-configuration	The CacheContainer configuration. Used if the cacheContainer is not defined. The option is a org.infinispan.client.hotrod.configuration.Configuration type.		Configuration
camel.component.infinispan.configuration	Component configuration. The option is a org.apache.camel.component.infinispan.remote.InfinispanRemoteConfiguration type.		InfinispanRemoteConfiguration
camel.component.infinispan.configuration-properties	Implementation specific properties for the CacheManager.		Map
camel.component.infinispan.configuration-uri	An implementation specific URI for the CacheManager.		String
camel.component.infinispan.custom-listener	Returns the custom listener in use, if provided. The option is a org.apache.camel.component.infinispan.remote.InfinispanRemoteCustomListener type.		InfinispanRemoteCustomListener
camel.component.infinispan.enabled	Whether to enable auto configuration of the infinispan component. This is enabled by default.		Boolean
camel.component.infinispan.event-types	Specifies the set of event types to register by the consumer.Multiple event can be separated by comma. The possible event types are: CLIENT_CACHE_ENTRY_CREATED, CLIENT_CACHE_ENTRY_MODIFIED, CLIENT_CACHE_ENTRY_REMOVED, CLIENT_CACHE_ENTRY_EXPIRED, CLIENT_CACHE_FAILOVER.		String
camel.component.infinispan.flags	A comma separated list of org.infinispan.client.hotrod.Flag to be applied by default on each cache invocation.		String
camel.component.infinispan.hosts	Specifies the host of the cache on Infinispan instance.		String

Name	Description	Default	Type
camel.component.infinispan.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.infinispan.operation	The operation to perform.		InfinispanOperation
camel.component.infinispan.password	Define the password to access the infinispan instance.		String
camel.component.infinispan.query-builder	Specifies the query builder. The option is a org.apache.camel.component.infinispan.InfinispanQueryBuilder type.		InfinispanQueryBuilder
camel.component.infinispan.remapping-function	Set a specific remappingFunction to use in a compute operation. The option is a java.util.function.BiFunction type.		BiFunction
camel.component.infinispan.result-header	Store the operation result in a header instead of the message body. By default, resultHeader == null and the query result is stored in the message body, any existing content in the message body is discarded. If resultHeader is set, the value is used as the name of the header to store the query result and the original message body is preserved. This value can be overridden by an in message header named: CamelInfinispanOperationResultHeader.		String
camel.component.infinispan.sasl-mechanism	Define the SASL Mechanism to access the infinispan instance.		String
camel.component.infinispan.secure	Define if we are connecting to a secured Infinispan instance.	false	Boolean
camel.component.infinispan.security-realm	Define the security realm to access the infinispan instance.		String

Name	Description	Default	Type
camel.component.infinispan.security-server-name	Define the security server name to access the infinispan instance.		String
camel.component.infinispan.username	Define the username to access the infinispan instance.		String

CHAPTER 30. JIRA

Both producer and consumer are supported

The JIRA component interacts with the JIRA API by encapsulating Atlassian's REST Java Client for JIRA. It currently provides polling for new issues and new comments. It is also able to create new issues, add comments, change issues, add/remove watchers, add attachment and transition the state of an issue.

Rather than webhooks, this endpoint relies on simple polling. Reasons include:

- Concern for reliability/stability
- The types of payloads we're polling aren't typically large (plus, paging is available in the API)
- The need to support apps running somewhere not publicly accessible where a webhook would fail

Note that the JIRA API is fairly expansive. Therefore, this component could be easily expanded to provide additional interactions.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jira</artifactId>
  <version>${camel-version}</version>
</dependency>
```

30.1. URI FORMAT

```
jira://type[?options]
```

The Jira type accepts the following operations:

For consumers:

- newIssues: retrieve only new issues after the route is started
- newComments: retrieve only new comments after the route is started
- watchUpdates: retrieve only updated fields/issues based on provided jql

For producers:

- addIssue: add an issue
- addComment: add a comment on a given issue
- attach: add an attachment on a given issue
- deleteIssue: delete a given issue
- updateIssue: update fields of a given issue

- transitionIssue: transition a status of a given issue
- watchers: add/remove watchers of a given issue

As Jira is fully customizable, you must assure the fields IDs exists for the project and workflow, as they can change between different Jira servers.

30.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

30.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

30.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

30.3. COMPONENT OPTIONS

The Jira component supports 12 options, which are listed below.

Name	Description	Default	Type
delay (common)	Time in milliseconds to elapse for the next poll.	6000	Integer
jiraUrl (common)	**Required** The Jira server url, example: .		String

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
configuration (advanced)	To use a shared base jira configuration.		JiraConfiguration
accessToken (security)	(OAuth only) The access token generated by the Jira server.		String
consumerKey (security)	(OAuth only) The consumer key from Jira settings.		String
password (security)	(Basic authentication only) The password to authenticate to the Jira server. Use only if username basic authentication is used.		String
privateKey (security)	(OAuth only) The private key generated by the client to encrypt the conversation to the server.		String

Name	Description	Default	Type
username (security)	(Basic authentication only) The username to authenticate to the Jira server. Use only if OAuth is not enabled on the Jira server. Do not set the username and OAuth token parameter, if they are both set, the username basic authentication takes precedence.		String
verificationCode (security)	(OAuth only) The verification code from Jira generated in the first step of the authorization proccess.		String

30.4. ENDPOINT OPTIONS

The Jira endpoint is configured using URI syntax:

```
jira:type
```

with the following path and query parameters:

30.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
type (common)	**Required** Operation to perform. Consumers: NewIssues, NewComments. Producers: AddIssue, AttachFile, DeleteIssue, TransitionIssue, UpdateIssue, Watchers. See this class javadoc description for more information. Enum values: • ADDCOMMENT • ADDISSUE • ATTACH • DELETEISSUE • NEWISSUES • NEWCOMMENTS • WATCHUPDATES • UPDATEISSUE • TRANSITIONISSUE • WATCHERS • ADDISSUELINK • ADDWORKLOG • FETCHISSUE • FETCHCOMMENTS		JiraType

30.4.2. Query Parameters (16 parameters)

Name	Description	Default	Type
delay (common)	Time in milliseconds to elapse for the next poll.	6000	Integer
jiraUrl (common)	**Required** The Jira server url, example: .		String

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
jql (consumer)	JQL is the query language from JIRA which allows you to retrieve the data you want. For example jql=project=MyProject Where MyProject is the product key in Jira. It is important to use the RAW() and set the JQL inside it to prevent camel parsing it, example: RAW(project in (MYP, COM) AND resolution = Unresolved).		String
maxResults (consumer)	Max number of issues to search for.	50	Integer
sendOnlyUpdatedField (consumer)	Indicator for sending only changed fields in exchange body or issue object. By default consumer sends only changed fields.	true	boolean
watchedFields (consumer)	Comma separated list of fields to watch for changes. Status,Priority are the defaults.	Status, Priority	String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
accessToken (security)	(OAuth only) The access token generated by the Jira server.		String
consumerKey (security)	(OAuth only) The consumer key from Jira settings.		String
password (security)	(Basic authentication only) The password to authenticate to the Jira server. Use only if username basic authentication is used.		String
privateKey (security)	(OAuth only) The private key generated by the client to encrypt the conversation to the server.		String
username (security)	(Basic authentication only) The username to authenticate to the Jira server. Use only if OAuth is not enabled on the Jira server. Do not set the username and OAuth token parameter, if they are both set, the username basic authentication takes precedence.		String
verificationCode (security)	(OAuth only) The verification code from Jira generated in the first step of the authorization proccess.		String

30.5. CLIENT FACTORY

You can bind the **JiraRestClientFactory** with name **JiraRestClientFactory** in the registry to have it automatically set in the Jira endpoint.

30.6. AUTHENTICATION

Camel-jira supports Basic Authentication and OAuth 3 legged authentication.

We recommend to use OAuth whenever possible, as it provides the best security for your users and system.

30.6.1. Basic authentication requirements:

- An username and password

30.6.2. OAuth authentication requirements:

Follow the tutorial in Jira OAuth documentation to generate the client private key, consumer key, verification code and access token.

- a private key, generated locally on your system.
- A verification code, generated by Jira server.
- The consumer key, set in the Jira server settings.
- An access token, generated by Jira server.

30.7. JQL

The JQL URI option is used by both consumer endpoints. Theoretically, items like "project key", etc. could be URI options themselves. However, by requiring the use of JQL, the consumers become much more flexible and powerful.

At the bare minimum, the consumers will require the following:

```
jira://[type]?[required options]&jql=project=[project key]
```

One important thing to note is that the newIssues consumer will automatically set the JQL as:

- append **ORDER BY key desc** to your JQL
- prepend **id > latestIssueId** to retrieve issues added after the camel route was started.

This is in order to optimize startup processing, rather than having to index every single issue in the project.

Another note is that, similarly, the newComments consumer will have to index every single issue **and** comment in the project. Therefore, for large projects, it's **vital** to optimize the JQL expression as much as possible. For example, the JIRA Toolkit Plugin includes a "Number of comments" custom field – use '"Number of comments" > 0' in your query. Also try to minimize based on state (status=Open), increase the polling delay, etc. Example:

```
jira://[type]?[required options]&jql=RAW(project=[project key] AND status in (Open, \"Coding In Progress\") AND \"Number of comments\">0)"
```

30.8. OPERATIONS

See a list of required headers to set when using the Jira operations. The author field for the producers is automatically set to the authenticated user in the Jira side.

If any required field is not set, then an IllegalArgumentException is throw.

There are operations that requires **id** for fields suchs as: issue type, priority, transition. Check the valid **id** on your jira project as they may differ on a jira installation and project workflow.

30.9. ADDISSUE

Required:

- **ProjectKey**: The project key, example: CAMEL, HHH, MYP.
- **IssueTypeId** or **IssueTypeName**: The **id** of the issue type or the name of the issue type, you can see the valid list in **http://jira_server/rest/api/2/issue/createmeta?projectKeys=SAMPLE_KEY**.
- **IssueSummary**: The summary of the issue.

Optional:

- **IssueAssignee**: the assignee user
- **IssuePriorityId** or **IssuePriorityName**: The priority of the issue, you can see the valid list in **http://jira_server/rest/api/2/priority**.
- **IssueComponents**: A list of string with the valid component names.
- **IssueWatchersAdd**: A list of strings with the usernames to add to the watcher list.
- **IssueDescription**: The description of the issue.

30.10. ADDCOMMENT

Required:

- **IssueKey**: The issue key identifier.
- body of the exchange is the description.

30.11. ATTACH

Only one file should attach per invocation.

Required:

- **IssueKey**: The issue key identifier.
- body of the exchange should be of type **File**

30.12. DELETEISSUE

Required:

- **IssueKey**: The issue key identifier.

30.13. TRANSITIONISSUE

Required:

- **IssueKey**: The issue key identifier.
- **IssueTransitionId**: The issue transition **id**.
- body of the exchange is the description.

30.14. UPDATEISSUE

- **IssueKey**: The issue key identifier.
- **IssueTypeId** or **IssueTypeName**: The **id** of the issue type or the name of the issue type, you can see the valid list in **http://jira_server/rest/api/2/issue/createmeta?projectKeys=SAMPLE_KEY**.
- **IssueSummary**: The summary of the issue.
- **IssueAssignee**: the assignee user
- **IssuePriorityId** or **IssuePriorityName**: The priority of the issue, you can see the valid list in **http://jira_server/rest/api/2/priority**.
- **IssueComponents**: A list of string with the valid component names.
- **IssueDescription**: The description of the issue.

30.15. WATCHER

- **IssueKey**: The issue key identifier.
- **IssueWatchersAdd**: A list of strings with the usernames to add to the watcher list.
- **IssueWatchersRemove**: A list of strings with the usernames to remove from the watcher list.

30.16. WATCHUPDATES (CONSUMER)

- **watchedFields** Comma separated list of fields to watch for changes i.e **Status,Priority,Assignee,Components** etc.
- **sendOnlyUpdatedField** By default only changed field is send as the body.

All messages also contain following headers that add additional info about the change:

- **issueKey**: Key of the updated issue
- **changed**: name of the updated field (i.e Status)
- **watchedIssues**: list of all issue keys that are watched in the time of update

30.17. SPRING BOOT AUTO-CONFIGURATION

When using jira with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
```

```
  <artifactId>camel-jira-starter</artifactId>
</dependency>
```

The component supports 13 options, which are listed below.

Name	Description	Default	Type
camel.component.jira.access-token	(OAuth only) The access token generated by the Jira server.		String
camel.component.jira.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.jira.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.jira.configuration	To use a shared base jira configuration. The option is a org.apache.camel.component.jira.JiraConfiguration type.		JiraConfiguration
camel.component.jira.consumer-key	(OAuth only) The consumer key from Jira settings.		String
camel.component.jira.delay	Time in milliseconds to elapse for the next poll.	6000	Integer
camel.component.jira.enabled	Whether to enable auto configuration of the jira component. This is enabled by default.		Boolean
camel.component.jira.jira-url	The Jira server url, example: http://my_jira.com:8081/.		String

Name	Description	Default	Type
camel.component.jira.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.jira.password	(Basic authentication only) The password to authenticate to the Jira server. Use only if username basic authentication is used.		String
camel.component.jira.private-key	(OAuth only) The private key generated by the client to encrypt the conversation to the server.		String
camel.component.jira.username	(Basic authentication only) The username to authenticate to the Jira server. Use only if OAuth is not enabled on the Jira server. Do not set the username and OAuth token parameter, if they are both set, the username basic authentication takes precedence.		String
camel.component.jira.verification-code	(OAuth only) The verification code from Jira generated in the first step of the authorization proccess.		String

CHAPTER 31. JMS

Both producer and consumer are supported

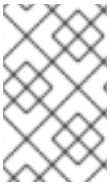
This component allows messages to be sent to (or consumed from) a JMS Queue or Topic. It uses Spring's JMS support for declarative transactions, including Spring's **JmsTemplate** for sending and a **MessageListenerContainer** for consuming.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jms</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

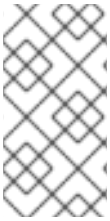
NOTE

Using ActiveMQ
If you are using Apache ActiveMQ, you should prefer the ActiveMQ component as it has been optimized for ActiveMQ. All of the options and samples on this page are also valid for the ActiveMQ component.



NOTE

Transacted and caching
See section **Transactions and Cache Levels**below if you are using transactions with JMS as it can impact performance.



NOTE

Request/Reply over JMS
Make sure to read the section *Request-reply over JMS* further below on this page for important notes about request/reply, as Camel offers a number of options to configure for performance, and clustered environments.

31.1. URI FORMAT

```
jms:[queue:|topic:]destinationName[?options]
```

Where **destinationName** is a JMS queue or topic name. By default, the **destinationName** is interpreted as a queue name. For example, to connect to the queue, **FOO.BAR** use:

```
jms:FOO.BAR
```

You can include the optional **queue:** prefix, if you prefer:

```
jms:queue:FOO.BAR
```

To connect to a topic, you *must* include the **topic:** prefix. For example, to connect to the topic, **Stocks.Prices**, use:

```
jms:topic:Stocks.Prices
```

You append query options to the URI by using the following format,

?option=value&option=value&...

31.1.1. Using ActiveMQ

The JMS component reuses Spring 2's **JmsTemplate** for sending messages. This is not ideal for use in a non-J2EE container and typically requires some caching in the JMS provider to avoid poor performance .

If you intend to use Apache ActiveMQ as your message broker, the recommendation is that you do one of the following:

- Use the ActiveMQ component, which is already optimized to use ActiveMQ efficiently
- Use the **PoolingConnectionFactory** in ActiveMQ.

31.1.2. Transactions and Cache Levels

If you are consuming messages and using transactions (**transacted=true**) then the default settings for cache level can impact performance.

If you are using XA transactions then you cannot cache as it can cause the XA transaction to not work properly.

If you are **not** using XA, then you should consider caching as it speeds up performance, such as setting **cacheLevelName=CACHE_CONSUMER**.

The default setting for **cacheLevelName** is **CACHE_AUTO**. This default auto detects the mode and sets the cache level accordingly to:

- **CACHE_CONSUMER** if **transacted=false**
- **CACHE_NONE** if **transacted=true**

So you can say the default setting is conservative. Consider using **cacheLevelName=CACHE_CONSUMER** if you are using non-XA transactions.

31.1.3. Durable Subscriptions

If you wish to use durable topic subscriptions, you need to specify both **clientId** and **durableSubscriptionName**. The value of the **clientId** must be unique and can only be used by a single JMS connection instance in your entire network. You may prefer to use Virtual Topics instead to avoid this limitation. More background on durable messaging here.

31.1.4. Message Header Mapping

When using message headers, the JMS specification states that header names must be valid Java identifiers. So try to name your headers to be valid Java identifiers. One benefit of doing this is that you can then use your headers inside a JMS Selector (whose SQL92 syntax mandates Java identifier syntax for headers).

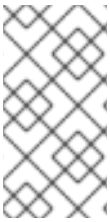
A simple strategy for mapping header names is used by default. The strategy is to replace any dots and

hyphens in the header name as shown below and to reverse the replacement when the header name is restored from a JMS message sent over the wire. What does this mean? No more losing method names to invoke on a bean component, no more losing the filename header for the File Component, and so on.

The current header name strategy for accepting header names in Camel is as follows:

- Dots are replaced by `*DOT*` and the replacement is reversed when Camel consume the message
- Hyphen is replaced by `*HYPHEN*` and the replacement is reversed when Camel consumes the message

You can configure many different properties on the JMS endpoint, which map to properties on the **JMSConfiguration** object.



NOTE

Mapping to Spring JMS
Many of these properties map to properties on Spring JMS, which Camel uses for sending and receiving messages. So you can get more information about these properties by consulting the relevant Spring documentation.

31.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

31.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

31.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

31.3. COMPONENT OPTIONS

The JMS component supports 98 options, which are listed below.

Name	Description	Default	Type
clientId (common)	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String
connectionFactory (common)	The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory
disableReplyTo (common)	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	boolean
durableSubscriptionName (common)	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String

Name	Description	Default	Type
jmsMessageType (common)	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it. Enum values: • Bytes • Map • Object • Stream • Text		JmsMessageType
replyTo (common)	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean
acknowledgementModeName (consumer)	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE. Enum values: • SESSION_TRANSACTED • CLIENT_ACKNOWLEDGE • AUTO_ACKNOWLEDGE • DUPS_OK_ACKNOWLEDGE	AUTO_ACKNOWLEDGE	String

Name	Description	Default	Type
artemisConsumerPriority (consumer)	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		int
asyncConsumer (consumer)	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean
cacheLevel (consumer)	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		int

Name	Description	Default	Type
cacheLevelName (consumer)	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION	CACHE_AUTO	String
concurrentConsumers (consumer)	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	int
maxConcurrentConsumers (consumer)	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		int
replyToDeliveryPersistent (consumer)	Specifies whether to use persistent delivery by default for replies.	true	boolean
selector (consumer)	Sets the JMS selector to use.		String

Name	Description	Default	Type
subscriptionDurable (consumer)	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	boolean
subscriptionName (consumer)	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String
subscriptionShared (consumer)	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	boolean
acceptMessagesWhileStopping (consumer (advanced))	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	boolean

Name	Description	Default	Type
allowReplyManagerQuickStop (consumer (advanced))	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	boolean
consumerType (consumer (advanced))	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType

Name	Description	Default	Type
defaultTaskExecutorType (consumer (advanced))	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers. Enum values: • ThreadPool • SimpleAsync		DefaultTaskExecutorType
eagerLoadingOfProperties (consumer (advanced))	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	boolean
eagerPoisonBody (consumer (advanced))	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
exposeListenerSession (consumer (advanced))	Specifies whether the listener session should be exposed when consuming messages.	false	boolean
replyToSameDestinationAllowed (consumer (advanced))	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	boolean
taskExecutor (consumer (advanced))	Allows you to specify a custom task executor for consuming messages.		TaskExecutor

Name	Description	Default	Type
deliveryDelay (producer)	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	long
deliveryMode (producer)	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2. Enum values: • 1 • 2		Integer
deliveryPersistent (producer)	Specifies whether persistent delivery is used by default.	true	boolean
explicitQosEnabled (producer)	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
formatDateHeadersToIso8601 (producer)	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
preserveMessageQos (producer)	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	boolean
priority (producer)	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect. Enum values: • 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9	4	int
replyToConcurrentConsumers (producer)	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	int
replyToMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		int

Name	Description	Default	Type
replyToOnTimeoutMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	int
replyToOverride (producer)	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String
replyToType (producer)	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive. Enum values: • Temporary • Shared • Exclusive		ReplyToType
requestTimeout (producer)	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option.	20000	long
timeToLive (producer)	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	long

Name	Description	Default	Type
allowAdditionalHeaders (producer (advanced))	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
allowNullBody (producer (advanced))	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	boolean
alwaysCopyMessage (producer (advanced))	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	boolean
correlationProperty (producer (advanced))	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String
disableTimeToLive (producer (advanced))	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	boolean
forceSendOriginalMessage (producer (advanced))	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	boolean

Name	Description	Default	Type
includeSentJMSMessageID (producer (advanced))	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	boolean
replyToCacheLevelName (producer (advanced))	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION		String
replyToDestinationSelectorName (producer (advanced))	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String
streamMessageTypeEnabled (producer (advanced))	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	boolean

Name	Description	Default	Type
allowAutoWiredConnectionFactory (advanced)	Whether to auto-discover ConnectionFactory from the registry, if no connection factory has been configured. If only one instance of ConnectionFactory is found then it will be used. This is enabled by default.	true	boolean
allowAutoWiredDestinationResolver (advanced)	Whether to auto-discover DestinationResolver from the registry, if no destination resolver has been configured. If only one instance of DestinationResolver is found then it will be used. This is enabled by default.	true	boolean
allowSerializedHeaders (advanced)	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
artemisStreamingEnabled (advanced)	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	boolean
asyncStartListener (advanced)	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	boolean
asyncStopListener (advanced)	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
configuration (advanced)	To use a shared JMS configuration.		JmsConfiguration
destinationResolver (advanced)	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry).		DestinationResolver
errorHandler (advanced)	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler.		ErrorHandler
exceptionListener (advanced)	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions.		ExceptionListener
idleConsumerLimit (advanced)	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	int
idleTaskExecutionLimit (advanced)	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	int
includeAllJMSXProperties (advanced)	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	boolean

Name	Description	Default	Type
jmsKeyFormatStrategy (advanced)	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation. Enum values: • default • passthrough		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	boolean
maxMessagesPerTask (advanced)	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	int
messageConverter (advanced)	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message.		MessageConverter
messageCreatedStrategy (advanced)	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message.		MessageCreatedStrategy
messageIdEnabled (advanced)	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	boolean

Name	Description	Default	Type
messageListenerContainerFactory (advanced)	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom.		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	boolean
pubSubNoLocal (advanced)	Specifies whether to inhibit the delivery of messages published by its own connection.	false	boolean
queueBrowseStrategy (advanced)	To use a custom QueueBrowseStrategy when browsing queues.		QueueBrowseStrategy
receiveTimeout (advanced)	The timeout for receiving messages (in milliseconds).	1000	long
recoveryInterval (advanced)	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds.	5000	long
requestTimeoutCheckerInterval (advanced)	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout.	1000	long
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean

Name	Description	Default	Type
transferException (advanced)	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	boolean
transferExchange (advanced)	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	boolean
useMessageIDAsCorrelationID (advanced)	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	boolean
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleepingTime (advanced)	Interval in millis to sleep each time while waiting for provisional correlation id to be updated.	100	long

Name	Description	Default	Type
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
errorHandlerLoggingLevel (logging)	Allows to configure the default errorHandler logging level for logging uncaught exceptions. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	boolean
password (security)	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
username (security)	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
transacted (transaction)	Specifies whether to use transacted mode.	false	boolean

Name	Description	Default	Type
transactedInOut (transaction)	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	boolean
lazyCreateTransactionManager (transaction (advanced))	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	boolean
transactionManager (transaction (advanced))	The Spring transaction manager to use.		PlatformTransactionManager
transactionName (transaction (advanced))	The name of the transaction to use.		String
transactionTimeout (transaction (advanced))	The timeout value of the transaction (in seconds), if using transacted mode.	-1	int

31.4. ENDPOINT OPTIONS

The JMS endpoint is configured using URI syntax:

```
jms:destinationType:destinationName
```

with the following path and query parameters:

31.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
destinationType (common)	The kind of destination to use. Enum values: • queue • topic • temp-queue • temp-topic	queue	String
destinationName (common)	**Required** Name of the queue or topic to use as destination.		String

31.4.2. Query Parameters (95 parameters)

Name	Description	Default	Type
clientId (common)	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String
connectionFactory (common)	The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory
disableReplyTo (common)	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	boolean
durableSubscriptionName (common)	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String

Name	Description	Default	Type
jmsMessageType (common)	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it. Enum values: • Bytes • Map • Object • Stream • Text		JmsMessageType
replyTo (common)	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean
acknowledgementModeName (consumer)	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE. Enum values: • SESSION_TRANSACTED • CLIENT_ACKNOWLEDGE • AUTO_ACKNOWLEDGE • DUPS_OK_ACKNOWLEDGE	AUTO_ACKNOWLEDGE	String

Name	Description	Default	Type
artemisConsumerPriority (consumer)	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		int
asyncConsumer (consumer)	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean
cacheLevel (consumer)	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		int

Name	Description	Default	Type
cacheLevelName (consumer)	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION	CACHE_AUTO	String
concurrentConsumers (consumer)	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	int
maxConcurrentConsumers (consumer)	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		int
replyToDeliveryPersistent (consumer)	Specifies whether to use persistent delivery by default for replies.	true	boolean
selector (consumer)	Sets the JMS selector to use.		String

Name	Description	Default	Type
subscriptionDurable (consumer)	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	boolean
subscriptionName (consumer)	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String
subscriptionShared (consumer)	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	boolean
acceptMessagesWhileStopping (consumer (advanced))	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	boolean

Name	Description	Default	Type
allowReplyManagerQuickStop (consumer (advanced))	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	boolean
consumerType (consumer (advanced))	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use. Enum values: • Simple • Default • Custom	Default	ConsumerType

Name	Description	Default	Type
defaultTaskExecutorType (consumer (advanced))	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers. Enum values: • ThreadPool • SimpleAsync		DefaultTaskExecutorType
eagerLoadingOfProperties (consumer (advanced))	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	boolean
eagerPoisonBody (consumer (advanced))	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
exposeListenerSession (consumer (advanced))	Specifies whether the listener session should be exposed when consuming messages.	false	boolean
replyToSameDestinationAllowed (consumer (advanced))	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	boolean
taskExecutor (consumer (advanced))	Allows you to specify a custom task executor for consuming messages.		TaskExecutor
deliveryDelay (producer)	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	long
deliveryMode (producer)	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2. Enum values: • 1 • 2		Integer
deliveryPersistent (producer)	Specifies whether persistent delivery is used by default.	true	boolean

Name	Description	Default	Type
explicitQosEnabled (producer)	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
formatDateHeadersToIso8601 (producer)	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
preserveMessageQos (producer)	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	boolean

Name	Description	Default	Type
priority (producer)	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect. Enum values: • 1 • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9	4	int
replyToConcurrentConsumers (producer)	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	int
replyToMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		int
replyToOnTimeoutMaxConcurrentConsumers (producer)	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	int
replyToOverride (producer)	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String

Name	Description	Default	Type
replyToType (producer)	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive. Enum values: • Temporary • Shared • Exclusive		ReplyToType
requestTimeout (producer)	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option.	20000	long
timeToLive (producer)	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	long
allowAdditionalHeaders (producer (advanced))	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
allowNullBody (producer (advanced))	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	boolean

Name	Description	Default	Type
alwaysCopyMessage (producer (advanced))	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	boolean
correlationProperty (producer (advanced))	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String
disableTimeToLive (producer (advanced))	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	boolean
forceSendOriginalMessage (producer (advanced))	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	boolean
includeSentJMSMessageID (producer (advanced))	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	boolean

Name	Description	Default	Type
replyToCacheLevelName (producer (advanced))	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION. Enum values: • CACHE_AUTO • CACHE_CONNECTION • CACHE_CONSUMER • CACHE_NONE • CACHE_SESSION		String
replyToDestinationSelectorName (producer (advanced))	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String
streamMessageTypeEnabled (producer (advanced))	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	boolean
allowSerializedHeaders (advanced)	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
artemisStreamingEnabled (advanced)	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	boolean

Name	Description	Default	Type
asyncStartListener (advanced)	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	boolean
asyncStopListener (advanced)	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	boolean
destinationResolver (advanced)	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry).		DestinationResolver
errorHandler (advanced)	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler.		ErrorHandler
exceptionListener (advanced)	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions.		ExceptionListener
headerFilterStrategy (advanced)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
idleConsumerLimit (advanced)	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	int

Name	Description	Default	Type
idleTaskExecutionLimit (advanced)	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	int
includeAllJMSXProperties (advanced)	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	boolean
jmsKeyFormatStrategy (advanced)	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation. Enum values: • default • passthrough		JmsKeyFormatStrategy
mapJmsMessage (advanced)	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	boolean
maxMessagesPerTask (advanced)	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	int
messageConverter (advanced)	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message.		MessageConverter

Name	Description	Default	Type
messageCreatedStrategy (advanced)	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message.		MessageCreatedStrategy
messageIdEnabled (advanced)	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	boolean
messageListenerContainerFactory (advanced)	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom.		MessageListenerContainerFactory
messageTimestampEnabled (advanced)	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	boolean
pubSubNoLocal (advanced)	Specifies whether to inhibit the delivery of messages published by its own connection.	false	boolean
receiveTimeout (advanced)	The timeout for receiving messages (in milliseconds).	1000	long
recoveryInterval (advanced)	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds.	5000	long
requestTimeoutCheckerInterval (advanced)	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout.	1000	long
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean

Name	Description	Default	Type
transferException (advanced)	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	boolean
transferExchange (advanced)	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	boolean
useMessageIDAsCorrelationID (advanced)	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	boolean
waitForProvisionCorrelationToBeUpdatedCounter (advanced)	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	int
waitForProvisionCorrelationToBeUpdatedThreadSleepingTime (advanced)	Interval in millis to sleep each time while waiting for provisional correlation id to be updated.	100	long

Name	Description	Default	Type
errorHandlerLoggingLevel (logging)	Allows to configure the default errorHandler logging level for logging uncaught exceptions. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
errorHandlerLogStackTrace (logging)	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	boolean
password (security)	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
username (security)	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
transacted (transaction)	Specifies whether to use transacted mode.	false	boolean

Name	Description	Default	Type
transactedInOut (transaction)	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	boolean
lazyCreateTransactionManager (transaction (advanced))	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	boolean
transactionManager (transaction (advanced))	The Spring transaction manager to use.		PlatformTransactionManager
transactionName (transaction (advanced))	The name of the transaction to use.		String
transactionTimeout (transaction (advanced))	The timeout value of the transaction (in seconds), if using transacted mode.	-1	int

31.5. SAMPLES

JMS is used in many examples for other components as well. But we provide a few samples below to get started.

31.5.1. Receiving from JMS

In the following sample we configure a route that receives JMS messages and routes the message to a POJO:

```
from("jms:queue:foo").
  to("bean:myBusinessLogic");
```

You can of course use any of the EIP patterns so the route can be context based. For example, here's how to filter an order topic for the big spenders:

```
from("jms:topic:OrdersTopic").
  filter().method("myBean", "isGoldCustomer").
  to("jms:queue:BigSpendersQueue");
```

31.5.2. Sending to JMS

In the sample below we poll a file folder and send the file content to a JMS topic. As we want the content of the file as a **TextMessage** instead of a **BytesMessage**, we need to convert the body to a **String**:

```
from("file://orders").
  convertBodyTo(String.class).
  to("jms:topic:OrdersTopic");
```

31.5.3. Using Annotations

Camel also has annotations so you can use POJO Consuming and POJO Producing.

31.5.4. Spring DSL sample

The preceding examples use the Java DSL. Camel also supports Spring XML DSL. Here is the big spender sample using Spring DSL:

```
<route>
  <from uri="jms:topic:OrdersTopic"/>
  <filter>
    <method ref="myBean" method="isGoldCustomer"/>
    <to uri="jms:queue:BigSpendersQueue"/>
  </filter>
</route>
```

31.5.5. Other samples

JMS appears in many of the examples for other components and EIP patterns, as well in this Camel documentation. So feel free to browse the documentation.

31.5.6. Using JMS as a Dead Letter Queue storing Exchange

Normally, when using JMS as the transport, it only transfers the body and headers as the payload. If you want to use JMS with a Dead Letter Channel, using a JMS queue as the Dead Letter Queue, then normally the caused Exception is not stored in the JMS message. You can, however, use the **transferExchange** option on the JMS dead letter queue to instruct Camel to store the entire Exchange in the queue as a **javax.jms.ObjectMessage** that holds a

org.apache.camel.support.DefaultExchangeHolder. This allows you to consume from the Dead Letter Queue and retrieve the caused exception from the Exchange property with the key **Exchange.EXCEPTION_CAUGHT**. The demo below illustrates this:

```
// setup error handler to use JMS as queue and store the entire Exchange
errorHandler(deadLetterChannel("jms:queue:dead?transferExchange=true"));
```

Then you can consume from the JMS queue and analyze the problem:

```
from("jms:queue:dead").to("bean:myErrorAnalyzer");

// and in our bean
String body = exchange.getIn().getBody();
Exception cause = exchange.getProperty(Exchange.EXCEPTION_CAUGHT, Exception.class);
// the cause message is
String problem = cause.getMessage();
```

31.5.7. Using JMS as a Dead Letter Channel storing error only

You can use JMS to store the cause error message or to store a custom body, which you can initialize yourself. The following example uses the Message Translator EIP to do a transformation on the failed exchange before it is moved to the JMS dead letter queue:

```
// we sent it to a seda dead queue first
errorHandler(deadLetterChannel("seda:dead"));

// and on the seda dead queue we can do the custom transformation before its sent to the JMS queue
from("seda:dead").transform(exceptionMessage()).to("jms:queue:dead");
```

Here we only store the original cause error message in the transform. You can, however, use any Expression to send whatever you like. For example, you can invoke a method on a Bean or use a custom processor.

31.6. MESSAGE MAPPING BETWEEN JMS AND CAMEL

Camel automatically maps messages between **javax.jms.Message** and **org.apache.camel.Message**.

When sending a JMS message, Camel converts the message body to the following JMS message types:

Body Type	JMS Message	Comment
String	**javax.jms.TextMessage**	
org.w3c.dom.Node	**javax.jms.TextMessage**	The DOM will be converted to **String**.
Map	**javax.jms.MapMessage**	
java.io.Serializable	**javax.jms.ObjectMessage**	
byte[]	**javax.jms.BytesMessage**	

Body Type	JMS Message	Comment
java.io.File	**javax.jms.BytesMessage**	
java.io.Reader	**javax.jms.BytesMessage**	
java.io.InputStream	**javax.jms.BytesMessage**	
java.nio.ByteBuffer	**javax.jms.BytesMessage**	

When receiving a JMS message, Camel converts the JMS message to the following body type:

JMS Message	Body Type
javax.jms.TextMessage	**String**
javax.jms.BytesMessage	**byte[]**
javax.jms.MapMessage	**Map<String, Object>**
javax.jms.ObjectMessage	**Object**

31.6.1. Disabling auto-mapping of JMS messages

You can use the **mapJmsMessage** option to disable the auto-mapping above. If disabled, Camel will not try to map the received JMS message, but instead uses it directly as the payload. This allows you to avoid the overhead of mapping and let Camel just pass through the JMS message. For instance, it even allows you to route **javax.jms.ObjectMessage** JMS messages with classes you do **not** have on the classpath.

31.6.2. Using a custom MessageConverter

You can use the **messageConverter** option to do the mapping yourself in a Spring **org.springframework.jms.support.converter.MessageConverter** class.

For example, in the route below we use a custom message converter when sending a message to the JMS order queue:

```
from("file://inbox/order").to("jms:queue:order?messageConverter=#myMessageConverter");
```

You can also use a custom message converter when consuming from a JMS destination.

31.6.3. Controlling the mapping strategy selected

You can use the **jmsMessageType** option on the endpoint URL to force a specific message type for all messages.

In the route below, we poll files from a folder and send them as **javax.jms.TextMessage** as we have forced the JMS producer endpoint to use text messages:

```
from("file://inbox/order").to("jms:queue:order?jmsMessageType=Text");
```

You can also specify the message type to use for each message by setting the header with the key **CamelJmsMessageType**. For example:

```
from("file://inbox/order").setHeader("CamelJmsMessageType",
JmsMessageType.Text).to("jms:queue:order");
```

The possible values are defined in the **enum** class, **org.apache.camel.jms.JmsMessageType**.

31.7. MESSAGE FORMAT WHEN SENDING

The exchange that is sent over the JMS wire must conform to the JMS Message spec.

For the **exchange.in.header** the following rules apply for the header **keys**:

- Keys starting with **JMS** or **JMSX** are reserved.
- **exchange.in.headers** keys must be literals and all be valid Java identifiers (do not use dots in the key name).
- Camel replaces dots & hyphens and the reverse when when consuming JMS messages:
  . is replaced by *`DOT`* and the reverse replacement when Camel consumes the message.
  - is replaced by *`HYPHEN`* and the reverse replacement when Camel consumes the message.
- See also the option **jmsKeyFormatStrategy**, which allows use of your own custom strategy for formatting keys.

For the **exchange.in.header**, the following rules apply for the header **values**:

- The values must be primitives or their counter objects (such as **Integer**, **Long**, **Character**). The types, **String**, **CharSequence**, **Date**, **BigDecimal** and **BigInteger** are all converted to their **toString()** representation. All other types are dropped.

Camel will log with category **org.apache.camel.component.jms.JmsBinding** at **DEBUG** level if it drops a given header value. For example:

```
2008-07-09 06:43:04,046 [main           ] DEBUG JmsBinding
  - Ignoring non primitive header: order of class:
org.apache.camel.component.jms.issues.DummyOrder with value: DummyOrder{orderId=333,
itemId=4444, quantity=2}
```

31.8. MESSAGE FORMAT WHEN RECEIVING

Camel adds the following properties to the **Exchange** when it receives a message:

Property	Type	Description
org.apache.camel.jms.replyDestination	**javax.jms.Destination**	The reply destination.

Camel adds the following JMS properties to the In message headers when it receives a JMS message:

Header	Type	Description
JMSCorrelationID	**String**	The JMS correlation ID.
JMSDeliveryMode	**int**	The JMS delivery mode.
JMSDestination	**javax.jms.Destination**	The JMS destination.
JMSExpiration	**long**	The JMS expiration.
JMSMessageID	**String**	The JMS unique message ID.
JMSPriority	**int**	The JMS priority (with 0 as the lowest priority and 9 as the highest).
JMSRedelivered	**boolean**	Is the JMS message redelivered.
JMSReplyTo	**javax.jms.Destination**	The JMS reply-to destination.
JMSTimestamp	**long**	The JMS timestamp.
JMSType	**String**	The JMS type.
JMSXGroupID	**String**	The JMS group ID.

As all the above information is standard JMS you can check the JMS documentation for further details.

31.9. ABOUT USING CAMEL TO SEND AND RECEIVE MESSAGES AND JMSREPLYTO

The JMS component is complex and you have to pay close attention to how it works in some cases. So this is a short summary of some of the areas/pitfalls to look for.

When Camel sends a message using its **JMSProducer**, it checks the following conditions:

- The message exchange pattern,
- Whether a **JMSReplyTo** was set in the endpoint or in the message headers,
- Whether any of the following options have been set on the JMS endpoint: **disableReplyTo**, **preserveMessageQos**, **explicitQosEnabled**.

All this can be a tad complex to understand and configure to support your use case.

31.9.1. JmsProducer

The **JmsProducer** behaves as follows, depending on configuration:

Exchange Pattern	Other options	Description
InOut	-	Camel will expect a reply, set a temporary **JMSReplyTo**, and after sending the message, it will start to listen for the reply message on the temporary queue.
InOut	**JMSReplyTo** is set	Camel will expect a reply and, after sending the message, it will start to listen for the reply message on the specified **JMSReplyTo** queue.
InOnly	-	Camel will send the message and **not** expect a reply.
InOnly	**JMSReplyTo** is set	By default, Camel discards the **JMSReplyTo** destination and clears the **JMSReplyTo** header before sending the message. Camel then sends the message and does **not** expect a reply. Camel logs this in the log at **WARN** level (changed to **DEBUG** level from **Camel 2.6** onwards. You can use **preserveMessageQuo=true** to instruct Camel to keep the **JMSReplyTo**. In all situations the **JmsProducer** does **not** expect any reply and thus continue after sending the message.

31.9.2. JmsConsumer

The **JmsConsumer** behaves as follows, depending on configuration:

Exchange Pattern	Other options	Description
InOut	-	Camel will send the reply back to the **JMSReplyTo** queue.
InOnly	-	Camel will not send a reply back, as the pattern is *InOnly*.
-	**disableReplyTo =true**	This option suppresses replies.

So pay attention to the message exchange pattern set on your exchanges.

If you send a message to a JMS destination in the middle of your route you can specify the exchange pattern to use, see more at Request Reply.
This is useful if you want to send an **InOnly** message to a JMS topic:

```
from("activemq:queue:in")
  .to("bean:validateOrder")
  .to(ExchangePattern.InOnly, "activemq:topic:order")
  .to("bean:handleOrder");
```

31.10. REUSE ENDPOINT AND SEND TO DIFFERENT DESTINATIONS COMPUTED AT RUNTIME

If you need to send messages to a lot of different JMS destinations, it makes sense to reuse a JMS endpoint and specify the real destination in a message header. This allows Camel to reuse the same endpoint, but send to different destinations. This greatly reduces the number of endpoints created and economizes on memory and thread resources.

You can specify the destination in the following headers:

Header	Type	Description
CamelJmsDestination	**javax.jms.Destination**	A destination object.
CamelJmsDestinationName	**String**	The destination name.

For example, the following route shows how you can compute a destination at run time and use it to override the destination appearing in the JMS URL:

```
from("file://inbox")
  .to("bean:computeDestination")
  .to("activemq:queue:dummy");
```

The queue name, **dummy**, is just a placeholder. It must be provided as part of the JMS endpoint URL, but it will be ignored in this example.

In the **computeDestination** bean, specify the real destination by setting the **CamelJmsDestinationName** header as follows:

```
public void setJmsHeader(Exchange exchange) {
   String id = ....
   exchange.getIn().setHeader("CamelJmsDestinationName", "order:" + id");
}
```

Then Camel will read this header and use it as the destination instead of the one configured on the endpoint. So, in this example Camel sends the message to **activemq:queue:order:2**, assuming the **id** value was 2.

If both the **CamelJmsDestination** and the **CamelJmsDestinationName** headers are set, **CamelJmsDestination** takes priority. Keep in mind that the JMS producer removes both **CamelJmsDestination** and **CamelJmsDestinationName** headers from the exchange and do not propagate them to the created JMS message in order to avoid the accidental loops in the routes (in scenarios when the message will be forwarded to the another JMS endpoint).

31.11. CONFIGURING DIFFERENT JMS PROVIDERS

You can configure your JMS provider in Spring XML as follows:

Basically, you can configure as many JMS component instances as you wish and give them **a unique name using the id attribute**. The preceding example configures an **activemq** component. You could do the same to configure MQSeries, TibCo, BEA, Sonic and so on.

Once you have a named JMS component, you can then refer to endpoints within that component using URIs. For example for the component name, **activemq**, you can then refer to destinations using the URI format, **activemq:[queue:|topic:]destinationName**. You can use the same approach for all other JMS providers.

This works by the SpringCamelContext lazily fetching components from the spring context for the scheme name you use for Endpoint URIs and having the Component resolve the endpoint URIs.

31.11.1. Using JNDI to find the ConnectionFactory

If you are using a J2EE container, you might need to look up JNDI to find the JMS **ConnectionFactory** rather than use the usual **<bean>** mechanism in Spring. You can do this using Spring's factory bean or the new Spring XML namespace. For example:

```
<bean id="weblogic" class="org.apache.camel.component.jms.JmsComponent">
  <property name="connectionFactory" ref="myConnectionFactory"/>
</bean>

<jee:jndi-lookup id="myConnectionFactory" jndi-name="jms/connectionFactory"/>
```

See The jee schema in the Spring reference documentation for more details about JNDI lookup.

31.12. CONCURRENT CONSUMING

A common requirement with JMS is to consume messages concurrently in multiple threads in order to make an application more responsive. You can set the **concurrentConsumers** option to specify the number of threads servicing the JMS endpoint, as follows:

```
from("jms:SomeQueue?concurrentConsumers=20").
  bean(MyClass.class);
```

You can configure this option in one of the following ways:

- On the **JmsComponent**,
- On the endpoint URI or,
- By invoking **setConcurrentConsumers()** directly on the **JmsEndpoint**.

31.12.1. Concurrent Consuming with async consumer

Notice that each concurrent consumer will only pickup the next available message from the JMS broker, when the current message has been fully processed. You can set the option **asyncConsumer=true** to let the consumer pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). See more details in the table on top of the page about the **asyncConsumer** option.

```
from("jms:SomeQueue?concurrentConsumers=20&asyncConsumer=true").
  bean(MyClass.class);
```

31.13. REQUEST-REPLY OVER JMS

Camel supports Request Reply over JMS. In essence the MEP of the Exchange should be **InOut** when you send a message to a JMS queue.

Camel offers a number of options to configure request/reply over JMS that influence performance and clustered environments. The table below summaries the options.

Option	Performance	Cluster	Description
Temporary	Fast	Yes	A temporary queue is used as reply queue, and automatic created by Camel. To use this do **not** specify a replyTo queue name. And you can optionally configure **replyToType=Temporary** to make it stand out that temporary queues are in use.
Shared	Slow	Yes	A shared persistent queue is used as reply queue. The queue must be created beforehand, although some brokers can create them on the fly such as Apache ActiveMQ. To use this you must specify the replyTo queue name. And you can optionally configure **replyToType=Shared** to make it stand out that shared queues are in use. A shared queue can be used in a clustered environment with multiple nodes running this Camel application at the same time. All using the same shared reply queue. This is possible because JMS Message selectors are used to correlate expected reply messages; this impacts performance though. JMS Message selectors is slower, and therefore not as fast as **Temporary** or **Exclusive** queues. See further below how to tweak this for better performance.

Option	Performance	Cluster	Description
Exclusive	Fast	No (*Yes)	An exclusive persistent queue is used as reply queue. The queue must be created beforehand, although some brokers can create them on the fly such as Apache ActiveMQ. To use this you must specify the replyTo queue name. And you **must** configure **replyToType=Exclusive** to instruct Camel to use exclusive queues, as **Shared** is used by default, if a **replyTo** queue name was configured. When using exclusive reply queues, then JMS Message selectors are **not** in use, and therefore other applications must not use this queue as well. An exclusive queue **cannot** be used in a clustered environment with multiple nodes running this Camel application at the same time; as we do not have control if the reply queue comes back to the same node that sent the request message; that is why shared queues use JMS Message selectors to make sure of this. **Though** if you configure each Exclusive reply queue with an unique name per node, then you can run this in a clustered environment. As then the reply message will be sent back to that queue for the given node, that awaits the reply message.
concurrentConsumers	Fast	Yes	Allows to process reply messages concurrently using concurrent message listeners in use. You can specify a range using the **concurrentConsumers** and **maxConcurrentConsumers** options. **Notice:** That using **Shared** reply queues may not work as well with concurrent listeners, so use this option with care.
maxConcurrentConsumers	Fast	Yes	Allows to process reply messages concurrently using concurrent message listeners in use. You can specify a range using the **concurrentConsumers** and **maxConcurrentConsumers** options. **Notice:** That using **Shared** reply queues may not work as well with concurrent listeners, so use this option with care.

The **JmsProducer** detects the **InOut** and provides a **JMSReplyTo** header with the reply destination to be used. By default Camel uses a temporary queue, but you can use the **replyTo** option on the endpoint to specify a fixed reply queue (see more below about fixed reply queue).

Camel will automatically setup a consumer which listen on the reply queue, so you should **not** do anything.
This consumer is a Spring **DefaultMessageListenerContainer** which listen for replies. However it's fixed to 1 concurrent consumer.
That means replies will be processed in sequence as there are only 1 thread to process the replies. You can configure the listener to use concurrent threads using the **concurrentConsumers** and **maxConcurrentConsumers** options. This allows you to easier configure this in Camel as shown below:

```
from(xxx)
.inOut().to("activemq:queue:foo?concurrentConsumers=5")
.to(yyy)
.to(zzz);
```

In this route we instruct Camel to route replies asynchronously using a thread pool with 5 threads.

31.13.1. Request-reply over JMS and using a shared fixed reply queue

If you use a fixed reply queue when doing Request Reply over JMS as shown in the example below, then pay attention.

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar")
.to(yyy)
```

In this example the fixed reply queue named "bar" is used. By default Camel assumes the queue is shared when using fixed reply queues, and therefore it uses a **JMSSelector** to only pickup the expected reply messages (eg based on the **JMSCorrelationID**). See next section for exclusive fixed reply queues. That means its not as fast as temporary queues. You can speedup how often Camel will pull for reply messages using the **receiveTimeout** option. By default its 1000 millis. So to make it faster you can set it to 250 millis to pull 4 times per second as shown:

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&receiveTimeout=250")
.to(yyy)
```

Notice this will cause the Camel to send pull requests to the message broker more frequent, and thus require more network traffic.
It is generally recommended to use temporary queues if possible.

31.13.2. Request-reply over JMS and using an exclusive fixed reply queue

In the previous example, Camel would anticipate the fixed reply queue named "bar" was shared, and thus it uses a **JMSSelector** to only consume reply messages which it expects. However there is a drawback doing this as the JMS selector is slower. Also the consumer on the reply queue is slower to update with new JMS selector ids. In fact it only updates when the **receiveTimeout** option times out, which by default is 1 second. So in theory the reply messages could take up till about 1 sec to be detected. On the other hand if the fixed reply queue is exclusive to the Camel reply consumer, then we can avoid using the JMS selectors, and thus be more performant. In fact as fast as using temporary queues. There is the **ReplyToType** option which you can configure to **Exclusive**
to tell Camel that the reply queue is exclusive as shown in the example below:

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)
```

Mind that the queue must be exclusive to each and every endpoint. So if you have two routes, then they each need an unique reply queue as shown in the next example:

```
from(xxx)
.inOut().to("activemq:queue:foo?replyTo=bar&replyToType=Exclusive")
.to(yyy)
```

```
from(aaa)
.inOut().to("activemq:queue:order?replyTo=order.reply&replyToType=Exclusive")
.to(bbb)
```

The same applies if you run in a clustered environment. Then each node in the cluster must use an unique reply queue name. As otherwise each node in the cluster may pickup messages which was intended as a reply on another node. For clustered environments its recommended to use shared reply queues instead.

31.14. SYNCHRONIZING CLOCKS BETWEEN SENDERS AND RECEIVERS

When doing messaging between systems, its desirable that the systems have synchronized clocks. For example when sending a JMS message, then you can set a time to live value on the message. Then the receiver can inspect this value, and determine if the message is already expired, and thus drop the message instead of consume and process it. However this requires that both sender and receiver have synchronized clocks. If you are using ActiveMQ then you can use the timestamp plugin to synchronize clocks.

31.15. ABOUT TIME TO LIVE

Read first above about synchronized clocks.

When you do request/reply (InOut) over JMS with Camel then Camel uses a timeout on the sender side, which is default 20 seconds from the **requestTimeout** option. You can control this by setting a higher/lower value. However the time to live value is still set on the message being send. So that requires the clocks to be synchronized between the systems. If they are not, then you may want to disable the time to live value being set. This is now possible using the **disableTimeToLive** option from **Camel 2.8** onwards. So if you set this option to **disableTimeToLive=true**, then Camel does **not** set any time to live value when sending JMS messages. **But** the request timeout is still active. So for example if you do request/reply over JMS and have disabled time to live, then Camel will still use a timeout by 20 seconds (the **requestTimeout** option). That option can of course also be configured. So the two options **requestTimeout** and **disableTimeToLive** gives you fine grained control when doing request/reply.

You can provide a header in the message to override and use as the request timeout value instead of the endpoint configured value. For example:

```
from("direct:someWhere")
  .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
  .to("bean:processReply");
```

In the route above we have a endpoint configured **requestTimeout** of 30 seconds. So Camel will wait up till 30 seconds for that reply message to come back on the bar queue. If no reply message is received then a **org.apache.camel.ExchangeTimedOutException** is set on the Exchange and Camel continues routing the message, which would then fail due the exception, and Camel's error handler reacts.

If you want to use a per message timeout value, you can set the header with key **org.apache.camel.component.jms.JmsConstants#JMS_REQUEST_TIMEOUT** which has constant value **"CamelJmsRequestTimeout"** with a timeout value as long type.

For example we can use a bean to compute the timeout value per individual message, such as calling the **"whatIsTheTimeout"** method on the service bean as shown below:

```
from("direct:someWhere")
  .setHeader("CamelJmsRequestTimeout", method(ServiceBean.class, "whatIsTheTimeout"))
```

```
    .to("jms:queue:foo?replyTo=bar&requestTimeout=30s")
    .to("bean:processReply");
```

When you do fire and forget (InOut) over JMS with Camel then Camel by default does **not** set any time to live value on the message. You can configure a value by using the **timeToLive** option. For example to indicate a 5 sec., you set **timeToLive=5000**. The option **disableTimeToLive** can be used to force disabling the time to live, also for InOnly messaging. The **requestTimeout** option is not being used for InOnly messaging.

31.16. ENABLING TRANSACTED CONSUMPTION

A common requirement is to consume from a queue in a transaction and then process the message using the Camel route. To do this, just ensure that you set the following properties on the component/endpoint:

- **transacted** = true
- **transactionManager** = a *Transsaction Manager* - typically the **JmsTransactionManager**

See the Transactional Client EIP pattern for further details.

Transactions and [Request Reply] over JMS

When using Request Reply over JMS you cannot use a single transaction; JMS will not send any messages until a commit is performed, so the server side won't receive anything at all until the transaction commits. Therefore to use Request Reply you must commit a transaction after sending the request and then use a separate transaction for receiving the response.

To address this issue the JMS component uses different properties to specify transaction use for oneway messaging and request reply messaging:

The **transacted** property applies **only** to the InOnly message Exchange Pattern (MEP).

You can leverage the DMLC transacted session API using the following properties on component/endpoint:

- **transacted** = true
- **lazyCreateTransactionManager** = false

The benefit of doing so is that the cacheLevel setting will be honored when using local transactions without a configured TransactionManager. When a TransactionManager is configured, no caching happens at DMLC level and it is necessary to rely on a pooled connection factory. For more details about this kind of setup, see here and here.

31.17. USING JMSREPLYTO FOR LATE REPLIES

When using Camel as a JMS listener, it sets an Exchange property with the value of the ReplyTo **javax.jms.Destination** object, having the key **ReplyTo**. You can obtain this **Destination** as follows:

```
Destination replyDestination =
exchange.getIn().getHeader(JmsConstants.JMS_REPLY_DESTINATION, Destination.class);
```

And then later use it to send a reply using regular JMS or Camel.

```
// we need to pass in the JMS component, and in this sample we use ActiveMQ
JmsEndpoint endpoint = JmsEndpoint.newInstance(replyDestination, activeMQComponent);
// now we have the endpoint we can use regular Camel API to send a message to it
template.sendBody(endpoint, "Here is the late reply.");
```

A different solution to sending a reply is to provide the **replyDestination** object in the same Exchange property when sending. Camel will then pick up this property and use it for the real destination. The endpoint URI must include a dummy destination, however. For example:

```
// we pretend to send it to some non existing dummy queue
template.send("activemq:queue:dummy, new Processor() {
  public void process(Exchange exchange) throws Exception {
    // and here we override the destination with the ReplyTo destination object so the message is sent
to there instead of dummy
    exchange.getIn().setHeader(JmsConstants.JMS_DESTINATION, replyDestination);
    exchange.getIn().setBody("Here is the late reply.");
  }
}
```

31.18. USING A REQUEST TIMEOUT

In the sample below we send a Request Reply style message Exchange (we use the **requestBody** method = **InOut**) to the slow queue for further processing in Camel and we wait for a return reply:

31.19. SENDING AN INONLY MESSAGE AND KEEPING THE JMSREPLYTO HEADER

When sending to a JMS destination using **camel-jms** the producer will use the MEP to detect if its *InOnly* or *InOut* messaging. However there can be times where you want to send an *InOnly* message but keeping the **JMSReplyTo** header. To do so you have to instruct Camel to keep it, otherwise the **JMSReplyTo** header will be dropped.

For example to send an *InOnly* message to the foo queue, but with a **JMSReplyTo** with bar queue you can do as follows:

```
template.send("activemq:queue:foo?preserveMessageQos=true", new Processor() {
  public void process(Exchange exchange) throws Exception {
    exchange.getIn().setBody("World");
    exchange.getIn().setHeader("JMSReplyTo", "bar");
  }
});
```

Notice we use **preserveMessageQos=true** to instruct Camel to keep the **JMSReplyTo** header.

31.20. SETTING JMS PROVIDER OPTIONS ON THE DESTINATION

Some JMS providers, like IBM's WebSphere MQ need options to be set on the JMS destination. For example, you may need to specify the **targetClient** option. Since **targetClient** is a WebSphere MQ option and not a Camel URI option, you need to set that on the JMS destination name like so:

```
// ...
.setHeader("CamelJmsDestinationName", constant("queue:///MY_QUEUE?targetClient=1"))
.to("wmq:queue:MY_QUEUE?useMessageIDAsCorrelationID=true");
```

Some versions of WMQ won't accept this option on the destination name and you will get an exception like:

```
com.ibm.msg.client.jms.DetailedJMSException: JMSCC0005: The specified
value 'MY_QUEUE?targetClient=1' is not allowed for
'XMSC_DESTINATION_NAME'
```

A workaround is to use a custom DestinationResolver:

```
JmsComponent wmq = new JmsComponent(connectionFactory);

wmq.setDestinationResolver(new DestinationResolver() {
    public Destination resolveDestinationName(Session session, String destinationName, boolean
pubSubDomain) throws JMSException {
        MQQueueSession wmqSession = (MQQueueSession) session;
        return wmqSession.createQueue("queue:///" + destinationName + "?targetClient=1");
    }
});
```

31.21. SPRING BOOT AUTO-CONFIGURATION

When using jms with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jms-starter</artifactId>
</dependency>
```

The component supports 99 options, which are listed below.

Name	Description	Default	Type
camel.component.jms.accept-messages-while-stopping	Specifies whether the consumer accept messages while it is stopping. You may consider enabling this option, if you start and stop JMS routes at runtime, while there are still messages enqueued on the queue. If this option is false, and you stop the JMS route, then messages may be rejected, and the JMS broker would have to attempt redeliveries, which yet again may be rejected, and eventually the message may be moved at a dead letter queue on the JMS broker. To avoid this its recommended to enable this option.	false	Boolean

Name	Description	Default	Type
camel.component.jms.acknowledgement-mode-name	The JMS acknowledgement name, which is one of: SESSION_TRANSACTED, CLIENT_ACKNOWLEDGE, AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE.	AUTO_ACKNOWLEDGE	String
camel.component.jms.allow-additional-headers	This option is used to allow additional headers which may have values that are invalid according to JMS specification. For example some message systems such as WMQ do this with header names using prefix JMS_IBM_MQMD_ containing values with byte array or other invalid types. You can specify multiple header names separated by comma, and use as suffix for wildcard matching.		String
camel.component.jms.allow-auto-wired-connection-factory	Whether to auto-discover ConnectionFactory from the registry, if no connection factory has been configured. If only one instance of ConnectionFactory is found then it will be used. This is enabled by default.	true	Boolean
camel.component.jms.allow-auto-wired-destination-resolver	Whether to auto-discover DestinationResolver from the registry, if no destination resolver has been configured. If only one instance of DestinationResolver is found then it will be used. This is enabled by default.	true	Boolean
camel.component.jms.allow-null-body	Whether to allow sending messages with no body. If this option is false and the message body is null, then an JMSException is thrown.	true	Boolean
camel.component.jms.allow-reply-manager-quick-stop	Whether the DefaultMessageListenerContainer used in the reply managers for request-reply messaging allow the DefaultMessageListenerContainer.runningAllowed flag to quick stop in case JmsConfiguration#isAcceptMessagesWhileStopping is enabled, and org.apache.camel.CamelContext is currently being stopped. This quick stop ability is enabled by default in the regular JMS consumers but to enable for reply managers you must enable this flag.	false	Boolean
camel.component.jms.allow-serialized-headers	Controls whether or not to include serialized headers. Applies only when transferExchange is true. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	Boolean

Name	Description	Default	Type
camel.component.jms.always-copy-message	If true, Camel will always make a JMS message copy of the message when it is passed to the producer for sending. Copying the message is needed in some situations, such as when a replyToDestinationSelectorName is set (incidentally, Camel will set the alwaysCopyMessage option to true, if a replyToDestinationSelectorName is set).	false	Boolean
camel.component.jms.artemis-consumer-priority	Consumer priorities allow you to ensure that high priority consumers receive messages while they are active. Normally, active consumers connected to a queue receive messages from it in a round-robin fashion. When consumer priorities are in use, messages are delivered round-robin if multiple active consumers exist with the same high priority. Messages will only going to lower priority consumers when the high priority consumers do not have credit available to consume the message, or those high priority consumers have declined to accept the message (for instance because it does not meet the criteria of any selectors associated with the consumer).		Integer
camel.component.jms.artemis-streaming-enabled	Whether optimizing for Apache Artemis streaming mode. This can reduce memory overhead when using Artemis with JMS StreamMessage types. This option must only be enabled if Apache Artemis is being used.	false	Boolean
camel.component.jms.async-consumer	Whether the JmsConsumer processes the Exchange asynchronously. If enabled then the JmsConsumer may pickup the next message from the JMS queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the JmsConsumer will pickup the next message from the JMS queue. Note if transacted has been enabled, then asyncConsumer=true does not run asynchronously, as transaction must be executed synchronously (Camel 3.0 may support async transactions).	false	Boolean

Name	Description	Default	Type
camel.component.jms.async-start-listener	Whether to startup the JmsConsumer message listener asynchronously, when starting a route. For example if a JmsConsumer cannot get a connection to a remote JMS broker, then it may block while retrying and/or failover. This will cause Camel to block while starting routes. By setting this option to true, you will let routes startup, while the JmsConsumer connects to the JMS broker using a dedicated thread in asynchronous mode. If this option is used, then beware that if the connection could not be established, then an exception is logged at WARN level, and the consumer will not be able to receive messages; You can then restart the route to retry.	false	Boolean
camel.component.jms.async-stop-listener	Whether to stop the JmsConsumer message listener asynchronously, when stopping a route.	false	Boolean
camel.component.jms.auto-startup	Specifies whether the consumer container should auto-startup.	true	Boolean
camel.component.jms.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.jms.cache-level	Sets the cache level by ID for the underlying JMS resources. See cacheLevelName option for more details.		Integer
camel.component.jms.cache-level-name	Sets the cache level by name for the underlying JMS resources. Possible values are: CACHE_AUTO, CACHE_CONNECTION, CACHE_CONSUMER, CACHE_NONE, and CACHE_SESSION. The default setting is CACHE_AUTO. See the Spring documentation and Transactions Cache Levels for more information.	CACHE_AUTO	String
camel.component.jms.client-id	Sets the JMS client ID to use. Note that this value, if specified, must be unique and can only be used by a single JMS connection instance. It is typically only required for durable topic subscriptions. If using Apache ActiveMQ you may prefer to use Virtual Topics instead.		String

Name	Description	Default	Type
camel.component.jms.concurrent-consumers	Specifies the default number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.	1	Integer
camel.component.jms.configuration	To use a shared JMS configuration. The option is a org.apache.camel.component.jms.JmsConfiguration type.		JmsConfiguration
camel.component.jms.connection-factory	The connection factory to be use. A connection factory must be configured either on the component or endpoint. The option is a javax.jms.ConnectionFactory type.		ConnectionFactory
camel.component.jms.consumer-type	The consumer type to use, which can be one of: Simple, Default, or Custom. The consumer type determines which Spring JMS listener to use. Default will use org.springframework.jms.listener.DefaultMessageListenerContainer, Simple will use org.springframework.jms.listener.SimpleMessageListenerContainer. When Custom is specified, the MessageListenerContainerFactory defined by the messageListenerContainerFactory option will determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use.		ConsumerType
camel.component.jms.correlation-property	When using InOut exchange pattern use this JMS property instead of JMSCorrelationID JMS property to correlate messages. If set messages will be correlated solely on the value of this property JMSCorrelationID property will be ignored and not set by Camel.		String

Name	Description	Default	Type
camel.component.jms.default-task-executor-type	Specifies what default TaskExecutor type to use in the DefaultMessageListenerContainer, for both consumer endpoints and the ReplyTo consumer of producer endpoints. Possible values: SimpleAsync (uses Spring's SimpleAsyncTaskExecutor) or ThreadPool (uses Spring's ThreadPoolTaskExecutor with optimal values - cached threadpool-like). If not set, it defaults to the previous behaviour, which uses a cached thread pool for consumer endpoints and SimpleAsync for reply consumers. The use of ThreadPool is recommended to reduce thread trash in elastic configurations with dynamically increasing and decreasing concurrent consumers.		DefaultTaskExecutorType
camel.component.jms.delivery-delay	Sets delivery delay to use for send calls for JMS. This option requires JMS 2.0 compliant broker.	-1	Long
camel.component.jms.delivery-mode	Specifies the delivery mode to be used. Possible values are those defined by javax.jms.DeliveryMode. NON_PERSISTENT = 1 and PERSISTENT = 2.		Integer
camel.component.jms.delivery-persistent	Specifies whether persistent delivery is used by default.	true	Boolean
camel.component.jms.destination-resolver	A pluggable org.springframework.jms.support.destination.DestinationResolver that allows you to use your own resolver (for example, to lookup the real destination in a JNDI registry). The option is a org.springframework.jms.support.destination.DestinationResolver type.		DestinationResolver
camel.component.jms.disable-reply-to	Specifies whether Camel ignores the JMSReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the JMSReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	Boolean

Name	Description	Default	Type
camel.component.jms.disable-time-to-live	Use this option to force disabling time to live. For example when you do request/reply over JMS, then Camel will by default use the requestTimeout value as time to live on the message being sent. The problem is that the sender and receiver systems have to have their clocks synchronized, so they are in sync. This is not always so easy to archive. So you can use disableTimeToLive=true to not set a time to live value on the sent message. Then the message will not expire on the receiver system. See below in section About time to live for more details.	false	Boolean
camel.component.jms.durable-subscription-name	The durable subscriber name for specifying durable topic subscriptions. The clientId option must be configured as well.		String
camel.component.jms.eager-loading-of-properties	Enables eager loading of JMS properties and payload as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties. See also the option eagerPoisonBody.	false	Boolean
camel.component.jms.eager-poison-body	If eagerLoadingOfProperties is enabled and the JMS message payload (JMS body or JMS properties) is poison (cannot be read/mapped), then set this text as the message body instead so the message can be processed (the cause of the poison are already stored as exception on the Exchange). This can be turned off by setting eagerPoisonBody=false. See also the option eagerLoadingOfProperties.	Poison JMS message due to $\{exception.message}	String
camel.component.jms.enabled	Whether to enable auto configuration of the jms component. This is enabled by default.		Boolean
camel.component.jms.error-handler	Specifies a org.springframework.util.ErrorHandler to be invoked in case of any uncaught exceptions thrown while processing a Message. By default these exceptions will be logged at the WARN level, if no errorHandler has been configured. You can configure logging level and whether stack traces should be logged using errorHandlerLoggingLevel and errorHandlerLogStackTrace options. This makes it much easier to configure, than having to code a custom errorHandler. The option is a org.springframework.util.ErrorHandler type.		ErrorHandler

Name	Description	Default	Type
camel.component.jms.error-handler-log-stack-trace	Allows to control whether stacktraces should be logged or not, by the default errorHandler.	true	Boolean
camel.component.jms.error-handler-logging-level	Allows to configure the default errorHandler logging level for logging uncaught exceptions.		LoggingLevel
camel.component.jms.exception-listener	Specifies the JMS Exception Listener that is to be notified of any underlying JMS exceptions. The option is a javax.jms.ExceptionListener type.		ExceptionListener
camel.component.jms.explicit-qos-enabled	Set if the deliveryMode, priority or timeToLive qualities of service should be used when sending messages. This option is based on Spring's JmsTemplate. The deliveryMode, priority and timeToLive options are applied to the current endpoint. This contrasts with the preserveMessageQos option, which operates at message granularity, reading QoS properties exclusively from the Camel In message headers.	false	Boolean
camel.component.jms.expose-listener-session	Specifies whether the listener session should be exposed when consuming messages.	false	Boolean
camel.component.jms.force-send-original-message	When using mapJmsMessage=false Camel will create a new JMS message to send to a new JMS destination if you touch the headers (get or set) during the route. Set this option to true to force Camel to send the original JMS message that was received.	false	Boolean
camel.component.jms.format-date-headers-to-iso8601	Sets whether JMS date properties should be formatted according to the ISO 8601 standard.	false	Boolean
camel.component.jms.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.jms.idle-consumer-limit	Specify the limit for the number of consumers that are allowed to be idle at any given time.	1	Integer

Name	Description	Default	Type
camel.component.jms.idle-task-execution-limit	Specifies the limit for idle executions of a receive task, not having received any message within its execution. If this limit is reached, the task will shut down and leave receiving to other executing tasks (in the case of dynamic scheduling; see the maxConcurrentConsumers setting). There is additional doc available from Spring.	1	Integer
camel.component.jms.include-all-j-m-s-x-properties	Whether to include all JMSXxxx properties when mapping from JMS to Camel Message. Setting this to true will include properties such as JMSXAppID, and JMSXUserID etc. Note: If you are using a custom headerFilterStrategy then this option does not apply.	false	Boolean
camel.component.jms.include-sent-j-m-s-message-i-d	Only applicable when sending to JMS destination using InOnly (eg fire and forget). Enabling this option will enrich the Camel Exchange with the actual JMSMessageID that was used by the JMS client when the message was sent to the JMS destination.	false	Boolean
camel.component.jms.jms-key-format-strategy	Pluggable strategy for encoding and decoding JMS keys so they can be compliant with the JMS specification. Camel provides two implementations out of the box: default and passthrough. The default strategy will safely marshal dots and hyphens (. and -). The passthrough strategy leaves the key as is. Can be used for JMS brokers which do not care whether JMS header keys contain illegal characters. You can provide your own implementation of the org.apache.camel.component.jms.JmsKeyFormatStrategy and refer to it using the # notation.		JmsKeyFormatStrategy
camel.component.jms.jms-message-type	Allows you to force the use of a specific javax.jms.Message implementation for sending JMS messages. Possible values are: Bytes, Map, Object, Stream, Text. By default, Camel would determine which JMS message type to use from the In body type. This option allows you to specify it.		JmsMessageType
camel.component.jms.lazy-create-transaction-manager	If true, Camel will create a JmsTransactionManager, if there is no transactionManager injected when option transacted=true.	true	Boolean

Name	Description	Default	Type
camel.component.jms.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.jms.map-jms-message	Specifies whether Camel should auto map the received JMS message to a suited payload type, such as javax.jms.TextMessage to a String etc.	true	Boolean
camel.component.jms.max-concurrent-consumers	Specifies the maximum number of concurrent consumers when consuming from JMS (not for request/reply over JMS). See also the maxMessagesPerTask option to control dynamic scaling up/down of threads. When doing request/reply over JMS then the option replyToMaxConcurrentConsumers is used to control number of concurrent consumers on the reply message listener.		Integer
camel.component.jms.max-messages-per-task	The number of messages per task. -1 is unlimited. If you use a range for concurrent consumers (eg min max), then this option can be used to set a value to eg 100 to control how fast the consumers will shrink when less work is required.	-1	Integer
camel.component.jms.message-converter	To use a custom Spring org.springframework.jms.support.converter.MessageConverter so you can be in control how to map to/from a javax.jms.Message. The option is a org.springframework.jms.support.converter.MessageConverter type.		MessageConverter
camel.component.jms.message-created-strategy	To use the given MessageCreatedStrategy which are invoked when Camel creates new instances of javax.jms.Message objects when Camel is sending a JMS message. The option is a org.apache.camel.component.jms.MessageCreatedStrategy type.		MessageCreatedStrategy

Name	Description	Default	Type
camel.component.jms.message-id-enabled	When sending, specifies whether message IDs should be added. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.	true	Boolean
camel.component.jms.message-listener-container-factory	Registry ID of the MessageListenerContainerFactory used to determine what org.springframework.jms.listener.AbstractMessageListenerContainer to use to consume messages. Setting this will automatically set consumerType to Custom. The option is a org.apache.camel.component.jms.MessageListenerContainerFactory type.		MessageListenerContainerFactory
camel.component.jms.message-timestamp-enabled	Specifies whether timestamps should be enabled by default on sending messages. This is just an hint to the JMS broker. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint the timestamp must be set to its normal value.	true	Boolean
camel.component.jms.password	Password to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String
camel.component.jms.preserve-message-qos	Set to true, if you want to send message using the QoS settings specified on the message, instead of the QoS settings on the JMS endpoint. The following three headers are considered JMSPriority, JMSDeliveryMode, and JMSExpiration. You can provide all or only some of them. If not provided, Camel will fall back to use the values from the endpoint instead. So, when using this option, the headers override the values from the endpoint. The explicitQosEnabled option, by contrast, will only use options set on the endpoint, and not values from the message header.	false	Boolean
camel.component.jms.priority	Values greater than 1 specify the message priority when sending (where 1 is the lowest priority and 9 is the highest). The explicitQosEnabled option must also be enabled in order for this option to have any effect.	4	Integer
camel.component.jms.pub-sub-no-local	Specifies whether to inhibit the delivery of messages published by its own connection.	false	Boolean

Name	Description	Default	Type
camel.component.jms.queue-browse-strategy	To use a custom QueueBrowseStrategy when browsing queues. The option is a org.apache.camel.component.jms.QueueBrowseStrategy type.		QueueBrowseStrategy
camel.component.jms.receive-timeout	The timeout for receiving messages (in milliseconds). The option is a long type.	1000	Long
camel.component.jms.recovery-interval	Specifies the interval between recovery attempts, i.e. when a connection is being refreshed, in milliseconds. The default is 5000 ms, that is, 5 seconds. The option is a long type.	5000	Long
camel.component.jms.reply-to	Provides an explicit ReplyTo destination (overrides any incoming value of Message.getJMSReplyTo() in consumer).		String
camel.component.jms.reply-to-cache-level-name	Sets the cache level by name for the reply consumer when doing request/reply over JMS. This option only applies when using fixed reply queues (not temporary). Camel will by default use: CACHE_CONSUMER for exclusive or shared w/ replyToSelectorName. And CACHE_SESSION for shared without replyToSelectorName. Some JMS brokers such as IBM WebSphere may require to set the replyToCacheLevelName=CACHE_NONE to work. Note: If using temporary queues then CACHE_NONE is not allowed, and you must use a higher value such as CACHE_CONSUMER or CACHE_SESSION.		String
camel.component.jms.reply-to-concurrent-consumers	Specifies the default number of concurrent consumers when doing request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.	1	Integer
camel.component.jms.reply-to-delivery-persistent	Specifies whether to use persistent delivery by default for replies.	true	Boolean
camel.component.jms.reply-to-destination-selector-name	Sets the JMS Selector using the fixed name to be used so you can filter out your own replies from the others when using a shared queue (that is, if you are not using a temporary reply queue).		String

Name	Description	Default	Type
camel.component.jms.reply-to-max-concurrent-consumers	Specifies the maximum number of concurrent consumers when using request/reply over JMS. See also the maxMessagesPerTask option to control dynamic scaling up/down of threads.		Integer
camel.component.jms.reply-to-on-timeout-max-concurrent-consumers	Specifies the maximum number of concurrent consumers for continue routing when timeout occurred when using request/reply over JMS.	1	Integer
camel.component.jms.reply-to-override	Provides an explicit ReplyTo destination in the JMS message, which overrides the setting of replyTo. It is useful if you want to forward the message to a remote Queue and receive the reply message from the ReplyTo destination.		String
camel.component.jms.reply-to-same-destination-allowed	Whether a JMS consumer is allowed to send a reply message to the same destination that the consumer is using to consume from. This prevents an endless loop by consuming and sending back the same message to itself.	false	Boolean
camel.component.jms.reply-to-type	Allows for explicitly specifying which kind of strategy to use for replyTo queues when doing request/reply over JMS. Possible values are: Temporary, Shared, or Exclusive. By default Camel will use temporary queues. However if replyTo has been configured, then Shared is used by default. This option allows you to use exclusive queues instead of shared ones. See Camel JMS documentation for more details, and especially the notes about the implications if running in a clustered environment, and the fact that Shared reply queues has lower performance than its alternatives Temporary and Exclusive.		ReplyToType
camel.component.jms.request-timeout	The timeout for waiting for a reply when using the InOut Exchange Pattern (in milliseconds). The default is 20 seconds. You can include the header CamelJmsRequestTimeout to override this endpoint configured timeout value, and thus have per message individual timeout values. See also the requestTimeoutCheckerInterval option. The option is a long type.	20000	Long

Name	Description	Default	Type
camel.component.jms.request-timeout-checker-interval	Configures how often Camel should check for timed out Exchanges when doing request/reply over JMS. By default Camel checks once per second. But if you must react faster when a timeout occurs, then you can lower this interval, to check more frequently. The timeout is determined by the option requestTimeout. The option is a long type.	1000	Long
camel.component.jms.selector	Sets the JMS selector to use.		String
camel.component.jms.stream-message-type-enabled	Sets whether StreamMessage type is enabled or not. Message payloads of streaming kind such as files, InputStream, etc will either by sent as BytesMessage or StreamMessage. This option controls which kind will be used. By default BytesMessage is used which enforces the entire message payload to be read into memory. By enabling this option the message payload is read into memory in chunks and each chunk is then written to the StreamMessage until no more data.	false	Boolean
camel.component.jms.subscription-durable	Set whether to make the subscription durable. The durable subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a durable subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well.	false	Boolean
camel.component.jms.subscription-name	Set the name of a subscription to create. To be applied in case of a topic (pub-sub domain) with a shared or durable subscription. The subscription name needs to be unique within this client's JMS client id. Default is the class name of the specified message listener. Note: Only 1 concurrent consumer (which is the default of this message listener container) is allowed for each subscription, except for a shared subscription (which requires JMS 2.0).		String

Name	Description	Default	Type
camel.component.jms.subscription-shared	Set whether to make the subscription shared. The shared subscription name to be used can be specified through the subscriptionName property. Default is false. Set this to true to register a shared subscription, typically in combination with a subscriptionName value (unless your message listener class name is good enough as subscription name). Note that shared subscriptions may also be durable, so this flag can (and often will) be combined with subscriptionDurable as well. Only makes sense when listening to a topic (pub-sub domain), therefore this method switches the pubSubDomain flag as well. Requires a JMS 2.0 compatible message broker.	false	Boolean
camel.component.jms.synchronous	Sets whether synchronous processing should be strictly used.	false	Boolean
camel.component.jms.task-executor	Allows you to specify a custom task executor for consuming messages. The option is a org.springframework.core.task.TaskExecutor type.		TaskExecutor
camel.component.jms.test-connection-on-startup	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	Boolean
camel.component.jms.time-to-live	When sending messages, specifies the time-to-live of the message (in milliseconds).	-1	Long
camel.component.jms.transacted	Specifies whether to use transacted mode.	false	Boolean

Name	Description	Default	Type
camel.component.jms.transacted-in-out	Specifies whether InOut operations (request reply) default to using transacted mode If this flag is set to true, then Spring JmsTemplate will have sessionTransacted set to true, and the acknowledgeMode as transacted on the JmsTemplate used for InOut operations. Note from Spring JMS: that within a JTA transaction, the parameters passed to createQueue, createTopic methods are not taken into account. Depending on the Java EE transaction context, the container makes its own decisions on these values. Analogously, these parameters are not taken into account within a locally managed transaction either, since Spring JMS operates on an existing JMS Session in this case. Setting this flag to true will use a short local JMS transaction when running outside of a managed transaction, and a synchronized local JMS transaction in case of a managed transaction (other than an XA transaction) being present. This has the effect of a local JMS transaction being managed alongside the main transaction (which might be a native JDBC transaction), with the JMS transaction committing right after the main transaction.	false	Boolean
camel.component.jms.transaction-manager	The Spring transaction manager to use. The option is a org.springframework.transaction.PlatformTransactionManager type.		PlatformTransactionManager
camel.component.jms.transaction-name	The name of the transaction to use.		String
camel.component.jms.transaction-timeout	The timeout value of the transaction (in seconds), if using transacted mode.	-1	Integer

Name	Description	Default	Type
camel.component.jms.transfer-exception	If enabled and you are using Request Reply messaging (InOut) and an Exchange failed on the consumer side, then the caused Exception will be send back in response as a javax.jms.ObjectMessage. If the client is Camel, the returned Exception is rethrown. This allows you to use Camel JMS as a bridge in your routing - for example, using persistent queues to enable robust routing. Notice that if you also have transferExchange enabled, this option takes precedence. The caught exception is required to be serializable. The original Exception on the consumer side can be wrapped in an outer exception such as org.apache.camel.RuntimeCamelException when returned to the producer. Use this with caution as the data is using Java Object serialization and requires the received to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumer!.	false	Boolean
camel.component.jms.transfer-exchange	You can transfer the exchange over the wire instead of just the body and headers. The following fields are transferred: In body, Out body, Fault body, In headers, Out headers, Fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level. You must enable this option on both the producer and consumer side, so Camel knows the payloads is an Exchange and not a regular payload. Use this with caution as the data is using Java Object serialization and requires the receiver to be able to deserialize the data at Class level, which forces a strong coupling between the producers and consumers having to use compatible Camel versions!.	false	Boolean
camel.component.jms.use-message-i-d-as-correlation-i-d	Specifies whether JMSMessageID should always be used as JMSCorrelationID for InOut messages.	false	Boolean
camel.component.jms.username	Username to use with the ConnectionFactory. You can also configure username/password directly on the ConnectionFactory.		String

Name	Description	Default	Type
camel.component.jms.wait-for-provision-correlation-to-be-updated-counter	Number of times to wait for provisional correlation id to be updated to the actual correlation id when doing request/reply over JMS and when the option useMessageIDAsCorrelationID is enabled.	50	Integer
camel.component.jms.wait-for-provision-correlation-to-be-updated-thread-sleeping-time	Interval in millis to sleep each time while waiting for provisional correlation id to be updated. The option is a long type.	100	Long

CHAPTER 32. JPA

Since Camel 1.0

Both producer and consumer are supported.

The JPA component enables you to store and retrieve Java objects from persistent storage using EJB 3's Java Persistence Architecture (JPA). Java Persistence Architecture (JPA) is a standard interface layer that wraps Object/Relational Mapping (ORM) products such as OpenJPA, Hibernate, TopLink.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jpa</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

32.1. SENDING TO THE ENDPOINT

You can store a Java entity bean in a database by sending it to a JPA producer endpoint. The body of the **In** message is assumed to be an entity bean (that is, a POJO with an @Entity annotation on it) or a collection or array of entity beans.

If the body is a List of entities, use **entityType=java.util.List** as a configuration passed to the producer endpoint.

If the body does not contain one of the previous listed types, put a Message Translator before the endpoint to perform the necessary conversion first.

You can use **query**, **namedQuery** or **nativeQuery** for the producer as well. For the value of the **parameters**, you can use the Simple expression which allows you to retrieve parameter values from Message body, header and etc. Those query can be used for retrieving a set of data with using **SELECT** JPQL/SQL statement as well as executing bulk update/delete with using **UPDATE**/**DELETE** JPQL/SQL statement. Please note that you need to specify **useExecuteUpdate** to **true** if you execute **UPDATE**/**DELETE** with **namedQuery** as camel doesn't look into the named query unlike **query** and **nativeQuery**.

32.2. CONSUMING FROM THE ENDPOINT

Consuming messages from a JPA consumer endpoint removes (or updates) entity beans in the database. This allows you to use a database table as a logical queue: consumers take messages from the queue and then delete/update them to logically remove them from the queue.

If you do not wish to delete the entity bean when it has been processed (and when routing is done), you can specify **consumeDelete=false** on the URI. This will result in the entity being processed each poll.

If you would rather perform some update on the entity to mark it as processed (such as to exclude it from a future query) then you can annotate a method with @Consumed which will be invoked on your entity bean when the entity bean when it has been processed (and when routing is done).

You can use @PreConsumed which will be invoked on your entity bean before it has been processed (before routing).

If you are consuming a lot (100K+) of rows and experience OutOfMemory problems you should set the maximumResults to sensible value.

32.3. URI FORMAT

```
jpa:entityClassName[?options]
```

For sending to the endpoint, the **entityClassName** is optional. If specified, it helps the Type Converterto ensure the body is of the correct type.

For consuming, the **entityClassName** is mandatory.

32.4. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

32.4.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

32.4.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

32.4.3. Component Options

The JPA component supports 9 options, which are listed below.

Name	Description	Default	Type
aliases (common)	Maps an alias to a JPA entity class. The alias can then be used in the endpoint URI (instead of the fully qualified class name).		Map
entityManagerFactory (common)	To use the EntityManagerFactory. This is strongly recommended to configure.		EntityManagerFactory
joinTransaction (common)	The camel-jpa component will join transaction by default. You can use this option to turn this off, for example if you use LOCAL_RESOURCE and join transaction doesn't work with your JPA provider. This option can also be set globally on the JpaComponent, instead of having to set it on all endpoints.	true	boolean
sharedEntityManager (common)	Whether to use Spring's SharedEntityManager for the consumer/producer. Note in most cases joinTransaction should be set to false as this is not an EXTENDED EntityManager.	false	boolean
transactionManager (common)	To use the PlatformTransactionManager for managing transactions.		PlatformTransactionManager
transactionStrategy (common)	To use the TransactionStrategy for running the operations in a transaction.		TransactionStrategy
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

32.4.4. Endpoint Options

The JPA endpoint is configured using URI syntax:

```
jpa:entityType
```

with the following path and query parameters:

32.4.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
entityType (common)	**Required** Entity class name.		Class

32.4.4.2. Query Parameters (44 parameters)

Name	Description	Default	Type
joinTransaction (common)	The camel-jpa component will join transaction by default. You can use this option to turn this off, for example if you use LOCAL_RESOURCE and join transaction doesn't work with your JPA provider. This option can also be set globally on the JpaComponent, instead of having to set it on all endpoints.	true	boolean
maximumResults (common)	Set the maximum number of results to retrieve on the Query.	-1	int
namedQuery (common)	To use a named query.		String

Name	Description	Default	Type
nativeQuery (common)	To use a custom native query. You may want to use the option resultClass also when using native queries.		String
persistenceUnit (common)	**Required** The JPA persistence unit used by default.	camel	String
query (common)	To use a custom query.		String
resultClass (common)	Defines the type of the returned payload (we will call entityManager.createNativeQuery(nativeQuery, resultClass) instead of entityManager.createNativeQuery(nativeQuery)). Without this option, we will return an object array. Only has an affect when using in conjunction with native query when consuming data.		Class
consumeDelete (consumer)	If true, the entity is deleted after it is consumed; if false, the entity is not deleted.	true	boolean
consumeLockEntity (consumer)	Specifies whether or not to set an exclusive lock on each entity bean while processing the results from polling.	true	boolean
deleteHandler (consumer)	To use a custom DeleteHandler to delete the row after the consumer is done processing the exchange.		DeleteHandler
lockModeType (consumer)	To configure the lock mode on the consumer. Enum values: • READ • WRITE • OPTIMISTIC • OPTIMISTIC_FORCE_INCREMENT • PESSIMISTIC_READ • PESSIMISTIC_WRITE • PESSIMISTIC_FORCE_INCREMENT • NONE	PESSIMISTIC_WRITE	LockModeType

Name	Description	Default	Type
maxMessagesPerPoll (consumer)	An integer value to define the maximum number of messages to gather per poll. By default, no maximum is set. Can be used to avoid polling many thousands of messages when starting up the server. Set a value of 0 or negative to disable.		int
preDeleteHandler (consumer)	To use a custom Pre-DeleteHandler to delete the row after the consumer has read the entity.		DeleteHandler
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
skipLockedEntity (consumer)	To configure whether to use NOWAIT on lock and silently skip the entity.	false	boolean
transacted (consumer)	Whether to run the consumer in transacted mode, by which all messages will either commit or rollback, when the entire batch has been processed. The default behavior (false) is to commit all the previously successfully processed messages, and only rollback the last failed message.	false	boolean
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
parameters (consumer (advanced))	This key/value mapping is used for building the query parameters. It is expected to be of the generic type java.util.Map where the keys are the named parameters of a given JPA query and the values are their corresponding effective values you want to select for. When it's used for producer, Simple expression can be used as a parameter value. It allows you to retrieve parameter values from the message body, header and etc.		Map
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
findEntity (producer)	If enabled then the producer will find a single entity by using the message body as key and entityType as the class type. This can be used instead of a query to find a single entity.	false	boolean
flushOnSend (producer)	Flushes the EntityManager after the entity bean has been persisted.	true	boolean
remove (producer)	Indicates to use entityManager.remove(entity).	false	boolean
useExecuteUpdate (producer)	To configure whether to use executeUpdate() when producer executes a query. When you use INSERT, UPDATE or DELETE statement as a named query, you need to specify this option to 'true'.		Boolean

Name	Description	Default	Type
usePersist (producer)	Indicates to use entityManager.persist(entity) instead of entityManager.merge(entity). Note: entityManager.persist(entity) doesn't work for detached entities (where the EntityManager has to execute an UPDATE instead of an INSERT query)!.	false	boolean
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
usePassedInEntityManager (producer (advanced))	If set to true, then Camel will use the EntityManager from the header JpaConstants.ENTITY_MANAGER instead of the configured entity manager on the component/endpoint. This allows end users to control which entity manager will be in use.	false	boolean
entityManagerProperties (advanced)	Additional properties for the entity manager to use.		Map
sharedEntityManager (advanced)	Whether to use Spring's SharedEntityManager for the consumer/producer. Note in most cases joinTransaction should be set to false as this is not an EXTENDED EntityManager.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object

Name	Description	Default	Type
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean

32.5. MESSAGE HEADERS

The JPA component supports 2 message header(s), which are listed below:

Name	Description	Default	Type
CamelEntityManager (common) Constant: ENTITY_MANAGER	The JPA EntityManager object.		EntityManager

Name	Description	Default	Type
CamelJpaParameters (producer) Constant: link: JPA_PARAMETERS_HEADER	Alternative way for passing query parameters as an Exchange header.		Map

32.6. CONFIGURING ENTITYMANAGERFACTORY

It is recommended to configure the JPA component to use a specific **EntityManagerFactory** instance. If failed to do so each **JpaEndpoint** will auto create their own instance of **EntityManagerFactory** which most often is not what you want.

For example, you can instantiate a JPA component that references the **myEMFactory** entity manager factory, as follows:

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
</bean>
```

The **JpaComponent** automatically looks up the **EntityManagerFactory** from the Registry which means you do not need to configure this on the **JpaComponent** as shown above. You only need to do so if there is ambiguity, in which case Camel will log a WARN.

32.7. CONFIGURING TRANSACTIONMANAGER

The **JpaComponent** automatically looks up the **TransactionManager** from the Registry. If Camel won't find any **TransactionManager** instance registered, it will also look up for the **TransactionTemplate** and try to extract **TransactionManager** from it.

If none **TransactionTemplate** is available in the registry, **JpaEndpoint** will auto create their own instance of **TransactionManager** which most often is not what you want.

If more than single instance of the **TransactionManager** is found, Camel will log a WARN. In such cases you might want to instantiate and explicitly configure a JPA component that references the **myTransactionManager** transaction manager, as follows:

```
<bean id="jpa" class="org.apache.camel.component.jpa.JpaComponent">
  <property name="entityManagerFactory" ref="myEMFactory"/>
  <property name="transactionManager" ref="myTransactionManager"/>
</bean>
```

32.8. USING A CONSUMER WITH A NAMED QUERY

For consuming only selected entities, you can use the **namedQuery** URI query option. First, you have to define the named query in the JPA Entity class:

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
```

```
public class MultiSteps {
    ...
}
```

After that you can define a consumer uri as shown below:

```
from("jpa://org.apache.camel.examples.MultiSteps?namedQuery=step1")
.to("bean:myBusinessLogic");
```

32.9. USING A CONSUMER WITH A QUERY

For consuming only selected entities, you can use the **query** URI query option. You only have to define the query option:

```
from("jpa://org.apache.camel.examples.MultiSteps?query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1")
.to("bean:myBusinessLogic");
```

32.10. USING A CONSUMER WITH A NATIVE QUERY

For consuming only selected entities, you can use the **nativeQuery** URI query option. You only have to define the native query option:

```
from("jpa://org.apache.camel.examples.MultiSteps?nativeQuery=select * from MultiSteps where step
= 1")
.to("bean:myBusinessLogic");
```

If you use the native query option, you will receive an object array in the message body.

32.11. USING A PRODUCER WITH A NAMED QUERY

For retrieving selected entities or execute bulk update/delete, you can use the **namedQuery** URI query option. First, you have to define the named query in the JPA Entity class:

```
@Entity
@NamedQuery(name = "step1", query = "select x from MultiSteps x where x.step = 1")
public class MultiSteps {
    ...
}
```

After that you can define a producer uri as shown below:

```
from("direct:namedQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?namedQuery=step1");
```

Note that you need to specify **useExecuteUpdate** option to **true** to execute **UPDATE/DELETE** statement as a named query.

32.12. USING A PRODUCER WITH A QUERY

For retrieving selected entities or execute bulk update/delete, you can use the **query** URI query option. You only have to define the query option:

```
from("direct:query")
.to("jpa://org.apache.camel.examples.MultiSteps?query=select o from
org.apache.camel.examples.MultiSteps o where o.step = 1");
```

32.13. USING A PRODUCER WITH A NATIVE QUERY

For retrieving selected entities or execute bulk update/delete, you can use the **nativeQuery** URI query option. You only have to define the native query option:

```
from("direct:nativeQuery")
.to("jpa://org.apache.camel.examples.MultiSteps?
resultClass=org.apache.camel.examples.MultiSteps&nativeQuery=select * from MultiSteps where
step = 1");
```

If you use the native query option without specifying **resultClass**, you will receive an object array in the message body.

32.14. USING THE JPA-BASED IDEMPOTENT REPOSITORY

The Idempotent Consumer from the EIP patterns is used to filter out duplicate messages. A JPA-based idempotent repository is provided.

To use the JPA based idempotent repository.

Procedure

1. Set up a **persistence-unit** in the persistence.xml file.
2. Set up a **org.springframework.orm.jpa.JpaTemplate** which is used by the **org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository**.
3. Configure the error formatting macro: snippet: java.lang.IndexOutOfBoundsException: Index: 20, Size: 20
4. Configure the idempotent repository as **org.apache.camel.processor.idempotent.jpa.JpaMessageIdRepository**.
5. Create the JPA idempotent repository in the Spring XML file as shown below:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route id="JpaMessageIdRepositoryTest">
    <from uri="direct:start" />
    <idempotentConsumer idempotentRepository="jpaStore">
      <header>messageId</header>
      <to uri="mock:result" />
    </idempotentConsumer>
  </route>
</camelContext>
```

When running this Camel component tests inside your IDE

If you run the tests of this component directly inside your IDE, and not through Maven, then you could see exceptions like these:

```
org.springframework.transaction.CannotCreateTransactionException: Could not open JPA
EntityManager for transaction; nested exception is
<openjpa-2.2.1-r422266:1396819 nonfatal user error>
org.apache.openjpa.persistence.ArgumentException: This configuration disallows runtime
optimization,
but the following listed types were not enhanced at build time or at class load time with a javaagent:
"org.apache.camel.examples.SendEmail".
    at
org.springframework.orm.jpa.JpaTransactionManager.doBegin(JpaTransactionManager.java:427)
    at
org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(Abstract
PlatformTransactionManager.java:371)
    at
org.springframework.transaction.support.TransactionTemplate.execute(TransactionTemplate.java:127)

    at org.apache.camel.processor.jpa.JpaRouteTest.cleanupRepository(JpaRouteTest.java:96)
    at org.apache.camel.processor.jpa.JpaRouteTest.createCamelContext(JpaRouteTest.java:67)
    at org.apache.camel.test.junit5.CamelTestSupport.doSetUp(CamelTestSupport.java:238)
    at org.apache.camel.test.junit5.CamelTestSupport.setUp(CamelTestSupport.java:208)
```

The problem here is that the source has been compiled or recompiled through your IDE and not through Maven, which would enhance the byte-code at build time . To overcome this you need to enable dynamic byte-code enhancement of OpenJPA . For example, assuming the current OpenJPA version being used in Camel is 2.2.1, to run the tests inside your IDE you would need to pass the following argument to the JVM:

```
-javaagent:<path_to_your_local_m2_cache>/org/apache/openjpa/openjpa/2.2.1/openjpa-2.2.1.jar
```

32.15. SPRING BOOT AUTO-CONFIGURATION

When using jpa with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jpa-starter</artifactId>
</dependency>
```

The component supports 10 options, which are listed below.

Name	Description	Default	Type
camel.component.jpa.aliases	Maps an alias to a JPA entity class. The alias can then be used in the endpoint URI (instead of the fully qualified class name).		Map

Name	Description	Default	Type
camel.component.jpa.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.jpa.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.jpa.enabled	Whether to enable auto configuration of the jpa component. This is enabled by default.		Boolean
camel.component.jpa.entity-manager-factory	To use the EntityManagerFactory. This is strongly recommended to configure. The option is a javax.persistence.EntityManagerFactory type.		EntityManagerFactory
camel.component.jpa.join-transaction	The camel-jpa component will join transaction by default. You can use this option to turn this off, for example if you use LOCAL_RESOURCE and join transaction doesn't work with your JPA provider. This option can also be set globally on the JpaComponent, instead of having to set it on all endpoints.	true	Boolean
camel.component.jpa.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.jpa.shared-entity-manager	Whether to use Spring's SharedEntityManager for the consumer/producer. Note in most cases joinTransaction should be set to false as this is not an EXTENDED EntityManager.	false	Boolean
camel.component.jpa.transaction-manager	To use the PlatformTransactionManager for managing transactions. The option is a org.springframework.transaction.PlatformTransactionManager type.		PlatformTransactionManager
camel.component.jpa.transaction-strategy	To use the TransactionStrategy for running the operations in a transaction. The option is a org.apache.camel.component.jpa.TransactionStrategy type.		TransactionStrategy

CHAPTER 33. JSLT

Since Camel 3.1

Only producer is supported

The JSLT component allows you to process a JSON messages using an JSLT expression. This can be ideal when doing JSON to JSON transformation or querying data.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jslt</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

33.1. URI FORMAT

```
jslt:specName[?options]
```

Where **specName** is the classpath-local URI of the specification to invoke; or the complete URL of the remote specification (eg: file://folder/myfile.vm).

33.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

33.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

33.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

33.2.3. Component Options

The JSLT component supports 5 options, which are listed below.

Name	Description	Default	Type
allowTemplateFromHeader (producer)	Whether to allow to use resource template from header or not (default false). Enabling this allows to specify dynamic templates via message header. However this can be seen as a potential security vulnerability if the header is coming from a malicious user, so use this with care.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
functions (advanced)	JSLT can be extended by plugging in functions written in Java.		Collection
objectFilter (advanced)	JSLT can be extended by plugging in a custom jslt object filter.		JsonFilter

33.2.4. Endpoint Options

The JSLT endpoint is configured using URI syntax:

-

```
jslt:resourceUri
```

with the following path and query parameters:

33.2.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
resourceUri (producer)	**Required** Path to the resource. You can prefix with: classpath, file, http, ref, or bean. classpath, file and http loads the resource using these protocols (classpath is default). ref will lookup the resource in the registry. bean will call a method on a bean to be used as the resource. For bean you can specify the method name after dot, eg bean:myBean.myMethod.		String

33.2.4.2. Query Parameters (7 parameters)

Name	Description	Default	Type
allowContextMapAll (producer)	Sets whether the context map should allow access to all details. By default only the message body and headers can be accessed. This option can be enabled for full access to the current Exchange and CamelContext. Doing so impose a potential security risk as this opens access to the full power of CamelContext API.	false	boolean
allowTemplateFromHeader (producer)	Whether to allow to use resource template from header or not (default false). Enabling this allows to specify dynamic templates via message header. However this can be seen as a potential security vulnerability if the header is coming from a malicious user, so use this with care.	false	boolean
contentCache (producer)	Sets whether to use resource content cache or not.	false	boolean
mapBigDecimalAsFloats (producer)	If true, the mapper will use the USE_BIG_DECIMAL_FOR_FLOATS in serialization features.	false	boolean
objectMapper (producer)	Setting a custom JSON Object Mapper to be used.		ObjectMapper
prettyPrint (common)	If true, JSON in output message is pretty printed.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

33.3. MESSAGE HEADERS

The JSLT component supports 2 message header(s), which is/are listed below:

Name	Description	Default	Type
CamelJsltString (producer) Constant: HEADER_JSLT_STRING	The JSLT Template as String.		String
CamelJsltResourceUri (producer) Constant: HEADER_JSLT_RESOURCE_URI	The resource URI.		String

33.4. PASSING VALUES TO JSLT

Camel can supply exchange information as variables when applying a JSLT expression on the body. The available variables from the **Exchange** are:

name	value
headers	The headers of the **In** message as a json object
exchange.properties	The **Exchange** properties as a json object.**exchange** is the name of the variable and *properties* is the path to the exchange properties. Available if **allowContextMapAll** option is true.

All the values that cannot be converted to json with Jackson are denied and will not be available in the jslt expression.

For example, the header named "type" and the exchange property "instance" can be accessed like

```
{
  "type": $headers.type,
  "instance": $exchange.properties.instance
}
```

33.5. SAMPLES

The sample example is as given below.

```
from("activemq:My.Queue").
  to("jslt:com/acme/MyResponse.json");
```

And a file based resource:

```
from("activemq:My.Queue").
  to("jslt:file://myfolder/MyResponse.json?contentCache=true").
  to("activemq:Another.Queue");
```

You can also specify which JSLT expression the component should use dynamically via a header, so for example:

```
from("direct:in").
  setHeader("CamelJsltResourceUri").constant("path/to/my/spec.json").
  to("jslt:dummy?allowTemplateFromHeader=true");
```

Or send whole jslt expression via header: (suitable for querying)

```
from("direct:in").
  setHeader("CamelJsltString").constant(".published").
  to("jslt:dummy?allowTemplateFromHeader=true");
```

Passing exchange properties to the jslt expression can be done like this

```
from("direct:in").
  to("jslt:com/acme/MyResponse.json?allowContextMapAll=true");
```

33.6. SPRING BOOT AUTO-CONFIGURATION

When using jslt with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jslt-starter</artifactId>
</dependency>
```

The component supports 6 options, which are listed below.

Name	Description	Default	Type
camel.component.jslt.allow-template-from-header	Whether to allow to use resource template from header or not (default false). Enabling this allows to specify dynamic templates via message header. However this can be seen as a potential security vulnerability if the header is coming from a malicious user, so use this with care.	false	Boolean
camel.component.jslt.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.jslt.enabled	Whether to enable auto configuration of the jslt component. This is enabled by default.		Boolean
camel.component.jslt.functions	JSLT can be extended by plugging in functions written in Java.		Collection
camel.component.jslt.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.jslt.object-filter	JSLT can be extended by plugging in a custom jslt object filter. The option is a com.schibsted.spt.data.jslt.filters.JsonFilter type.		JsonFilter

CHAPTER 34. KAFKA

Both producer and consumer are supported

The Kafka component is used for communicating with Apache Kafka message broker.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-kafka</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

34.1. URI FORMAT

```
kafka:topic[?options]
```

34.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

34.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

34.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

34.3. COMPONENT OPTIONS

The Kafka component supports 104 options, which are listed below.

Name	Description	Default	Type
additionalProperties (common)	Sets additional properties for either kafka consumer or kafka producer in case they can't be set directly on the camel configurations (e.g: new Kafka properties that are not reflected yet in Camel configurations), the properties have to be prefixed with additionalProperties.. E.g: additionalProperties.transactional.id=12345&additionalProperties.schema.registry.url=http://localhost:8811/avro.		Map
brokers (common)	URL of the Kafka brokers to use. The format is host1:port1,host2:port2, and the list can be a subset of brokers or a VIP pointing to a subset of brokers. This option is known as bootstrap.servers in the Kafka documentation.		String
clientId (common)	The client id is a user-specified string sent in each request to help trace calls. It should logically identify the application making the request.		String
configuration (common)	Allows to pre-configure the Kafka component with common options that the endpoints will reuse.		KafkaConfiguration
headerFilterStrategy (common)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
reconnectBackoffMaxMs (common)	The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.	1000	Integer
shutdownTimeout (common)	Timeout in milliseconds to wait gracefully for the consumer or producer to shutdown and terminate its worker threads.	30000	int

Name	Description	Default	Type
allowManualCommit (consumer)	Whether to allow doing manual commits via KafkaManualCommit. If this option is enabled then an instance of KafkaManualCommit is stored on the Exchange message header, which allows end users to access this API and perform manual offset commits via the Kafka consumer.	false	boolean
autoCommitEnable (consumer)	If true, periodically commit to ZooKeeper the offset of messages already fetched by the consumer. This committed offset will be used when the process fails as the position from which the new consumer will begin.	true	Boolean
autoCommitIntervalMs (consumer)	The frequency in ms that the consumer offsets are committed to zookeeper.	5000	Integer
autoCommitOnStop (consumer)	Whether to perform an explicit auto commit when the consumer stops to ensure the broker has a commit from the last consumed message. This requires the option autoCommitEnable is turned on. The possible values are: sync, async, or none. And sync is the default value. Enum values: ● sync ● async ● none	sync	String
autoOffsetReset (consumer)	What to do when there is no initial offset in ZooKeeper or if an offset is out of range: earliest : automatically reset the offset to the earliest offset latest : automatically reset the offset to the latest offset fail: throw exception to the consumer. Enum values: ● latest ● earliest ● none	latest	String

Name	Description	Default	Type
breakOnFirstError (consumer)	This options controls what happens when a consumer is processing an exchange and it fails. If the option is false then the consumer continues to the next message and processes it. If the option is true then the consumer breaks out, and will seek back to offset of the message that caused a failure, and then re-attempt to process this message. However this can lead to endless processing of the same message if its bound to fail every time, eg a poison message. Therefore its recommended to deal with that for example by using Camel's error handler.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
checkCrcs (consumer)	Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk corruption to the messages occurred. This check adds some overhead, so it may be disabled in cases seeking extreme performance.	true	Boolean
commitTimeoutMs (consumer)	The maximum time, in milliseconds, that the code will wait for a synchronous commit to complete.	5000	Long
consumerRequestTimeoutMs (consumer)	The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.	40000	Integer
consumersCount (consumer)	The number of consumers that connect to kafka server. Each consumer is run on a separate thread, that retrieves and process the incoming data.	1	int

Name	Description	Default	Type
fetchMaxBytes (consumer)	The maximum amount of data the server should return for a fetch request This is not an absolute maximum, if the first message in the first non-empty partition of the fetch is larger than this value, the message will still be returned to ensure that the consumer can make progress. The maximum message size accepted by the broker is defined via message.max.bytes (broker config) or max.message.bytes (topic config). Note that the consumer performs multiple fetches in parallel.	52428800	Integer
fetchMinBytes (consumer)	The minimum amount of data the server should return for a fetch request. If insufficient data is available the request will wait for that much data to accumulate before answering the request.	1	Integer
fetchWaitMaxMs (consumer)	The maximum amount of time the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy fetch.min.bytes.	500	Integer
groupId (consumer)	A string that uniquely identifies the group of consumer processes to which this consumer belongs. By setting the same group id multiple processes indicate that they are all part of the same consumer group. This option is required for consumers.		String
groupInstanceId (consumer)	A unique identifier of the consumer instance provided by the end user. Only non-empty strings are permitted. If set, the consumer is treated as a static member, which means that only one instance with this ID is allowed in the consumer group at any time. This can be used in combination with a larger session timeout to avoid group rebalances caused by transient unavailability (e.g. process restarts). If not set, the consumer will join the group as a dynamic member, which is the traditional behavior.		String
headerDeserializer (consumer)	To use a custom KafkaHeaderDeserializer to deserialize kafka headers values.		KafkaHeaderDeserializer

Name	Description	Default	Type
heartbeatIntervalMs (consumer)	The expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Heartbeats are used to ensure that the consumer's session stays active and to facilitate rebalancing when new consumers join or leave the group. The value must be set lower than session.timeout.ms, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances.	3000	Integer
keyDeserializer (consumer)	Deserializer class for key that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String
maxPartitionFetchBytes (consumer)	The maximum amount of data per-partition the server will return. The maximum total memory used for a request will be #partitions max.partition.fetch.bytes. This size must be at least as large as the maximum message size the server allows or else it is possible for the producer to send messages larger than the consumer can fetch. If that happens, the consumer can get stuck trying to fetch a large message on a certain partition.	1048576	Integer
maxPollIntervalMs (consumer)	The maximum delay between invocations of poll() when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member.		Long
maxPollRecords (consumer)	The maximum number of records returned in a single call to poll().	500	Integer
offsetRepository (consumer)	The offset repository to use in order to locally store the offset of each partition of the topic. Defining one will disable the autocommit.		StateRepository

Name	Description	Default	Type
partitionAssignor (consumer)	The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used.	org.apache.kafka.clients.consumer.RangeAssignor	String
pollOnError (consumer)	What to do if kafka threw an exception while polling for new messages. Will by default use the value from the component configuration unless an explicit value has been configured on the endpoint level. DISCARD will discard the message and continue to poll next message. ERROR_HANDLER will use Camel's error handler to process the exception, and afterwards continue to poll next message. RECONNECT will re-connect the consumer and try poll the message again RETRY will let the consumer retry polling the same message again STOP will stop the consumer (have to be manually started/restarted if the consumer should be able to consume messages again). Enum values: • DISCARD • ERROR_HANDLER • RECONNECT • RETRY • STOP	ERROR_HANDLER	PollOnError
pollTimeoutMs (consumer)	The timeout used when polling the KafkaConsumer.	5000	Long
resumeStrategy (consumer)	This option allows the user to set a custom resume strategy. The resume strategy is executed when partitions are assigned (i.e.: when connecting or reconnecting). It allows implementations to customize how to resume operations and serve as more flexible alternative to the seekTo and the offsetRepository mechanisms. See the KafkaConsumerResumeStrategy for implementation details. This option does not affect the auto commit setting. It is likely that implementations using this setting will also want to evaluate using the manual commit option along with this.		KafkaConsumerResumeStrategy

Name	Description	Default	Type
seekTo (consumer)	Set if KafkaConsumer will read from beginning or end on startup: beginning : read from beginning end : read from end This is replacing the earlier property seekToBeginning. Enum values: • beginning • end		String
sessionTimeoutMs (consumer)	The timeout used to detect failures when using Kafka's group management facilities.	10000	Integer
specificAvroReader (consumer)	This enables the use of a specific Avro reader for use with the Confluent Platform schema registry and the io.confluent.kafka.serializers.KafkaAvroDeserializer. This option is only available in the Confluent Platform (not standard Apache Kafka).	false	boolean
topicIsPattern (consumer)	Whether the topic is a pattern (regular expression). This can be used to subscribe to dynamic number of topics matching the pattern.	false	boolean
valueDeserializer (consumer)	Deserializer class for value that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String
kafkaManualCommitFactory (consumer (advanced))	**Autowired** Factory to use for creating KafkaManualCommit instances. This allows to plugin a custom factory to create custom KafkaManualCommit instances in case special logic is needed when doing manual commits that deviates from the default implementation that comes out of the box.		KafkaManualCommitFactory
pollExceptionStrategy (consumer (advanced))	**Autowired** To use a custom strategy with the consumer to control how to handle exceptions thrown from the Kafka broker while pooling messages.		PollExceptionStrategy

Name	Description	Default	Type
bufferMemorySize (producer)	The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the server the producer will either block or throw an exception based on the preference specified by block.on.buffer.full.This setting should correspond roughly to the total memory the producer will use, but is not a hard bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.	33554432	Integer
compressionCodec (producer)	This parameter allows you to specify the compression codec for all data generated by this producer. Valid values are none, gzip and snappy. Enum values: • none • gzip • snappy • lz4	none	String
connectionMaxIdleMs (producer)	Close idle connections after the number of milliseconds specified by this config.	540000	Integer
deliveryTimeoutMs (producer)	An upper bound on the time to report success or failure after a call to send() returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retriable send failures.	120000	Integer
enableIdempotence (producer)	If set to 'true' the producer will ensure that exactly one copy of each message is written in the stream. If 'false', producer retries may write duplicates of the retried message in the stream. If set to true this option will require max.in.flight.requests.per.connection to be set to 1 and retries cannot be zero and additionally acks must be set to 'all'.	false	boolean
headerSerializer (producer)	To use a custom KafkaHeaderSerializer to serialize kafka headers values.		KafkaHeaderSerializer

Name	Description	Default	Type
key (producer)	The record key (or null if no key is specified). If this option has been configured then it take precedence over header KafkaConstants#KEY.		String
keySerializer (producer)	The serializer class for keys (defaults to the same as for messages if nothing is given).	org.apache.kafka.common.serialization.StringSerializer	String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
lingerMs (producer)	The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs only under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay that is, rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get batch.size worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up. This setting defaults to 0 (i.e. no delay). Setting linger.ms=5, for example, would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absense of load.	0	Integer

Name	Description	Default	Type
maxBlockMs (producer)	The configuration controls how long sending to kafka will block. These methods can be blocked for multiple reasons. For e.g: buffer full, metadata unavailable.This configuration imposes maximum limit on the total time spent in fetching metadata, serialization of key and value, partitioning and allocation of buffer memory when doing a send(). In case of partitionsFor(), this configuration imposes a maximum time threshold on waiting for metadata.	60000	Integer
maxInFlightRequest (producer)	The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e., if retries are enabled).	5	Integer
maxRequestSize (producer)	The maximum size of a request. This is also effectively a cap on the maximum record size. Note that the server has its own cap on record size which may be different from this. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests.	1048576	Integer
metadataMaxAgeMs (producer)	The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.	300000	Integer
metricReporters (producer)	A list of classes to use as metrics reporters. Implementing the MetricReporter interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.		String
metricsSampleWindowMs (producer)	The number of samples maintained to compute metrics.	30000	Integer
noOfMetricsSample (producer)	The number of samples maintained to compute metrics.	2	Integer

Name	Description	Default	Type
partitioner (producer)	The partitioner class for partitioning messages amongst sub-topics. The default partitioner is based on the hash of the key.	org.apache.kafka.clients.producer.internals.DefaultPartitioner	String
partitionKey (producer)	The partition to which the record will be sent (or null if no partition was specified). If this option has been configured then it take precedence over header KafkaConstants#PARTITION_KEY.		Integer
producerBatchSize (producer)	The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server. This configuration controls the default batch size in bytes. No attempt will be made to batch records larger than this size.Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.	16384	Integer
queueBufferingMaxMessages (producer)	The maximum number of unsent messages that can be queued up the producer when using async mode before either the producer must be blocked or data must be dropped.	10000	Integer
receiveBufferBytes (producer)	The size of the TCP receive buffer (SO_RCVBUF) to use when reading data.	65536	Integer
reconnectBackoffMs (producer)	The amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all requests sent by the consumer to the broker.	50	Integer

Name	Description	Default	Type
recordMetadata (producer)	Whether the producer should store the RecordMetadata results from sending to Kafka. The results are stored in a List containing the RecordMetadata metadata's. The list is stored on a header with the key KafkaConstants#KAFKA_RECORDMETA.	true	boolean
requestRequiredAcks (producer)	The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are common: acks=0 If set to zero then the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1. acks=1 This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost. acks=all This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. Enum values: • -1 • 0 • 1 • all	1	String
requestTimeoutMs (producer)	The amount of time the broker will wait trying to meet the request.required.acks requirement before sending back an error to the client.	30000	Integer

Name	Description	Default	Type
retries (producer)	Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries will potentially change the ordering of records because if two records are sent to a single partition, and the first fails and is retried but the second succeeds, then the second record may appear first.	0	Integer
retryBackoffMs (producer)	Before each retry, the producer refreshes the metadata of relevant topics to see if a new leader has been elected. Since leader election takes a bit of time, this property specifies the amount of time that the producer waits before refreshing the metadata.	100	Integer
sendBufferBytes (producer)	Socket write buffer size.	131072	Integer
valueSerializer (producer)	The serializer class for messages.	org.apache.kafka.common.serialization.StringSerializer	String
workerPool (producer)	To use a custom worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing. If using this option then you must handle the lifecycle of the thread pool to shut the pool down when no longer needed.		ExecutorService
workerPoolCoreSize (producer)	Number of core threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	10	Integer
workerPoolMaxSize (producer)	Maximum number of threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	20	Integer

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
kafkaClientFactory (advanced)	**Autowired** Factory to use for creating org.apache.kafka.clients.consumer.KafkaConsumer and org.apache.kafka.clients.producer.KafkaProducer instances. This allows to configure a custom factory to create instances with logic that extends the vanilla Kafka clients.		KafkaClientFactory
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
schemaRegistryURL (confluent)	URL of the Confluent Platform schema registry servers to use. The format is host1:port1,host2:port2. This is known as schema.registry.url in the Confluent Platform documentation. This option is only available in the Confluent Platform (not standard Apache Kafka).		String
interceptorClasses (monitoring)	Sets interceptors for producer or consumers. Producer interceptors have to be classes implementing org.apache.kafka.clients.producer.ProducerInterceptor Consumer interceptors have to be classes implementing org.apache.kafka.clients.consumer.ConsumerInterceptor Note that if you use Producer interceptor on a consumer it will throw a class cast exception in runtime.		String
kerberosBeforeReloginMinTime (security)	Login thread sleep time between refresh attempts.	60000	Integer
kerberosInitCmd (security)	Kerberos kinit command path. Default is /usr/bin/kinit.	/usr/bin/kinit	String

Name	Description	Default	Type
kerberosPrincipalToLocalRules (security)	A list of rules for mapping from principal names to short names (typically operating system usernames). The rules are evaluated in order and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, principal names of the form {username}/{hostname}{REALM} are mapped to {username}. For more details on the format please see the security authorization and acls documentation.. Multiple values can be separated by comma.	DEFAULT	String
kerberosRenewJitter (security)	Percentage of random jitter added to the renewal time.	0.05	Double
kerberosRenewWindowFactor (security)	Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.	0.8	Double
saslJaasConfig (security)	Expose the kafka sasl.jaas.config parameter Example: org.apache.kafka.common.security.plain.PlainLoginModule required username=USERNAME password=PASSWORD;.		String
saslKerberosServiceName (security)	The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.		String
saslMechanism (security)	The Simple Authentication and Security Layer (SASL) Mechanism used. For the valid values see .	GSSAPI	String
securityProtocol (security)	Protocol used to communicate with brokers. SASL_PLAINTEXT, PLAINTEXT and SSL are supported.	PLAINTEXT	String
sslCipherSuites (security)	A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol.By default all the available cipher suites are supported.		String

Name	Description	Default	Type
sslContextParameters (security)	SSL configuration using a Camel SSLContextParameters object. If configured it's applied before the other SSL endpoint parameters. NOTE: Kafka only supports loading keystore from file locations, so prefix the location with file: in the KeyStoreParameters.resource option.		SSLContextParameters
sslEnabledProtocols (security)	The list of protocols enabled for SSL connections. TLSv1.2, TLSv1.1 and TLSv1 are enabled by default.		String
sslEndpointAlgorithm (security)	The endpoint identification algorithm to validate server hostname using server certificate.	https	String
sslKeymanagerAlgorithm (security)	The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.	SunX509	String
sslKeyPassword (security)	The password of the private key in the key store file. This is optional for client.		String
sslKeystoreLocation (security)	The location of the key store file. This is optional for client and can be used for two-way authentication for client.		String
sslKeystorePassword (security)	The store password for the key store file.This is optional for client and only needed if ssl.keystore.location is configured.		String
sslKeystoreType (security)	The file format of the key store file. This is optional for client. Default value is JKS.	JKS	String
sslProtocol (security)	The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.		String
sslProvider (security)	The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.		String

Name	Description	Default	Type
sslTrustmanagerAlgorithm (security)	The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.	PKIX	String
sslTruststoreLocation (security)	The location of the trust store file.		String
sslTruststorePassword (security)	The password for the trust store file.		String
sslTruststoreType (security)	The file format of the trust store file. Default value is JKS.	JKS	String
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean

34.4. ENDPOINT OPTIONS

The Kafka endpoint is configured using URI syntax:

```
kafka:topic
```

with the following path and query parameters:

34.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
topic (common)	**Required** Name of the topic to use. On the consumer you can use comma to separate multiple topics. A producer can only send a message to a single topic.		String

34.4.2. Query Parameters (102 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
additionalProperties (common)	Sets additional properties for either kafka consumer or kafka producer in case they can't be set directly on the camel configurations (e.g: new Kafka properties that are not reflected yet in Camel configurations), the properties have to be prefixed with additionalProperties.. E.g: additionalProperties.transactional.id=12345&additionalProperties.schema.registry.url=http://localhost:8811/avro.		Map
brokers (common)	URL of the Kafka brokers to use. The format is host1:port1,host2:port2, and the list can be a subset of brokers or a VIP pointing to a subset of brokers. This option is known as bootstrap.servers in the Kafka documentation.		String
clientId (common)	The client id is a user-specified string sent in each request to help trace calls. It should logically identify the application making the request.		String
headerFilterStrategy (common)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
reconnectBackoffMaxMs (common)	The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.	1000	Integer
shutdownTimeout (common)	Timeout in milliseconds to wait gracefully for the consumer or producer to shutdown and terminate its worker threads.	30000	int
allowManualCommit (consumer)	Whether to allow doing manual commits via KafkaManualCommit. If this option is enabled then an instance of KafkaManualCommit is stored on the Exchange message header, which allows end users to access this API and perform manual offset commits via the Kafka consumer.	false	boolean
autoCommitEnable (consumer)	If true, periodically commit to ZooKeeper the offset of messages already fetched by the consumer. This committed offset will be used when the process fails as the position from which the new consumer will begin.	true	Boolean

Name	Description	Default	Type
autoCommitIntervalMs (consumer)	The frequency in ms that the consumer offsets are committed to zookeeper.	5000	Integer
autoCommitOnStop (consumer)	Whether to perform an explicit auto commit when the consumer stops to ensure the broker has a commit from the last consumed message. This requires the option autoCommitEnable is turned on. The possible values are: sync, async, or none. And sync is the default value. Enum values: • sync • async • none	sync	String
autoOffsetReset (consumer)	What to do when there is no initial offset in ZooKeeper or if an offset is out of range: earliest : automatically reset the offset to the earliest offset latest : automatically reset the offset to the latest offset fail: throw exception to the consumer. Enum values: • latest • earliest • none	latest	String
breakOnFirstError (consumer)	This options controls what happens when a consumer is processing an exchange and it fails. If the option is false then the consumer continues to the next message and processes it. If the option is true then the consumer breaks out, and will seek back to offset of the message that caused a failure, and then re-attempt to process this message. However this can lead to endless processing of the same message if its bound to fail every time, eg a poison message. Therefore its recommended to deal with that for example by using Camel's error handler.	false	boolean

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
checkCrcs (consumer)	Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk corruption to the messages occurred. This check adds some overhead, so it may be disabled in cases seeking extreme performance.	true	Boolean
commitTimeoutMs (consumer)	The maximum time, in milliseconds, that the code will wait for a synchronous commit to complete.	5000	Long
consumerRequestTimeoutMs (consumer)	The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.	40000	Integer
consumersCount (consumer)	The number of consumers that connect to kafka server. Each consumer is run on a separate thread, that retrieves and process the incoming data.	1	int
fetchMaxBytes (consumer)	The maximum amount of data the server should return for a fetch request This is not an absolute maximum, if the first message in the first non-empty partition of the fetch is larger than this value, the message will still be returned to ensure that the consumer can make progress. The maximum message size accepted by the broker is defined via message.max.bytes (broker config) or max.message.bytes (topic config). Note that the consumer performs multiple fetches in parallel.	52428800	Integer
fetchMinBytes (consumer)	The minimum amount of data the server should return for a fetch request. If insufficient data is available the request will wait for that much data to accumulate before answering the request.	1	Integer
fetchWaitMaxMs (consumer)	The maximum amount of time the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy fetch.min.bytes.	500	Integer

Name	Description	Default	Type
groupId (consumer)	A string that uniquely identifies the group of consumer processes to which this consumer belongs. By setting the same group id multiple processes indicate that they are all part of the same consumer group. This option is required for consumers.		String
groupInstanceId (consumer)	A unique identifier of the consumer instance provided by the end user. Only non-empty strings are permitted. If set, the consumer is treated as a static member, which means that only one instance with this ID is allowed in the consumer group at any time. This can be used in combination with a larger session timeout to avoid group rebalances caused by transient unavailability (e.g. process restarts). If not set, the consumer will join the group as a dynamic member, which is the traditional behavior.		String
headerDeserializer (consumer)	To use a custom KafkaHeaderDeserializer to deserialize kafka headers values.		KafkaHeaderDeserializer
heartbeatIntervalMs (consumer)	The expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Heartbeats are used to ensure that the consumer's session stays active and to facilitate rebalancing when new consumers join or leave the group. The value must be set lower than session.timeout.ms, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances.	3000	Integer
keyDeserializer (consumer)	Deserializer class for key that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String

Name	Description	Default	Type
maxPartitionFetchBytes (consumer)	The maximum amount of data per-partition the server will return. The maximum total memory used for a request will be #partitions max.partition.fetch.bytes. This size must be at least as large as the maximum message size the server allows or else it is possible for the producer to send messages larger than the consumer can fetch. If that happens, the consumer can get stuck trying to fetch a large message on a certain partition.	1048576	Integer
maxPollIntervalMs (consumer)	The maximum delay between invocations of poll() when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member.		Long
maxPollRecords (consumer)	The maximum number of records returned in a single call to poll().	500	Integer
offsetRepository (consumer)	The offset repository to use in order to locally store the offset of each partition of the topic. Defining one will disable the autocommit.		StateRepository
partitionAssignor (consumer)	The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used.	org.apache.kafka.clients.consumer.RangeAssignor	String

Name	Description	Default	Type
pollOnError (consumer)	What to do if kafka threw an exception while polling for new messages. Will by default use the value from the component configuration unless an explicit value has been configured on the endpoint level. DISCARD will discard the message and continue to poll next message. ERROR_HANDLER will use Camel's error handler to process the exception, and afterwards continue to poll next message. RECONNECT will re-connect the consumer and try poll the message again RETRY will let the consumer retry polling the same message again STOP will stop the consumer (have to be manually started/restarted if the consumer should be able to consume messages again). Enum values: • DISCARD • ERROR_HANDLER • RECONNECT • RETRY • STOP	ERROR_HANDLER	PollOnError
pollTimeoutMs (consumer)	The timeout used when polling the KafkaConsumer.	5000	Long
resumeStrategy (consumer)	This option allows the user to set a custom resume strategy. The resume strategy is executed when partitions are assigned (i.e.: when connecting or reconnecting). It allows implementations to customize how to resume operations and serve as more flexible alternative to the seekTo and the offsetRepository mechanisms. See the KafkaConsumerResumeStrategy for implementation details. This option does not affect the auto commit setting. It is likely that implementations using this setting will also want to evaluate using the manual commit option along with this.		KafkaConsumerResumeStrategy

Name	Description	Default	Type
seekTo (consumer)	Set if KafkaConsumer will read from beginning or end on startup: beginning : read from beginning end : read from end This is replacing the earlier property seekToBeginning. Enum values: • beginning • end		String
sessionTimeoutMs (consumer)	The timeout used to detect failures when using Kafka's group management facilities.	10000	Integer
specificAvroReader (consumer)	This enables the use of a specific Avro reader for use with the Confluent Platform schema registry and the io.confluent.kafka.serializers.KafkaAvroDeserializer. This option is only available in the Confluent Platform (not standard Apache Kafka).	false	boolean
topicIsPattern (consumer)	Whether the topic is a pattern (regular expression). This can be used to subscribe to dynamic number of topics matching the pattern.	false	boolean
valueDeserializer (consumer)	Deserializer class for value that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
kafkaManualCommitFactory (consumer (advanced))	Factory to use for creating KafkaManualCommit instances. This allows to plugin a custom factory to create custom KafkaManualCommit instances in case special logic is needed when doing manual commits that deviates from the default implementation that comes out of the box.		KafkaManualCommitFactory
bufferMemorySize (producer)	The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the server the producer will either block or throw an exception based on the preference specified by block.on.buffer.full.This setting should correspond roughly to the total memory the producer will use, but is not a hard bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.	33554432	Integer
compressionCodec (producer)	This parameter allows you to specify the compression codec for all data generated by this producer. Valid values are none, gzip and snappy. Enum values: • none • gzip • snappy • lz4	none	String
connectionMaxIdleMs (producer)	Close idle connections after the number of milliseconds specified by this config.	540000	Integer

Name	Description	Default	Type
deliveryTimeoutMs (producer)	An upper bound on the time to report success or failure after a call to send() returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retriable send failures.	120000	Integer
enableIdempotence (producer)	If set to 'true' the producer will ensure that exactly one copy of each message is written in the stream. If 'false', producer retries may write duplicates of the retried message in the stream. If set to true this option will require max.in.flight.requests.per.connection to be set to 1 and retries cannot be zero and additionally acks must be set to 'all'.	false	boolean
headerSerializer (producer)	To use a custom KafkaHeaderSerializer to serialize kafka headers values.		KafkaHeaderSerializer
key (producer)	The record key (or null if no key is specified). If this option has been configured then it take precedence over header KafkaConstants#KEY.		String
keySerializer (producer)	The serializer class for keys (defaults to the same as for messages if nothing is given).	org.apache.kafka.common.serialization.StringSerializer	String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
lingerMs (producer)	The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs only under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay that is, rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get batch.size worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up. This setting defaults to 0 (i.e. no delay). Setting linger.ms=5, for example, would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absense of load.	0	Integer
maxBlockMs (producer)	The configuration controls how long sending to kafka will block. These methods can be blocked for multiple reasons. For e.g: buffer full, metadata unavailable.This configuration imposes maximum limit on the total time spent in fetching metadata, serialization of key and value, partitioning and allocation of buffer memory when doing a send(). In case of partitionsFor(), this configuration imposes a maximum time threshold on waiting for metadata.	60000	Integer
maxInFlightRequest (producer)	The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e., if retries are enabled).	5	Integer
maxRequestSize (producer)	The maximum size of a request. This is also effectively a cap on the maximum record size. Note that the server has its own cap on record size which may be different from this. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests.	1048576	Integer

Name	Description	Default	Type
metadataMaxAgeMs (producer)	The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.	300000	Integer
metricReporters (producer)	A list of classes to use as metrics reporters. Implementing the MetricReporter interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.		String
metricsSampleWindowMs (producer)	The number of samples maintained to compute metrics.	30000	Integer
noOfMetricsSample (producer)	The number of samples maintained to compute metrics.	2	Integer
partitioner (producer)	The partitioner class for partitioning messages amongst sub-topics. The default partitioner is based on the hash of the key.	org.apache.kafka.clients.producer.internals.DefaultPartitioner	String
partitionKey (producer)	The partition to which the record will be sent (or null if no partition was specified). If this option has been configured then it take precedence over header KafkaConstants#PARTITION_KEY.		Integer
producerBatchSize (producer)	The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server. This configuration controls the default batch size in bytes. No attempt will be made to batch records larger than this size.Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.	16384	Integer

Name	Description	Default	Type
queueBufferingMaxMessages (producer)	The maximum number of unsent messages that can be queued up the producer when using async mode before either the producer must be blocked or data must be dropped.	10000	Integer
receiveBufferBytes (producer)	The size of the TCP receive buffer (SO_RCVBUF) to use when reading data.	65536	Integer
reconnectBackoffMs (producer)	The amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all requests sent by the consumer to the broker.	50	Integer
recordMetadata (producer)	Whether the producer should store the RecordMetadata results from sending to Kafka. The results are stored in a List containing the RecordMetadata metadata's. The list is stored on a header with the key KafkaConstants#KAFKA_RECORDMETA.	true	boolean

Name	Description	Default	Type
requestRequiredAcks (producer)	The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are common: acks=0 If set to zero then the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1. acks=1 This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost. acks=all This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. Enum values: • -1 • 0 • 1 • all	1	String
requestTimeoutMs (producer)	The amount of time the broker will wait trying to meet the request.required.acks requirement before sending back an error to the client.	30000	Integer
retries (producer)	Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries will potentially change the ordering of records because if two records are sent to a single partition, and the first fails and is retried but the second succeeds, then the second record may appear first.	0	Integer

Name	Description	Default	Type
retryBackoffMs (producer)	Before each retry, the producer refreshes the metadata of relevant topics to see if a new leader has been elected. Since leader election takes a bit of time, this property specifies the amount of time that the producer waits before refreshing the metadata.	100	Integer
sendBufferBytes (producer)	Socket write buffer size.	131072	Integer
valueSerializer (producer)	The serializer class for messages.	org.apache.kafka.common.serialization.StringSerializer	String
workerPool (producer)	To use a custom worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing. If using this option then you must handle the lifecycle of the thread pool to shut the pool down when no longer needed.		ExecutorService
workerPoolCoreSize (producer)	Number of core threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	10	Integer
workerPoolMaxSize (producer)	Maximum number of threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	20	Integer
kafkaClientFactory (advanced)	Factory to use for creating org.apache.kafka.clients.consumer.KafkaConsumer and org.apache.kafka.clients.producer.KafkaProducer instances. This allows to configure a custom factory to create instances with logic that extends the vanilla Kafka clients.		KafkaClientFactory
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean

Name	Description	Default	Type
schemaRegistryURL (confluent)	URL of the Confluent Platform schema registry servers to use. The format is host1:port1,host2:port2. This is known as schema.registry.url in the Confluent Platform documentation. This option is only available in the Confluent Platform (not standard Apache Kafka).		String
interceptorClasses (monitoring)	Sets interceptors for producer or consumers. Producer interceptors have to be classes implementing org.apache.kafka.clients.producer.ProducerInterceptor Consumer interceptors have to be classes implementing org.apache.kafka.clients.consumer.ConsumerInterceptor Note that if you use Producer interceptor on a consumer it will throw a class cast exception in runtime.		String
kerberosBeforeReloginMinTime (security)	Login thread sleep time between refresh attempts.	60000	Integer
kerberosInitCmd (security)	Kerberos kinit command path. Default is /usr/bin/kinit.	/usr/bin/kinit	String
kerberosPrincipalToLocalRules (security)	A list of rules for mapping from principal names to short names (typically operating system usernames). The rules are evaluated in order and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, principal names of the form {username}/{hostname}{REALM} are mapped to {username}. For more details on the format please see the security authorization and acls documentation.. Multiple values can be separated by comma.	DEFAULT	String
kerberosRenewJitter (security)	Percentage of random jitter added to the renewal time.	0.05	Double
kerberosRenewWindowFactor (security)	Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.	0.8	Double

Name	Description	Default	Type
saslJaasConfig (security)	Expose the kafka sasl.jaas.config parameter Example: org.apache.kafka.common.security.plain.PlainLoginModule required username=USERNAME password=PASSWORD;.		String
saslKerberosServiceName (security)	The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.		String
saslMechanism (security)	The Simple Authentication and Security Layer (SASL) Mechanism used. For the valid values see .	GSSAPI	String
securityProtocol (security)	Protocol used to communicate with brokers. SASL_PLAINTEXT, PLAINTEXT and SSL are supported.	PLAINTEXT	String
sslCipherSuites (security)	A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol.By default all the available cipher suites are supported.		String
sslContextParameters (security)	SSL configuration using a Camel SSLContextParameters object. If configured it's applied before the other SSL endpoint parameters. NOTE: Kafka only supports loading keystore from file locations, so prefix the location with file: in the KeyStoreParameters.resource option.		SSLContextParameters
sslEnabledProtocols (security)	The list of protocols enabled for SSL connections. TLSv1.2, TLSv1.1 and TLSv1 are enabled by default.		String
sslEndpointAlgorithm (security)	The endpoint identification algorithm to validate server hostname using server certificate.	https	String
sslKeymanagerAlgorithm (security)	The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.	SunX509	String
sslKeyPassword (security)	The password of the private key in the key store file. This is optional for client.		String
sslKeystoreLocation (security)	The location of the key store file. This is optional for client and can be used for two-way authentication for client.		String

Name	Description	Default	Type
sslKeystorePassword (security)	The store password for the key store file.This is optional for client and only needed if ssl.keystore.location is configured.		String
sslKeystoreType (security)	The file format of the key store file. This is optional for client. Default value is JKS.	JKS	String
sslProtocol (security)	The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.		String
sslProvider (security)	The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.		String
sslTrustmanagerAlgorithm (security)	The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.	PKIX	String
sslTruststoreLocation (security)	The location of the trust store file.		String
sslTruststorePassword (security)	The password for the trust store file.		String
sslTruststoreType (security)	The file format of the trust store file. Default value is JKS.	JKS	String

For more information about Producer/Consumer configuration see:

- http://kafka.apache.org/documentation.html#newconsumerconfigs
- http://kafka.apache.org/documentation.html#producerconfigs

34.5. MESSAGE HEADERS

34.5.1. Consumer headers

The following headers are available when consuming messages from Kafka.

Header constant	Header value	Type	Description
KafkaConstants.TOPIC	**"kafka.TOPIC"**	**String**	The topic from where the message originated
KafkaConstants.PARTITION	**"kafka.PARTITION"**	**Integer**	The partition where the message was stored
KafkaConstants.OFFSET	**"kafka.OFFSET"**	**Long**	The offset of the message
KafkaConstants.KEY	**"kafka.KEY"**	**Object**	The key of the message if configured
KafkaConstants.HEADERS	**"kafka.HEADERS"**	**org.apache.kafka.common.header.Headers**	The record headers
KafkaConstants.LAST_RECORD_BEFORE_COMMIT	**"kafka.LAST_RECORD_BEFORE_COMMIT"**	**Boolean**	Whether or not it's the last record before commit (only available if **autoCommitEnable** endpoint parameter is **false**)
KafkaConstants.LAST_POLL_RECORD	**"kafka.LAST_POLL_RECORD"**	**Boolean**	Indicates the last record within the current poll request (only available if **autoCommitEnable** endpoint parameter is **false** or **allowManualCommit** is **true**)
KafkaConstants.MANUAL_COMMIT	**"CamelKafkaManualCommit"**	**KafkaManualCommit**	Can be used for forcing manual offset commit when using Kafka consumer.

34.5.2. Producer headers

Before sending a message to Kafka you can configure the following headers.

Header constant	Header value	Type	Description
KafkaConstants.KEY	**"kafka.KEY"**	**Object**	**Required** The key of the message in order to ensure that all related message goes in the same partition

Header constant	Header value	Type	Description
KafkaConstants.OVERRIDE_TOPIC	**"kafka.OVERRIDE_TOPIC"**	**String**	The topic to which send the message (override and takes precedence), and the header is not preserved.
KafkaConstants.OVERRIDE_TIMESTAMP	**"kafka.OVERRIDE_TIMESTAMP"**	**Long**	The ProducerRecord also has an associated timestamp. If the user did provide a timestamp, the producer will stamp the record with the provided timestamp and the header is not preserved.
KafkaConstants.PARTITION_KEY	**"kafka.PARTITION_KEY"**	**Integer**	Explicitly specify the partition

If you want to send a message to a dynamic topic then use **KafkaConstants.OVERRIDE_TOPIC** as its used as a one-time header that are not send along the message, as its removed in the producer.

After the message is sent to Kafka, the following headers are available

Header constant	Header value	Type	Description
KafkaConstants.KAFKA_RECORDMETA	**"org.apache.kafka.clients.producer.RecordMetadata"**	**List<RecordMetadata>**	The metadata (only configured if **recordMetadata** endpoint parameter is **true**

34.6. CONSUMER ERROR HANDLING

While kafka consumer is polling messages from the kafka broker, then errors can happen. This section describes what happens and what you can configure.

The consumer may throw exception when invoking the Kafka **poll** API. For example if the message cannot be de-serialized due invalid data, and many other kind of errors. Those errors are in the form of **KafkaException** which are either *retryable* or not. The exceptions which can be retried (**RetriableException**) will be retried again (with a poll timeout in between). All other kind of exceptions are handled according to the *pollOnError* configuration. This configuration has the following values:

- DISCARD will discard the message and continue to poll next message.
- ERROR_HANDLER will use Camel's error handler to process the exception, and afterwards continue to poll next message.
- RECONNECT will re-connect the consumer and try poll the message again.
- RETRY will let the consumer retry polling the same message again

- STOP will stop the consumer (have to be manually started/restarted if the consumer should be able to consume messages again).

The default is **ERROR_HANDLER** which will let Camel's error handler (if any configured) process the caused exception. And then afterwards continue to poll the next message. This behavior is similar to the *bridgeErrorHandler* option that Camel components have.

For advanced control then a custom implementation of **org.apache.camel.component.kafka.PollExceptionStrategy** can be configured on the component level, which allows to control which exceptions causes which of the strategies above.

34.7. SAMPLES

34.7.1. Consuming messages from Kafka

Here is the minimal route you need in order to read messages from Kafka.

```
from("kafka:test?brokers=localhost:9092")
    .log("Message received from Kafka : ${body}")
    .log("    on the topic ${headers[kafka.TOPIC]}")
    .log("    on the partition ${headers[kafka.PARTITION]}")
    .log("    with the offset ${headers[kafka.OFFSET]}")
    .log("    with the key ${headers[kafka.KEY]}")
```

If you need to consume messages from multiple topics you can use a comma separated list of topic names.

```
from("kafka:test,test1,test2?brokers=localhost:9092")
    .log("Message received from Kafka : ${body}")
    .log("    on the topic ${headers[kafka.TOPIC]}")
    .log("    on the partition ${headers[kafka.PARTITION]}")
    .log("    with the offset ${headers[kafka.OFFSET]}")
    .log("    with the key ${headers[kafka.KEY]}")
```

It's also possible to subscribe to multiple topics giving a pattern as the topic name and using the **topicIsPattern** option.

```
from("kafka:test*?brokers=localhost:9092&topicIsPattern=true")
    .log("Message received from Kafka : ${body}")
    .log("    on the topic ${headers[kafka.TOPIC]}")
    .log("    on the partition ${headers[kafka.PARTITION]}")
    .log("    with the offset ${headers[kafka.OFFSET]}")
    .log("    with the key ${headers[kafka.KEY]}")
```

When consuming messages from Kafka you can use your own offset management and not delegate this management to Kafka. In order to keep the offsets the component needs a **StateRepository** implementation such as **FileStateRepository**. This bean should be available in the registry. Here how to use it :

```
// Create the repository in which the Kafka offsets will be persisted
FileStateRepository repository = FileStateRepository.fileStateRepository(new
File("/path/to/repo.dat"));

// Bind this repository into the Camel registry
```

```
Registry registry = createCamelRegistry();
registry.bind("offsetRepo", repository);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
                // Setup the topic and broker address
                "&groupId=A" +
                // The consumer processor group ID
                "&autoOffsetReset=earliest" +
                // Ask to start from the beginning if we have unknown offset
                "&offsetRepository=#offsetRepo")
                // Keep the offsets in the previously configured repository
            .to("mock:result");
    }
});
```

34.7.2. Producing messages to Kafka

Here is the minimal route you need in order to write messages to Kafka.

```
from("direct:start")
    .setBody(constant("Message from Camel"))          // Message to send
    .setHeader(KafkaConstants.KEY, constant("Camel")) // Key of the message
    .to("kafka:test?brokers=localhost:9092");
```

34.8. SSL CONFIGURATION

You have 2 different ways to configure the SSL communication on the Kafka` component.

The first way is through the many SSL endpoint parameters

```
from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
            "&groupId=A" +
            "&sslKeystoreLocation=/path/to/keystore.jks" +
            "&sslKeystorePassword=changeit" +
            "&sslKeyPassword=changeit" +
            "&securityProtocol=SSL")
        .to("mock:result");
```

The second way is to use the **sslContextParameters** endpoint parameter.

```
// Configure the SSLContextParameters object
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/path/to/keystore.jks");
ksp.setPassword("changeit");
KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("changeit");
SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);
```

```
// Bind this SSLContextParameters into the Camel registry
Registry registry = createCamelRegistry();
registry.bind("ssl", scp);

// Configure the camel context
DefaultCamelContext camelContext = new DefaultCamelContext(registry);
camelContext.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        from("kafka:" + TOPIC + "?brokers=localhost:{{kafkaPort}}" +
                // Setup the topic and broker address
                "&groupId=A" +
                // The consumer processor group ID
                "&sslContextParameters=#ssl" +
                // The security protocol
                "&securityProtocol=SSL)
                // Reference the SSL configuration
            .to("mock:result");
    }
});
```

34.9. USING THE KAFKA IDEMPOTENT REPOSITORY

The **camel-kafka** library provides a Kafka topic-based idempotent repository.

This repository stores broadcasts all changes to idempotent state (add/remove) in a Kafka topic, and populates a local in-memory cache for each repository's process instance through event sourcing. The topic used must be unique per idempotent repository instance.

The mechanism does not have any requirements about the number of topic partitions; as the repository consumes from all partitions at the same time. It also does not have any requirements about the replication factor of the topic.

Each repository instance that uses the topic (e.g. typically on different machines running in parallel) controls its own consumer group, so in a cluster of 10 Camel processes using the same topic each will control its own offset.

On startup, the instance subscribes to the topic and rewinds the offset to the beginning, rebuilding the cache to the latest state. The cache will not be considered warmed up until one poll of **pollDurationMs** in length returns 0 records. Startup will not be completed until either the cache has warmed up, or 30 seconds go by; if the latter happens the idempotent repository may be in an inconsistent state until its consumer catches up to the end of the topic.

Be mindful of the format of the header used for the uniqueness check. By default, it uses Strings as the data types. When using primitive numeric formats, the header must be deserialized accordingly. Check the samples below for examples.

A **KafkaIdempotentRepository** has the following properties:

Property	Description
topic	The name of the Kafka topic to use to broadcast changes. (required)

Property	Description
bootstrapServers	The **bootstrap.servers** property on the internal Kafka producer and consumer. Use this as shorthand if not setting **consumerConfig** and **producerConfig**. If used, this component will apply sensible default configurations for the producer and consumer.
producerConfig	Sets the properties that will be used by the Kafka producer that broadcasts changes. Overrides **bootstrapServers**, so must define the Kafka **bootstrap.servers** property itself
consumerConfig	Sets the properties that will be used by the Kafka consumer that populates the cache from the topic. Overrides **bootstrapServers**, so must define the Kafka **bootstrap.servers** property itself
maxCacheSize	How many of the most recently used keys should be stored in memory (default 1000).
pollDurationMs	The poll duration of the Kafka consumer. The local caches are updated immediately. This value will affect how far behind other peers that update their caches from the topic are relative to the idempotent consumer instance that sent the cache action message. The default value of this is 100 ms. If setting this value explicitly, be aware that there is a tradeoff between the remote cache liveness and the volume of network traffic between this repository's consumer and the Kafka brokers. The cache warmup process also depends on there being one poll that fetches nothing - this indicates that the stream has been consumed up to the current point. If the poll duration is excessively long for the rate at which messages are sent on the topic, there exists a possibility that the cache cannot be warmed up and will operate in an inconsistent state relative to its peers until it catches up.

The repository can be instantiated by defining the **topic** and **bootstrapServers**, or the **producerConfig** and **consumerConfig** property sets can be explicitly defined to enable features such as SSL/SASL. To use, this repository must be placed in the Camel registry, either manually or by registration as a bean in Spring/Blueprint, as it is **CamelContext** aware.

Sample usage is as follows:

```
KafkaIdempotentRepository kafkaIdempotentRepository = new
KafkaIdempotentRepository("idempotent-db-inserts", "localhost:9091");

SimpleRegistry registry = new SimpleRegistry();
registry.put("insertDbIdemRepo", kafkaIdempotentRepository); // must be registered in the registry, to
enable access to the CamelContext
CamelContext context = new CamelContext(registry);

// later in RouteBuilder...
from("direct:performInsert")
    .idempotentConsumer(header("id")).messageIdRepositoryRef("insertDbIdemRepo")
        // once-only insert into database
    .end()
```

In XML:

```
<!-- simple -->
<bean id="insertDbIdemRepo"
  class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="bootstrapServers" value="localhost:9091"/>
</bean>

<!-- complex -->
<bean id="insertDbIdemRepo"
  class="org.apache.camel.processor.idempotent.kafka.KafkaIdempotentRepository">
  <property name="topic" value="idempotent-db-inserts"/>
  <property name="maxCacheSize" value="10000"/>
  <property name="consumerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
  <property name="producerConfig">
    <props>
      <prop key="bootstrap.servers">localhost:9091</prop>
    </props>
  </property>
</bean>
```

There are 3 alternatives to choose from when using idempotency with numeric identifiers. The first one is to use the static method **numericHeader** method from **org.apache.camel.component.kafka.serde.KafkaSerdeHelper** to perform the conversion for you:

```
from("direct:performInsert")
    .idempotentConsumer(numericHeader("id")).messageIdRepositoryRef("insertDbIdemRepo")
        // once-only insert into database
    .end()
```

Alternatively, it is possible use a custom serializer configured via the route URL to perform the conversion:

```
public class CustomHeaderDeserializer extends DefaultKafkaHeaderDeserializer {
    private static final Logger LOG = LoggerFactory.getLogger(CustomHeaderDeserializer.class);

    @Override
    public Object deserialize(String key, byte[] value) {
        if (key.equals("id")) {
            BigInteger bi = new BigInteger(value);

            return String.valueOf(bi.longValue());
        } else {
            return super.deserialize(key, value);
        }
    }
}
```

Lastly, it is also possible to do so in a processor:

```
from(from).routeId("foo")
    .process(exchange -> {
        byte[] id = exchange.getIn().getHeader("id", byte[].class);

        BigInteger bi = new BigInteger(id);
        exchange.getIn().setHeader("id", String.valueOf(bi.longValue()));
    })
    .idempotentConsumer(header("id"))
    .messageIdRepositoryRef("kafkaIdempotentRepository")
    .to(to);
```

34.10. USING MANUAL COMMIT WITH KAFKA CONSUMER

By default the Kafka consumer will use auto commit, where the offset will be committed automatically in the background using a given interval.

In case you want to force manual commits, you can use **KafkaManualCommit** API from the Camel Exchange, stored on the message header. This requires to turn on manual commits by either setting the option **allowManualCommit** to **true** on the **KafkaComponent** or on the endpoint, for example:

```
KafkaComponent kafka = new KafkaComponent();
kafka.setAllowManualCommit(true);
...
camelContext.addComponent("kafka", kafka);
```

You can then use the **KafkaManualCommit** from Java code such as a Camel **Processor**:

```
public void process(Exchange exchange) {
    KafkaManualCommit manual =
        exchange.getIn().getHeader(KafkaConstants.MANUAL_COMMIT, KafkaManualCommit.class);
    manual.commit();
}
```

This will force a synchronous commit which will block until the commit is acknowledge on Kafka, or if it fails an exception is thrown. You can use an asynchronous commit as well by configuring the **KafkaManualCommitFactory** with the `DefaultKafkaManualAsyncCommitFactory` implementation.

The commit will then be done in the next consumer loop using the kafka asynchronous commit api. Be aware that records from a partition must be processed and committed by a unique thread. If not, this could lead with non consistent behaviors. This is mostly useful with aggregation's completion timeout strategies.

If you want to use a custom implementation of **KafkaManualCommit** then you can configure a custom **KafkaManualCommitFactory** on the **KafkaComponent** that creates instances of your custom implementation.

34.11. KAFKA HEADERS PROPAGATION

When consuming messages from Kafka, headers will be propagated to camel exchange headers automatically. Producing flow backed by same behaviour - camel headers of particular exchange will be propagated to kafka message headers.

Since kafka headers allows only **byte[]** values, in order camel exchange header to be propagated its value should be serialized to **bytes[]**, otherwise header will be skipped. Following header value types are

supported: **String**, **Integer**, **Long**, **Double**, **Boolean**, **byte[]**. Note: all headers propagated **from** kafka **to** camel exchange will contain **byte[]** value by default. In order to override default functionality uri parameters can be set: **headerDeserializer** for **from** route and **headerSerializer** for **to** route. Example:

```
from("kafka:my_topic?headerDeserializer=#myDeserializer")
...
.to("kafka:my_topic?headerSerializer=#mySerializer")
```

By default all headers are being filtered by **KafkaHeaderFilterStrategy**. Strategy filters out headers which start with **Camel** or **org.apache.camel** prefixes. Default strategy can be overridden by using **headerFilterStrategy** uri parameter in both **to** and **from** routes:

```
from("kafka:my_topic?headerFilterStrategy=#myStrategy")
...
.to("kafka:my_topic?headerFilterStrategy=#myStrategy")
```

myStrategy object should be subclass of **HeaderFilterStrategy** and must be placed in the Camel registry, either manually or by registration as a bean in Spring/Blueprint, as it is **CamelContext** aware.

34.12. SPRING BOOT AUTO-CONFIGURATION

When using kafka with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-kafka-starter</artifactId>
</dependency>
```

The component supports 105 options, which are listed below.

Name	Description	Default	Type
camel.component.kafka.additional-properties	Sets additional properties for either kafka consumer or kafka producer in case they can't be set directly on the camel configurations (e.g: new Kafka properties that are not reflected yet in Camel configurations), the properties have to be prefixed with additionalProperties.. E.g: additionalProperties.transactional.id=12345&additionalProperties.schema.registry.url=http://localhost:8811/avro.		Map
camel.component.kafka.allow-manual-commit	Whether to allow doing manual commits via KafkaManualCommit. If this option is enabled then an instance of KafkaManualCommit is stored on the Exchange message header, which allows end users to access this API and perform manual offset commits via the Kafka consumer.	false	Boolean

Name	Description	Default	Type
camel.component.kafka.auto-commit-enable	If true, periodically commit to ZooKeeper the offset of messages already fetched by the consumer. This committed offset will be used when the process fails as the position from which the new consumer will begin.	true	Boolean
camel.component.kafka.auto-commit-interval-ms	The frequency in ms that the consumer offsets are committed to zookeeper.	5000	Integer
camel.component.kafka.auto-commit-on-stop	Whether to perform an explicit auto commit when the consumer stops to ensure the broker has a commit from the last consumed message. This requires the option autoCommitEnable is turned on. The possible values are: sync, async, or none. And sync is the default value.	sync	String
camel.component.kafka.auto-offset-reset	What to do when there is no initial offset in ZooKeeper or if an offset is out of range: earliest : automatically reset the offset to the earliest offset latest : automatically reset the offset to the latest offset fail: throw exception to the consumer.	latest	String
camel.component.kafka.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.kafka.break-on-first-error	This options controls what happens when a consumer is processing an exchange and it fails. If the option is false then the consumer continues to the next message and processes it. If the option is true then the consumer breaks out, and will seek back to offset of the message that caused a failure, and then re-attempt to process this message. However this can lead to endless processing of the same message if its bound to fail every time, eg a poison message. Therefore its recommended to deal with that for example by using Camel's error handler.	false	Boolean

Name	Description	Default	Type
camel.component.kafka.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.kafka.brokers	URL of the Kafka brokers to use. The format is host1:port1,host2:port2, and the list can be a subset of brokers or a VIP pointing to a subset of brokers. This option is known as bootstrap.servers in the Kafka documentation.		String
camel.component.kafka.buffer-memory-size	The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the server the producer will either block or throw an exception based on the preference specified by block.on.buffer.full.This setting should correspond roughly to the total memory the producer will use, but is not a hard bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.	33554432	Integer
camel.component.kafka.check-crcs	Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk corruption to the messages occurred. This check adds some overhead, so it may be disabled in cases seeking extreme performance.	true	Boolean
camel.component.kafka.client-id	The client id is a user-specified string sent in each request to help trace calls. It should logically identify the application making the request.		String
camel.component.kafka.commit-timeout-ms	The maximum time, in milliseconds, that the code will wait for a synchronous commit to complete. The option is a java.lang.Long type.	5000	Long
camel.component.kafka.compression-codec	This parameter allows you to specify the compression codec for all data generated by this producer. Valid values are none, gzip and snappy.	none	String

Name	Description	Default	Type
camel.component.kafka.configuration	Allows to pre-configure the Kafka component with common options that the endpoints will reuse. The option is a org.apache.camel.component.kafka.KafkaConfiguration type.		KafkaConfiguration
camel.component.kafka.connection-max-idle-ms	Close idle connections after the number of milliseconds specified by this config.	540000	Integer
camel.component.kafka.consumer-request-timeout-ms	The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.	40000	Integer
camel.component.kafka.consumers-count	The number of consumers that connect to kafka server. Each consumer is run on a separate thread, that retrieves and process the incoming data.	1	Integer
camel.component.kafka.delivery-timeout-ms	An upper bound on the time to report success or failure after a call to send() returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retriable send failures.	120000	Integer
camel.component.kafka.enable-idempotence	If set to 'true' the producer will ensure that exactly one copy of each message is written in the stream. If 'false', producer retries may write duplicates of the retried message in the stream. If set to true this option will require max.in.flight.requests.per.connection to be set to 1 and retries cannot be zero and additionally acks must be set to 'all'.	false	Boolean
camel.component.kafka.enabled	Whether to enable auto configuration of the kafka component. This is enabled by default.		Boolean
camel.component.kafka.fetch-max-bytes	The maximum amount of data the server should return for a fetch request This is not an absolute maximum, if the first message in the first non-empty partition of the fetch is larger than this value, the message will still be returned to ensure that the consumer can make progress. The maximum message size accepted by the broker is defined via message.max.bytes (broker config) or max.message.bytes (topic config). Note that the consumer performs multiple fetches in parallel.	52428800	Integer

Name	Description	Default	Type
camel.component.kafka.fetch-min-bytes	The minimum amount of data the server should return for a fetch request. If insufficient data is available the request will wait for that much data to accumulate before answering the request.	1	Integer
camel.component.kafka.fetch-wait-max-ms	The maximum amount of time the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy fetch.min.bytes.	500	Integer
camel.component.kafka.group-id	A string that uniquely identifies the group of consumer processes to which this consumer belongs. By setting the same group id multiple processes indicate that they are all part of the same consumer group. This option is required for consumers.		String
camel.component.kafka.group-instance-id	A unique identifier of the consumer instance provided by the end user. Only non-empty strings are permitted. If set, the consumer is treated as a static member, which means that only one instance with this ID is allowed in the consumer group at any time. This can be used in combination with a larger session timeout to avoid group rebalances caused by transient unavailability (e.g. process restarts). If not set, the consumer will join the group as a dynamic member, which is the traditional behavior.		String
camel.component.kafka.header-deserializer	To use a custom KafkaHeaderDeserializer to deserialize kafka headers values. The option is a org.apache.camel.component.kafka.serde.KafkaHeaderDeserializer type.		KafkaHeaderDeserializer
camel.component.kafka.header-filter-strategy	To use a custom HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.kafka.header-serializer	To use a custom KafkaHeaderSerializer to serialize kafka headers values. The option is a org.apache.camel.component.kafka.serde.KafkaHeaderSerializer type.		KafkaHeaderSerializer

Name	Description	Default	Type
camel.component.kafka.heartbeat-interval-ms	The expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Heartbeats are used to ensure that the consumer's session stays active and to facilitate rebalancing when new consumers join or leave the group. The value must be set lower than session.timeout.ms, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances.	3000	Integer
camel.component.kafka.interceptor-classes	Sets interceptors for producer or consumers. Producer interceptors have to be classes implementing org.apache.kafka.clients.producer.ProducerInterceptor Consumer interceptors have to be classes implementing org.apache.kafka.clients.consumer.ConsumerInterceptor Note that if you use Producer interceptor on a consumer it will throw a class cast exception in runtime.		String
camel.component.kafka.kafka-client-factory	Factory to use for creating org.apache.kafka.clients.consumer.KafkaConsumer and org.apache.kafka.clients.producer.KafkaProducer instances. This allows to configure a custom factory to create instances with logic that extends the vanilla Kafka clients. The option is a org.apache.camel.component.kafka.KafkaClientFactory type.		KafkaClientFactory
camel.component.kafka.kafka-manual-commit-factory	Factory to use for creating KafkaManualCommit instances. This allows to plugin a custom factory to create custom KafkaManualCommit instances in case special logic is needed when doing manual commits that deviates from the default implementation that comes out of the box. The option is a org.apache.camel.component.kafka.KafkaManualCommitFactory type.		KafkaManualCommitFactory
camel.component.kafka.kerberos-before-relogin-min-time	Login thread sleep time between refresh attempts.	60000	Integer

Name	Description	Default	Type
camel.component.kafka.kerberos-init-cmd	Kerberos kinit command path. Default is /usr/bin/kinit.	/usr/bin/kinit	String
camel.component.kafka.kerberos-principal-to-local-rules	A list of rules for mapping from principal names to short names (typically operating system usernames). The rules are evaluated in order and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, principal names of the form {username}/{hostname}{REALM} are mapped to {username}. For more details on the format please see the security authorization and acls documentation.. Multiple values can be separated by comma.	DEFAULT	String
camel.component.kafka.kerberos-renew-jitter	Percentage of random jitter added to the renewal time.		Double
camel.component.kafka.kerberos-renew-window-factor	Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.		Double
camel.component.kafka.key	The record key (or null if no key is specified). If this option has been configured then it take precedence over header KafkaConstants#KEY.		String
camel.component.kafka.key-deserializer	Deserializer class for key that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String
camel.component.kafka.key-serializer	The serializer class for keys (defaults to the same as for messages if nothing is given).	org.apache.kafka.common.serialization.StringSerializer	String

Name	Description	Default	Type
camel.component.kafka.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.kafka.linger-ms	The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs only under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay that is, rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get batch.size worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up. This setting defaults to 0 (i.e. no delay). Setting linger.ms=5, for example, would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absense of load.	0	Integer
camel.component.kafka.max-block-ms	The configuration controls how long sending to kafka will block. These methods can be blocked for multiple reasons. For e.g: buffer full, metadata unavailable.This configuration imposes maximum limit on the total time spent in fetching metadata, serialization of key and value, partitioning and allocation of buffer memory when doing a send(). In case of partitionsFor(), this configuration imposes a maximum time threshold on waiting for metadata.	60000	Integer

Name	Description	Default	Type
camel.component.kafka.max-in-flight-request	The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e., if retries are enabled).	5	Integer
camel.component.kafka.max-partition-fetch-bytes	The maximum amount of data per-partition the server will return. The maximum total memory used for a request will be #partitions max.partition.fetch.bytes. This size must be at least as large as the maximum message size the server allows or else it is possible for the producer to send messages larger than the consumer can fetch. If that happens, the consumer can get stuck trying to fetch a large message on a certain partition.	1048576	Integer
camel.component.kafka.max-poll-interval-ms	The maximum delay between invocations of poll() when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member. The option is a java.lang.Long type.		Long
camel.component.kafka.max-poll-records	The maximum number of records returned in a single call to poll().	500	Integer
camel.component.kafka.max-request-size	The maximum size of a request. This is also effectively a cap on the maximum record size. Note that the server has its own cap on record size which may be different from this. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests.	1048576	Integer
camel.component.kafka.metadata-max-age-ms	The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.	300000	Integer
camel.component.kafka.metric-reporters	A list of classes to use as metrics reporters. Implementing the MetricReporter interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.		String

Name	Description	Default	Type
camel.component.kafka.metrics-sample-window-ms	The number of samples maintained to compute metrics.	30000	Integer
camel.component.kafka.no-of-metrics-sample	The number of samples maintained to compute metrics.	2	Integer
camel.component.kafka.offset-repository	The offset repository to use in order to locally store the offset of each partition of the topic. Defining one will disable the autocommit. The option is a org.apache.camel.spi.StateRepository<java.lang.String, java.lang.String> type.		StateRepository
camel.component.kafka.partition-assignor	The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used.	org.apache.kafka.clients.consumer.RangeAssignor	String
camel.component.kafka.partition-key	The partition to which the record will be sent (or null if no partition was specified). If this option has been configured then it take precedence over header KafkaConstants#PARTITION_KEY.		Integer
camel.component.kafka.partitioner	The partitioner class for partitioning messages amongst sub-topics. The default partitioner is based on the hash of the key.	org.apache.kafka.clients.producer.internals.DefaultPartitioner	String
camel.component.kafka.poll-exception-strategy	To use a custom strategy with the consumer to control how to handle exceptions thrown from the Kafka broker while pooling messages. The option is a org.apache.camel.component.kafka.PollExceptionStrategy type.		PollExceptionStrategy

Name	Description	Default	Type
camel.component.kafka.poll-on-error	What to do if kafka threw an exception while polling for new messages. Will by default use the value from the component configuration unless an explicit value has been configured on the endpoint level. DISCARD will discard the message and continue to poll next message. ERROR_HANDLER will use Camel's error handler to process the exception, and afterwards continue to poll next message. RECONNECT will re-connect the consumer and try poll the message again RETRY will let the consumer retry polling the same message again STOP will stop the consumer (have to be manually started/restarted if the consumer should be able to consume messages again).		PollOnError
camel.component.kafka.poll-timeout-ms	The timeout used when polling the KafkaConsumer. The option is a java.lang.Long type.	5000	Long
camel.component.kafka.producer-batch-size	The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server. This configuration controls the default batch size in bytes. No attempt will be made to batch records larger than this size.Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.	16384	Integer
camel.component.kafka.queue-buffering-max-messages	The maximum number of unsent messages that can be queued up the producer when using async mode before either the producer must be blocked or data must be dropped.	10000	Integer
camel.component.kafka.receive-buffer-bytes	The size of the TCP receive buffer (SO_RCVBUF) to use when reading data.	65536	Integer
camel.component.kafka.reconnect-backoff-max-ms	The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.	1000	Integer

Name	Description	Default	Type
camel.component.kafka.reconnect-backoff-ms	The amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all requests sent by the consumer to the broker.	50	Integer
camel.component.kafka.record-metadata	Whether the producer should store the RecordMetadata results from sending to Kafka. The results are stored in a List containing the RecordMetadata metadata's. The list is stored on a header with the key KafkaConstants#KAFKA_RECORDMETA.	true	Boolean
camel.component.kafka.request-required-acks	The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are common: acks=0 If set to zero then the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1. acks=1 This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost. acks=all This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.	1	String
camel.component.kafka.request-timeout-ms	The amount of time the broker will wait trying to meet the request.required.acks requirement before sending back an error to the client.	30000	Integer

Name	Description	Default	Type
camel.component.kafka.resume-strategy	This option allows the user to set a custom resume strategy. The resume strategy is executed when partitions are assigned (i.e.: when connecting or reconnecting). It allows implementations to customize how to resume operations and serve as more flexible alternative to the seekTo and the offsetRepository mechanisms. See the KafkaConsumerResumeStrategy for implementation details. This option does not affect the auto commit setting. It is likely that implementations using this setting will also want to evaluate using the manual commit option along with this. The option is a org.apache.camel.component.kafka.consumer.support.KafkaConsumerResumeStrategy type.		KafkaConsumerResumeStrategy
camel.component.kafka.retries	Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries will potentially change the ordering of records because if two records are sent to a single partition, and the first fails and is retried but the second succeeds, then the second record may appear first.	0	Integer
camel.component.kafka.retry-backoff-ms	Before each retry, the producer refreshes the metadata of relevant topics to see if a new leader has been elected. Since leader election takes a bit of time, this property specifies the amount of time that the producer waits before refreshing the metadata.	100	Integer
camel.component.kafka.sasl-jaas-config	Expose the kafka sasl.jaas.config parameter Example: org.apache.kafka.common.security.plain.PlainLoginModule required username=USERNAME password=PASSWORD;.		String
camel.component.kafka.sasl-kerberos-service-name	The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.		String
camel.component.kafka.sasl-mechanism	The Simple Authentication and Security Layer (SASL) Mechanism used. For the valid values see .	GSSAPI	String

Name	Description	Default	Type
camel.component.kafka.schema-registry-u-r-l	URL of the Confluent Platform schema registry servers to use. The format is host1:port1,host2:port2. This is known as schema.registry.url in the Confluent Platform documentation. This option is only available in the Confluent Platform (not standard Apache Kafka).		String
camel.component.kafka.security-protocol	Protocol used to communicate with brokers. SASL_PLAINTEXT, PLAINTEXT and SSL are supported.	PLAINTEXT	String
camel.component.kafka.seek-to	Set if KafkaConsumer will read from beginning or end on startup: beginning : read from beginning end : read from end This is replacing the earlier property seekToBeginning.		String
camel.component.kafka.send-buffer-bytes	Socket write buffer size.	131072	Integer
camel.component.kafka.session-timeout-ms	The timeout used to detect failures when using Kafka's group management facilities.	10000	Integer
camel.component.kafka.shutdown-timeout	Timeout in milliseconds to wait gracefully for the consumer or producer to shutdown and terminate its worker threads.	30000	Integer
camel.component.kafka.specific-avro-reader	This enables the use of a specific Avro reader for use with the Confluent Platform schema registry and the io.confluent.kafka.serializers.KafkaAvroDeserializer. This option is only available in the Confluent Platform (not standard Apache Kafka).	false	Boolean
camel.component.kafka.ssl-cipher-suites	A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol.By default all the available cipher suites are supported.		String

Name	Description	Default	Type
camel.component.kafka.ssl-context-parameters	SSL configuration using a Camel SSLContextParameters object. If configured it's applied before the other SSL endpoint parameters. NOTE: Kafka only supports loading keystore from file locations, so prefix the location with file: in the KeyStoreParameters.resource option. The option is a org.apache.camel.support.jsse.SSLContextParameters type.		SSLContextParameters
camel.component.kafka.ssl-enabled-protocols	The list of protocols enabled for SSL connections. TLSv1.2, TLSv1.1 and TLSv1 are enabled by default.		String
camel.component.kafka.ssl-endpoint-algorithm	The endpoint identification algorithm to validate server hostname using server certificate.	https	String
camel.component.kafka.ssl-key-password	The password of the private key in the key store file. This is optional for client.		String
camel.component.kafka.ssl-keymanager-algorithm	The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.	SunX509	String
camel.component.kafka.ssl-keystore-location	The location of the key store file. This is optional for client and can be used for two-way authentication for client.		String
camel.component.kafka.ssl-keystore-password	The store password for the key store file.This is optional for client and only needed if ssl.keystore.location is configured.		String
camel.component.kafka.ssl-keystore-type	The file format of the key store file. This is optional for client. Default value is JKS.	JKS	String
camel.component.kafka.ssl-protocol	The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.		String

Name	Description	Default	Type
camel.component.kafka.ssl-provider	The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.		String
camel.component.kafka.ssl-trustmanager-algorithm	The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.	PKIX	String
camel.component.kafka.ssl-truststore-location	The location of the trust store file.		String
camel.component.kafka.ssl-truststore-password	The password for the trust store file.		String
camel.component.kafka.ssl-truststore-type	The file format of the trust store file. Default value is JKS.	JKS	String
camel.component.kafka.synchronous	Sets whether synchronous processing should be strictly used.	false	Boolean
camel.component.kafka.topic-is-pattern	Whether the topic is a pattern (regular expression). This can be used to subscribe to dynamic number of topics matching the pattern.	false	Boolean
camel.component.kafka.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.kafka.value-deserializer	Deserializer class for value that implements the Deserializer interface.	org.apache.kafka.common.serialization.StringDeserializer	String

Name	Description	Default	Type
camel.component.kafka.value-serializer	The serializer class for messages.	org.apache.kafka.common.serialization.StringSerializer	String
camel.component.kafka.worker-pool	To use a custom worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing. If using this option then you must handle the lifecycle of the thread pool to shut the pool down when no longer needed. The option is a java.util.concurrent.ExecutorService type.		ExecutorService
camel.component.kafka.worker-pool-core-size	Number of core threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	10	Integer
camel.component.kafka.worker-pool-max-size	Maximum number of threads for the worker pool for continue routing Exchange after kafka server has acknowledge the message that was sent to it from KafkaProducer using asynchronous non-blocking processing.	20	Integer

CHAPTER 35. KAMELET

Both producer and consumer are supported

The Kamelet Component provides support for interacting with the Camel Route Template engine using Endpoint semantic.

35.1. URI FORMAT

```
kamelet:templateId/routeId[?options]
```

35.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

35.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

35.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

35.3. COMPONENT OPTIONS

The Kamelet component supports 9 options, which are listed below.

Name	Description	Default	Type
location (common)	The location(s) of the Kamelets on the file system. Multiple locations can be set separated by comma.	classpath:/kamelets	String
routeProperties (common)	Set route local parameters.		Map
templateProperties (common)	Set template local parameters.		Map
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
block (producer)	If sending a message to a kamelet endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
timeout (producer)	The timeout value to use if block is enabled.	30000	long
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
routeTemplateLoaderListener (advanced)	**Autowired** To plugin a custom listener for when the Kamelet component is loading Kamelets from external resources.		RouteTemplateLoaderListener

35.4. ENDPOINT OPTIONS

The Kamelet endpoint is configured using URI syntax:

```
kamelet:templateId/routeId
```

with the following path and query parameters:

35.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
templateId (common)	**Required** The Route Template ID.		String
routeId (common)	The Route ID. Default value notice: The ID will be auto-generated if not provided.		String

35.4.2. Query Parameters (8 parameters)

Name	Description	Default	Type
location (common)	Location of the Kamelet to use which can be specified as a resource from file system, classpath etc. The location cannot use wildcards, and must refer to a file including extension, for example file:/etc/foo-kamelet.xml.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
block (producer)	If sending a message to a direct endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	boolean
failIfNoConsumers (producer)	Whether the producer should fail by throwing an exception, when sending to a kamelet endpoint with no active consumers.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
timeout (producer)	The timeout value to use if block is enabled.	30000	long

NOTE

The **kamelet** endpoint is **lenient**, which means that the endpoint accepts additional parameters that are passed to the engine and consumed upon route materialization.

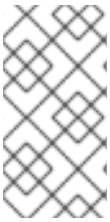
35.5. DISCOVERY

If a Route Template is not found, the **kamelet** endpoint tries to load the related **kamelet** definition from the file system (by default **classpath:/kamelets**). The default resolution mechanism expect kamelet files to have the extension **.kamelet.yaml**.

35.6. SAMPLES

Kamelets can be used as if they were standard Camel components. For example, suppose that we have created a Route Template as follows:

```
routeTemplate("setMyBody")
    .templateParameter("bodyValue")
    .from("kamelet:source")
        .setBody().constant("{{bodyValue}}");
```



NOTE

To let the **Kamelet** component wiring the materialized route to the caller processor, we need to be able to identify the input and output endpoint of the route and this is done by using **kamele:source** to mark the input endpoint and **kamelet:sink** for the output endpoint.

Then the template can be instantiated and invoked as shown below:

```
from("direct:setMyBody")
    .to("kamelet:setMyBody?bodyValue=myKamelet");
```

Behind the scenes, the **Kamelet** component does the following things:

1. It instantiates a route out of the Route Template identified by the given **templateId** path parameter (in this case **setBody**)
2. It will act like the **direct** component and connect the current route to the materialized one.

If you had to do it programmatically, it would have been something like:

```
routeTemplate("setMyBody")
    .templateParameter("bodyValue")
    .from("direct:{{foo}}")
        .setBody().constant("{{bodyValue}}");

TemplatedRouteBuilder.builder(context, "setMyBody")
    .parameter("foo", "bar")
    .parameter("bodyValue", "myKamelet")
    .add();

from("direct:template")
    .to("direct:bar");
```

35.7. SPRING BOOT AUTO-CONFIGURATION

When using kamelet with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

-

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-kamelet-starter</artifactId>
</dependency>
```

The component supports 10 options, which are listed below.

Name	Description	Default	Type
camel.component.kamelet.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.kamelet.block	If sending a message to a kamelet endpoint which has no active consumer, then we can tell the producer to block and wait for the consumer to become active.	true	Boolean
camel.component.kamelet.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.kamelet.enabled	Whether to enable auto configuration of the kamelet component. This is enabled by default.		Boolean
camel.component.kamelet.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.kamelet.location	The location(s) of the Kamelets on the file system. Multiple locations can be set separated by comma.	classpath:/kamelets	String

Name	Description	Default	Type
camel.component.kamelet.route-properties	Set route local parameters.		Map
camel.component.kamelet.route-template-loader-listener	To plugin a custom listener for when the Kamelet component is loading Kamelets from external resources. The option is a org.apache.camel.spi.RouteTemplateLoaderListener type.		RouteTemplateLoaderListener
camel.component.kamelet.template-properties	Set template local parameters.		Map
camel.component.kamelet.timeout	The timeout value to use if block is enabled.	30000	Long

CHAPTER 36. LANGUAGE

Only producer is supported

The Language component allows you to send Exchange to an endpoint which executes a script by any of the supported Languages in Camel. By having a component to execute language scripts, it allows more dynamic routing capabilities. For example by using the Routing Slip or Dynamic Router EIPs you can send messages to language endpoints where the script is dynamic defined as well.

This component is provided out of the box in camel-core and hence no additional JARs is needed. You only have to include additional Camel components if the language of choice mandates it, such as using Groovy or JavaScript languages.

36.1. URI FORMAT

```
language://languageName[:script][?options]
```

You can refer to an external resource for the script using same notation as supported by the other Languages in Camel.

```
language://languageName:resource:scheme:location][?options]
```

36.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

36.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

36.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

36.3. COMPONENT OPTIONS

The Language component supports 2 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

36.4. ENDPOINT OPTIONS

The Language endpoint is configured using URI syntax:

```
language:languageName:resourceUri
```

with the following path and query parameters:

36.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
languageName (producer)	**Required** Sets the name of the language to use. Enum values: • bean • constant • exchangeProperty • file • groovy • header • javascript • jsonpath • mvel • ognl • ref • simple • spel • sql • terser • tokenize • xpath • xquery • xtokenize		String
resourceUri (producer)	Path to the resource, or a reference to lookup a bean in the Registry to use as the resource.		String

36.4.2. Query Parameters (7 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
allowContextMapAll (producer)	Sets whether the context map should allow access to all details. By default only the message body and headers can be accessed. This option can be enabled for full access to the current Exchange and CamelContext. Doing so impose a potential security risk as this opens access to the full power of CamelContext API.	false	boolean
binary (producer)	Whether the script is binary content or text content. By default the script is read as text content (eg java.lang.String).	false	boolean
cacheScript (producer)	Whether to cache the compiled script and reuse Notice reusing the script can cause side effects from processing one Camel org.apache.camel.Exchange to the next org.apache.camel.Exchange.	false	boolean
contentCache (producer)	Sets whether to use resource content cache or not.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
script (producer)	Sets the script to execute.		String
transform (producer)	Whether or not the result of the script should be used as message body. This options is default true.	true	boolean

36.5. MESSAGE HEADERS

The following message headers can be used to affect the behavior of the component

Header	Description
CamelLanguageScript	The script to execute provided in the header. Takes precedence over script configured on the endpoint.

36.6. EXAMPLES

For example you can use the Simple language to Message Translator a message.

You can also provide the script as a header as shown below. Here we use XPath language to extract the text from the <foo> tag.

```
Object out = producer.requestBodyAndHeader("language:xpath", "<foo>Hello World</foo>",
Exchange.LANGUAGE_SCRIPT, "/foo/text()");
assertEquals("Hello World", out);
```

36.7. LOADING SCRIPTS FROM RESOURCES

You can specify a resource uri for a script to load in either the endpoint uri, or in the **Exchange.LANGUAGE_SCRIPT** header. The uri must start with one of the following schemes: file:, classpath:, or http:

By default the script is loaded once and cached. However you can disable the **contentCache** option and have the script loaded on each evaluation. For example if the file myscript.txt is changed on disk, then the updated script is used:

You can refer to the resource similar to the other Languages in Camel by prefixing with "resource:" as shown below.

36.8. SPRING BOOT AUTO-CONFIGURATION

When using language with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-language-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type
camel.component.language.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.language.enabled	Whether to enable auto configuration of the language component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.language.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 37. LOG

Only producer is supported

The Log component logs message exchanges to the underlying logging mechanism.

Camel uses SLF4J which allows you to configure logging via, among others:

- Log4j
- Logback
- Java Util Logging

37.1. URI FORMAT

```
log:loggingCategory[?options]
```

Where **loggingCategory** is the name of the logging category to use. You can append query options to the URI in the following format,

?option=value&option=value&…

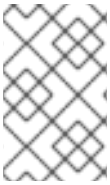
NOTE

Using Logger instance from the Registry
If there's single instance of **org.slf4j.Logger** found in the Registry, the **loggingCategory** is no longer used to create logger instance. The registered instance is used instead. Also it is possible to reference particular **Logger** instance using **?logger=#myLogger** URI parameter. Eventually, if there's no registered and URI **logger** parameter, the logger instance is created using **loggingCategory**.

For example, a log endpoint typically specifies the logging level using the **level** option, as follows:

```
log:org.apache.camel.example?level=DEBUG
```

The default logger logs every exchange (*regular logging*). But Camel also ships with the **Throughput** logger, which is used whenever the **groupSize** option is specified.



NOTE

Also a log in the DSL
There is also a **log** directly in the DSL, but it has a different purpose. Its meant for lightweight and human logs. See more details at LogEIP.

37.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

37.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

37.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

37.3. COMPONENT OPTIONS

The Log component supports 3 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
exchangeFormatter (advanced)	**Autowired** Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter.		ExchangeFormatter

37.4. ENDPOINT OPTIONS

The Log endpoint is configured using URI syntax:

```
log:loggerName
```

with the following path and query parameters:

37.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
loggerName (producer)	**Required** Name of the logging category to use.		String

37.4.2. Query Parameters (27 parameters)

Name	Description	Default	Type
groupActiveOnly (producer)	If true, will hide stats when no new messages have been received for a time interval, if false, show stats regardless of message traffic.	true	Boolean
groupDelay (producer)	Set the initial delay for stats (in millis).		Long
groupInterval (producer)	If specified will group message stats by this time interval (in millis).		Long
groupSize (producer)	An integer that specifies a group size for throughput logging.		Integer

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
level (producer)	Logging level to use. The default value is INFO. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	INFO	String
logMask (producer)	If true, mask sensitive information like password or passphrase in the log.		Boolean
marker (producer)	An optional Marker name to use.		String
exchangeFormatter (advanced)	To use a custom exchange formatter.		ExchangeFormatter
maxChars (formatting)	Limits the number of characters logged per line.	10000	int
multiline (formatting)	If enabled then each information is outputted on a newline.	false	boolean
showAll (formatting)	Quick option for turning all options on. (multiline, maxChars has to be manually set if to be used).	false	boolean
showAllProperties (formatting)	Show all of the exchange properties (both internal and custom).	false	boolean

Name	Description	Default	Type
showBody (formatting)	Show the message body.	true	boolean
showBodyType (formatting)	Show the body Java type.	true	boolean
showCaughtException (formatting)	If the exchange has a caught exception, show the exception message (no stack trace). A caught exception is stored as a property on the exchange (using the key org.apache.camel.Exchange#EXCEPTION_CAUGHT) and for instance a doCatch can catch exceptions.	false	boolean
showException (formatting)	If the exchange has an exception, show the exception message (no stacktrace).	false	boolean
showExchangeId (formatting)	Show the unique exchange ID.	false	boolean
showExchangePattern (formatting)	Shows the Message Exchange Pattern (or MEP for short).	true	boolean
showFiles (formatting)	If enabled Camel will output files.	false	boolean
showFuture (formatting)	If enabled Camel will on Future objects wait for it to complete to obtain the payload to be logged.	false	boolean
showHeaders (formatting)	Show the message headers.	false	boolean
showProperties (formatting)	Show the exchange properties (only custom). Use showAllProperties to show both internal and custom properties.	false	boolean
showStackTrace (formatting)	Show the stack trace, if an exchange has an exception. Only effective if one of showAll, showException or showCaughtException are enabled.	false	boolean
showStreams (formatting)	Whether Camel should show stream bodies or not (eg such as java.io.InputStream). Beware if you enable this option then you may not be able later to access the message body as the stream have already been read by this logger. To remedy this you will have to use Stream Caching.	false	boolean

Name	Description	Default	Type
skipBodyLineSeparator (formatting)	Whether to skip line separators when logging the message body. This allows to log the message body in one line, setting this option to false will preserve any line separators from the body, which then will log the body as is.	true	boolean
style (formatting)	Sets the outputs style to use. Enum values: • Default • Tab • Fixed	Default	OutputStyle

37.5. REGULAR LOGGER SAMPLE

In the route below we log the incoming orders at **DEBUG** level before the order is processed:

```
from("activemq:orders").to("log:com.mycompany.order?level=DEBUG").to("bean:processOrder");
```

Or using Spring XML to define the route:

```
<route>
  <from uri="activemq:orders"/>
  <to uri="log:com.mycompany.order?level=DEBUG"/>
  <to uri="bean:processOrder"/>
</route>
```

37.6. REGULAR LOGGER WITH FORMATTER SAMPLE

In the route below we log the incoming orders at **INFO** level before the order is processed.

```
from("activemq:orders").
    to("log:com.mycompany.order?showAll=true&multiline=true").to("bean:processOrder");
```

37.7. THROUGHPUT LOGGER WITH GROUPSIZE SAMPLE

In the route below we log the throughput of the incoming orders at **DEBUG** level grouped by 10 messages.

```
from("activemq:orders").
    to("log:com.mycompany.order?level=DEBUG&groupSize=10").to("bean:processOrder");
```

37.8. THROUGHPUT LOGGER WITH GROUPINTERVAL SAMPLE

This route will result in message stats logged every 10s, with an initial 60s delay and stats should be displayed even if there isn't any message traffic.

```
from("activemq:orders").
    to("log:com.mycompany.order?
level=DEBUG&groupInterval=10000&groupDelay=60000&groupActiveOnly=false").to("bean:process
Order");
```

The following will be logged:

```
"Received: 1000 new messages, with total 2000 so far. Last group took: 10000 millis which is: 100
messages per second. average: 100"
```

37.9. MASKING SENSITIVE INFORMATION LIKE PASSWORD

You can enable security masking for logging by setting **logMask** flag to **true**. Note that this option also affects Log EIP.

To enable mask in Java DSL at CamelContext level:

```
camelContext.setLogMask(true);
```

And in XML:

```
<camelContext logMask="true">
```

You can also turn it on|off at endpoint level. To enable mask in Java DSL at endpoint level, add logMask=true option in the URI for the log endpoint:

```
from("direct:start").to("log:foo?logMask=true");
```

And in XML:

```
<route>
  <from uri="direct:foo"/>
  <to uri="log:foo?logMask=true"/>
</route>
```

org.apache.camel.support.processor.DefaultMaskingFormatter is used for the masking by default. If you want to use a custom masking formatter, put it into registry with the name **CamelCustomLogMask**. Note that the masking formatter must implement **org.apache.camel.spi.MaskingFormatter**.

37.10. FULL CUSTOMIZATION OF THE LOGGING OUTPUT

With the options outlined in the section, you can control much of the output of the logger. However, log lines will always follow this structure:

```
Exchange[Id:ID-machine-local-50656-1234567901234-1-2, ExchangePattern:InOut,
Properties:{CamelToEndpoint=log://org.apache.camel.component.log.TEST?showAll=true,
CamelCreatedTimestamp=Thu Mar 28 00:00:00 WET 2013},
Headers:{breadcrumbId=ID-machine-local-50656-1234567901234-1-1}, BodyType:String, Body:Hello
World, Out: null]
```

This format is unsuitable in some cases, perhaps because you need to...

- Filter the headers and properties that are printed, to strike a balance between insight and verbosity.
- Adjust the log message to whatever you deem most readable.
- Tailor log messages for digestion by log mining systems, e.g. Splunk.
- Print specific body types differently.

Whenever you require absolute customization, you can create a class that implements the interface. Within the **format(Exchange)** method you have access to the full Exchange, so you can select and extract the precise information you need, format it in a custom manner and return it. The return value will become the final log message.

You can have the Log component pick up your custom **ExchangeFormatter** in either of two ways:

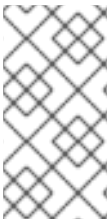
Explicitly instantiating the LogComponent in your Registry:

```
<bean name="log" class="org.apache.camel.component.log.LogComponent">
  <property name="exchangeFormatter" ref="myCustomFormatter" />
</bean>
```

37.10.1. Convention over configuration

Simply by registering a bean with the name **logFormatter**; the Log Component is intelligent enough to pick it up automatically.

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" />
```



NOTE

The **ExchangeFormatter** gets applied to **all Log endpoints within that Camel Context** If you need different ExchangeFormatters for different endpoints, just instantiate the LogComponent as many times as needed, and use the relevant bean name as the endpoint prefix.

When using a custom log formatter, you can specify parameters in the log uri, which gets configured on the custom log formatter. Though when you do that you should define the "logFormatter" as prototype scoped so its not shared if you have different parameters, for example,

```
<bean name="logFormatter" class="com.xyz.MyCustomExchangeFormatter" scope="prototype"/>
```

And then we can have Camel routes using the log uri with different options:

```
<to uri="log:foo?param1=foo&amp;param2=100"/>

<to uri="log:bar?param1=bar&amp;param2=200"/>
```

37.11. SPRING BOOT AUTO-CONFIGURATION

When using log with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-log-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.log.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.log.enabled	Whether to enable auto configuration of the log component. This is enabled by default.		Boolean
camel.component.log.exchange-formatter	Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter. The option is a org.apache.camel.spi.ExchangeFormatter type.		ExchangeFormatter
camel.component.log.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 38. MAIL

Both producer and consumer are supported

The Mail component provides access to Email via Spring's Mail support and the underlying JavaMail system.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

NOTE

POP3 or IMAP
POP3 has some limitations and end users are encouraged to use IMAP if possible.

NOTE

Using mock-mail for testing
You can use a mock framework for unit testing, which allows you to test without the need for a real mail server. However you should remember to not include the mock-mail when you go into production or other environments where you need to send mails to a real mail server. Just the presence of the mock-javamail.jar on the classpath means that it will kick in and avoid sending the mails.

38.1. URI FORMAT

Mail endpoints can have one of the following URI formats (for the protocols, SMTP, POP3, or IMAP, respectively):

```
smtp://[username@]host[:port][?options]
pop3://[username@]host[:port][?options]
imap://[username@]host[:port][?options]
```

The mail component also supports secure variants of these protocols (layered over SSL). You can enable the secure protocols by adding **s** to the scheme:

```
smtps://[username@]host[:port][?options]
pop3s://[username@]host[:port][?options]
imaps://[username@]host[:port][?options]
```

38.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level

- endpoint level

38.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

38.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

38.3. COMPONENT OPTIONS

The Mail component supports 43 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
closeFolder (consumer)	Whether the consumer should close the folder after polling. Setting this option to false and having disconnect=false as well, then the consumer keep the folder open between polls.	true	boolean

Name	Description	Default	Type
copyTo (consumer)	After processing a mail message, it can be copied to a mail folder with the given name. You can override this configuration value, with a header with the key copyTo, allowing you to copy messages to folder names configured at runtime.		String
decodeFilename (consumer)	If set to true, the MimeUtility.decodeText method will be used to decode the filename. This is similar to setting JVM system property mail.mime.encodefilename.	false	boolean
delete (consumer)	Deletes the messages after they have been processed. This is done by setting the DELETED flag on the mail message. If false, the SEEN flag is set instead. As of Camel 2.10 you can override this configuration option by setting a header with the key delete to determine if the mail should be deleted or not.	false	boolean
disconnect (consumer)	Whether the consumer should disconnect after polling. If enabled this forces Camel to connect on each poll.	false	boolean
handleFailedMessage (consumer)	If the mail consumer cannot retrieve a given mail message, then this option allows to handle the caused exception by the consumer's error handler. By enable the bridge error handler on the consumer, then the Camel routing error handler can handle the exception instead. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	boolean
mimeDecodeHeaders (consumer)	This option enables transparent MIME decoding and unfolding for mail headers.	false	boolean
moveTo (consumer)	After processing a mail message, it can be moved to a mail folder with the given name. You can override this configuration value, with a header with the key moveTo, allowing you to move messages to folder names configured at runtime.		String
peek (consumer)	Will mark the javax.mail.Message as peeked before processing the mail message. This applies to IMAPMessage messages types only. By using peek the mail will not be eager marked as SEEN on the mail server, which allows us to rollback the mail message if there is an error processing in Camel.	true	boolean

Name	Description	Default	Type
skipFailedMessage (consumer)	If the mail consumer cannot retrieve a given mail message, then this option allows to skip the message and move on to retrieve the next mail message. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	boolean
unseen (consumer)	Whether to limit by unseen mails only.	true	boolean
fetchSize (consumer (advanced))	Sets the maximum number of messages to consume during a poll. This can be used to avoid overloading a mail server, if a mailbox folder contains a lot of messages. Default value of -1 means no fetch size and all messages will be consumed. Setting the value to 0 is a special corner case, where Camel will not consume any messages at all.	-1	int
folderName (consumer (advanced))	The folder to poll.	INBOX	String
mapMailMessage (consumer (advanced))	Specifies whether Camel should map the received mail message to Camel body/headers/attachments. If set to true, the body of the mail message is mapped to the body of the Camel IN message, the mail headers are mapped to IN headers, and the attachments to Camel IN attachment message. If this option is set to false then the IN message contains a raw javax.mail.Message. You can retrieve this raw message by calling exchange.getIn().getBody(javax.mail.Message.class).	true	boolean
bcc (producer)	Sets the BCC email address. Separate multiple email addresses with comma.		String
cc (producer)	Sets the CC email address. Separate multiple email addresses with comma.		String
from (producer)	The from email address.	camel@localhost	String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
replyTo (producer)	The Reply-To recipients (the receivers of the response mail). Separate multiple email addresses with a comma.		String
subject (producer)	The Subject of the message being sent. Note: Setting the subject in the header takes precedence over this option.		String
to (producer)	Sets the To email address. Separate multiple email addresses with comma.		String
javaMailSender (producer (advanced))	To use a custom org.apache.camel.component.mail.JavaMailSender for sending emails.		JavaMailSender
additionalJavaMailProperties (advanced)	Sets additional java mail properties, that will append/override any default properties that is set based on all the other options. This is useful if you need to add some special options but want to keep the others as is.		Properties
alternativeBodyHeader (advanced)	Specifies the key to an IN message header that contains an alternative email body. For example, if you send emails in text/html format and want to provide an alternative mail body for non-HTML email clients, set the alternative mail body with this key as a header.	CamelMailAlternativeBody	String
attachmentsContentTransferEncodingResolver (advanced)	To use a custom AttachmentsContentTransferEncodingResolver to resolve what content-type-encoding to use for attachments.		AttachmentsContentTransferEncodingResolver

Name	Description	Default	Type
authenticator (advanced)	The authenticator for login. If set then the password and username are ignored. Can be used for tokens which can expire and therefore must be read dynamically.		MailAuthenticator
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
configuration (advanced)	Sets the Mail configuration.		MailConfiguration
connectionTimeout (advanced)	The connection timeout in milliseconds.	30000	int
contentType (advanced)	The mail message content type. Use text/html for HTML mails.	text/plain	String
contentTypeResolver (advanced)	Resolver to determine Content-Type for file attachments.		ContentTypeResolver
debugMode (advanced)	Enable debug mode on the underlying mail framework. The SUN Mail framework logs the debug messages to System.out by default.	false	boolean
ignoreUnsupportedCharset (advanced)	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	boolean
ignoreUriScheme (advanced)	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	boolean
javaMailProperties (advanced)	Sets the java mail options. Will clear any default properties and only use the properties provided for this method.		Properties

Name	Description	Default	Type
session (advanced)	Specifies the mail session that camel should use for all mail interactions. Useful in scenarios where mail sessions are created and managed by some other resource, such as a JavaEE container. When using a custom mail session, then the hostname and port from the mail session will be used (if configured on the session).		Session
useInlineAttachments (advanced)	Whether to use disposition inline or attachment.	false	boolean
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
password (security)	The password for login. See also setAuthenticator(MailAuthenticator).		String
sslContextParameters (security)	To configure security using SSLContextParameters.		SSLContextParameters
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean
username (security)	The username for login. See also setAuthenticator(MailAuthenticator).		String

38.4. ENDPOINT OPTIONS

The Mail endpoint is configured using URI syntax:

```
imap:host:port
```

with the following path and query parameters:

38.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
host (common)	**Required** The mail server host name.		String
port (common)	The port number of the mail server.		int

38.4.2. Query Parameters (66 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
closeFolder (consumer)	Whether the consumer should close the folder after polling. Setting this option to false and having disconnect=false as well, then the consumer keep the folder open between polls.	true	boolean
copyTo (consumer)	After processing a mail message, it can be copied to a mail folder with the given name. You can override this configuration value, with a header with the key copyTo, allowing you to copy messages to folder names configured at runtime.		String
decodeFilename (consumer)	If set to true, the MimeUtility.decodeText method will be used to decode the filename. This is similar to setting JVM system property mail.mime.encodefilename.	false	boolean
delete (consumer)	Deletes the messages after they have been processed. This is done by setting the DELETED flag on the mail message. If false, the SEEN flag is set instead. As of Camel 2.10 you can override this configuration option by setting a header with the key delete to determine if the mail should be deleted or not.	false	boolean
disconnect (consumer)	Whether the consumer should disconnect after polling. If enabled this forces Camel to connect on each poll.	false	boolean
handleFailedMessage (consumer)	If the mail consumer cannot retrieve a given mail message, then this option allows to handle the caused exception by the consumer's error handler. By enable the bridge error handler on the consumer, then the Camel routing error handler can handle the exception instead. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	boolean

Name	Description	Default	Type
maxMessagesPerPoll (consumer)	Specifies the maximum number of messages to gather per poll. By default, no maximum is set. Can be used to set a limit of e.g. 1000 to avoid downloading thousands of files when the server starts up. Set a value of 0 or negative to disable this option.		int
mimeDecodeHeaders (consumer)	This option enables transparent MIME decoding and unfolding for mail headers.	false	boolean
moveTo (consumer)	After processing a mail message, it can be moved to a mail folder with the given name. You can override this configuration value, with a header with the key moveTo, allowing you to move messages to folder names configured at runtime.		String
peek (consumer)	Will mark the javax.mail.Message as peeked before processing the mail message. This applies to IMAPMessage messages types only. By using peek the mail will not be eager marked as SEEN on the mail server, which allows us to rollback the mail message if there is an error processing in Camel.	true	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
skipFailedMessage (consumer)	If the mail consumer cannot retrieve a given mail message, then this option allows to skip the message and move on to retrieve the next mail message. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	boolean
unseen (consumer)	Whether to limit by unseen mails only.	true	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
fetchSize (consumer (advanced))	Sets the maximum number of messages to consume during a poll. This can be used to avoid overloading a mail server, if a mailbox folder contains a lot of messages. Default value of -1 means no fetch size and all messages will be consumed. Setting the value to 0 is a special corner case, where Camel will not consume any messages at all.	-1	int
folderName (consumer (advanced))	The folder to poll.	INBOX	String
mailUidGenerator (consumer (advanced))	A pluggable MailUidGenerator that allows to use custom logic to generate UUID of the mail message.		MailUidGenerator
mapMailMessage (consumer (advanced))	Specifies whether Camel should map the received mail message to Camel body/headers/attachments. If set to true, the body of the mail message is mapped to the body of the Camel IN message, the mail headers are mapped to IN headers, and the attachments to Camel IN attachment message. If this option is set to false then the IN message contains a raw javax.mail.Message. You can retrieve this raw message by calling exchange.getIn().getBody(javax.mail.Message.class).	true	boolean
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
postProcessAction (consumer (advanced))	Refers to an MailBoxPostProcessAction for doing post processing tasks on the mailbox once the normal processing ended.		MailBoxPostProcessAction

Name	Description	Default	Type
bcc (producer)	Sets the BCC email address. Separate multiple email addresses with comma.		String
cc (producer)	Sets the CC email address. Separate multiple email addresses with comma.		String
from (producer)	The from email address.	camel@localhost	String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
replyTo (producer)	The Reply-To recipients (the receivers of the response mail). Separate multiple email addresses with a comma.		String
subject (producer)	The Subject of the message being sent. Note: Setting the subject in the header takes precedence over this option.		String
to (producer)	Sets the To email address. Separate multiple email addresses with comma.		String
javaMailSender (producer (advanced))	To use a custom org.apache.camel.component.mail.JavaMailSender for sending emails.		JavaMailSender
additionalJavaMailProperties (advanced)	Sets additional java mail properties, that will append/override any default properties that is set based on all the other options. This is useful if you need to add some special options but want to keep the others as is.		Properties

Name	Description	Default	Type
alternativeBodyHeader (advanced)	Specifies the key to an IN message header that contains an alternative email body. For example, if you send emails in text/html format and want to provide an alternative mail body for non-HTML email clients, set the alternative mail body with this key as a header.	CamelMailAlternativeBody	String
attachmentsContentTransferEncodingResolver (advanced)	To use a custom AttachmentsContentTransferEncodingResolver to resolve what content-type-encoding to use for attachments.		AttachmentsContentTransferEncodingResolver
authenticator (advanced)	The authenticator for login. If set then the password and username are ignored. Can be used for tokens which can expire and therefore must be read dynamically.		MailAuthenticator
binding (advanced)	Sets the binding used to convert from a Camel message to and from a Mail message.		MailBinding
connectionTimeout (advanced)	The connection timeout in milliseconds.	30000	int
contentType (advanced)	The mail message content type. Use text/html for HTML mails.	text/plain	String
contentTypeResolver (advanced)	Resolver to determine Content-Type for file attachments.		ContentTypeResolver
debugMode (advanced)	Enable debug mode on the underlying mail framework. The SUN Mail framework logs the debug messages to System.out by default.	false	boolean
headerFilterStrategy (advanced)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter headers.		HeaderFilterStrategy
ignoreUnsupportedCharset (advanced)	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	boolean

Name	Description	Default	Type
ignoreUriScheme (advanced)	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	boolean
javaMailProperties (advanced)	Sets the java mail options. Will clear any default properties and only use the properties provided for this method.		Properties
session (advanced)	Specifies the mail session that camel should use for all mail interactions. Useful in scenarios where mail sessions are created and managed by some other resource, such as a JavaEE container. When using a custom mail session, then the hostname and port from the mail session will be used (if configured on the session).		Session
useInlineAttachments (advanced)	Whether to use disposition inline or attachment.	false	boolean
idempotentRepository (filter)	A pluggable repository org.apache.camel.spi.IdempotentRepository which allows to cluster consuming from the same mailbox, and let the repository coordinate whether a mail message is valid for the consumer to process. By default no repository is in use.		IdempotentRepository
idempotentRepositoryRemoveOnCommit (filter)	When using idempotent repository, then when the mail message has been successfully processed and is committed, should the message id be removed from the idempotent repository (default) or be kept in the repository. By default its assumed the message id is unique and has no value to be kept in the repository, because the mail message will be marked as seen/moved or deleted to prevent it from being consumed again. And therefore having the message id stored in the idempotent repository has little value. However this option allows to store the message id, for whatever reason you may have.	true	boolean
searchTerm (filter)	Refers to a javax.mail.search.SearchTerm which allows to filter mails based on search criteria such as subject, body, from, sent after a certain date etc.		SearchTerm
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	60000	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object

Name	Description	Default	Type
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
password (security)	The password for login. See also setAuthenticator(MailAuthenticator).		String
sslContextParameters (security)	To configure security using SSLContextParameters.		SSLContextParameters
username (security)	The username for login. See also setAuthenticator(MailAuthenticator).		String
sortTerm (sort)	Sorting order for messages. Only natively supported for IMAP. Emulated to some degree when using POP3 or when IMAP server does not have the SORT capability.		SortTerm[]

38.4.3. Sample endpoints

Typically, you specify a URI with login credentials as follows (taking SMTP as an example):

```
smtp://[username@]host[:port][?password=somepwd]
```

Alternatively, it is possible to specify both the user name and the password as query options:

```
smtp://host[:port]?password=somepwd&username=someuser
```

For example:

```
smtp://mycompany.mailserver:30?password=tiger&username=scott
```

38.4.4. Component alias names

- IMAP
- IMAPs
- POP3s
- SMTP
- SMTPs

38.4.5. Default ports

Default port numbers are supported. If the port number is omitted, Camel determines the port number to use based on the protocol.

Protocol	Default Port Number
SMTP	**25**
SMTPS	**465**
POP3	**110**
POP3S	**995**
IMAP	**143**
IMAPS	**993**

38.5. SSL SUPPORT

The underlying mail framework is responsible for providing SSL support. You may either configure SSL/TLS support by completely specifying the necessary Java Mail API configuration options, or you may provide a configured SSLContextParameters through the component or endpoint configuration.

38.5.1. Using the JSSE Configuration Utility

The mail component supports SSL/TLS configuration through the Camel JSSE Configuration Utility. This utility greatly decreases the amount of component specific code you need to write and is

configurable at the endpoint and component levels. The following examples demonstrate how to use the utility with the mail component.

Programmatic configuration of the endpoint

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/truststore.jks");
ksp.setPassword("keystorePassword");
TrustManagersParameters tmp = new TrustManagersParameters();
tmp.setKeyStore(ksp);
SSLContextParameters scp = new SSLContextParameters();
scp.setTrustManagers(tmp);
Registry registry = ...
registry.bind("sslContextParameters", scp);
...
from(...)
    .to("smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters");
```

Spring DSL based configuration of endpoint

```
...
<camel:sslContextParameters id="sslContextParameters">
  <camel:trustManagers>
    <camel:keyStore resource="/users/home/server/truststore.jks" password="keystorePassword"/>
  </camel:trustManagers>
</camel:sslContextParameters>...
...
<to uri="smtps://smtp.google.com?
username=user@gmail.com&password=password&sslContextParameters=#sslContextParameters"/
>...
```

38.5.2. Configuring JavaMail Directly

Camel uses Jakarta JavaMail, which only trusts certificates issued by well known Certificate Authorities (the default JVM trust configuration). If you issue your own certificates, you have to import the CA certificates into the JVM's Java trust/key store files, override the default JVM trust/key store files (see **SSLNOTES.txt** in JavaMail for details).

38.6. MAIL MESSAGE CONTENT

Camel uses the message exchange's IN body as the MimeMessage text content. The body is converted to String.class.

Camel copies all of the exchange's IN headers to the MimeMessage headers.

The subject of the MimeMessage can be configured using a header property on the IN message. The code below demonstrates this:

The same applies for other MimeMessage headers such as recipients, so you can use a header property as To:

When using the MailProducer the send the mail to server, you should be able to get the message id of the MimeMessage with the key CamelMailMessageId from the Camel message header.

38.7. HEADERS TAKE PRECEDENCE OVER PRE-CONFIGURED RECIPIENTS

The recipients specified in the message headers always take precedence over recipients pre-configured in the endpoint URI. The idea is that if you provide any recipients in the message headers, that is what you get. The recipients pre-configured in the endpoint URI are treated as a fallback.

In the sample code below, the email message is sent to **davsclaus@apache.org**, because it takes precedence over the pre-configured recipient, **info@mycompany.com**. Any **CC** and **BCC** settings in the endpoint URI are also ignored and those recipients will not receive any mail. The choice between headers and pre-configured settings is all or nothing: the mail component *either* takes the recipients exclusively from the headers or exclusively from the pre-configured settings. It is not possible to mix and match headers and pre-configured settings.

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("to", "davsclaus@apache.org");

template.sendBodyAndHeaders("smtp://admin@localhost?to=info@mycompany.com", "Hello World",
headers);
```

38.8. MULTIPLE RECIPIENTS FOR EASIER CONFIGURATION

It is possible to set multiple recipients using a comma-separated or a semicolon-separated list. This applies both to header settings and to settings in an endpoint URI. For example:

```
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("to", "davsclaus@apache.org ; jstrachan@apache.org ; ningjiang@apache.org");
```

The preceding example uses a semicolon, **;**, as the separator character.

38.9. SETTING SENDER NAME AND EMAIL

You can specify recipients in the format, **name <email>**, to include both the name and the email address of the recipient.

For example, you define the following headers on the a Message:

```
Map headers = new HashMap();
map.put("To", "Claus Ibsen <davsclaus@apache.org>");
map.put("From", "James Strachan <jstrachan@apache.org>");
map.put("Subject", "Camel is cool");
```

38.10. JAVAMAIL API (EX SUN JAVAMAIL)

JavaMail API is used under the hood for consuming and producing mails. We encourage end-users to consult these references when using either POP3 or IMAP protocol. Note particularly that POP3 has a much more limited set of features than IMAP.

- JavaMail POP3 API
- JavaMail IMAP API
- And generally about the MAIL Flags

38.11. SAMPLES

We start with a simple route that sends the messages received from a JMS queue as emails. The email account is the **admin** account on **mymailserver.com**.

```
from("jms://queue:subscription").to("smtp://admin@mymailserver.com?password=secret");
```

In the next sample, we poll a mailbox for new emails once every minute.

```
from("imap://admin@mymailserver.com?password=secret&unseen=true&delay=60000")
    .to("seda://mails");
```

38.12. SENDING MAIL WITH ATTACHMENT SAMPLE

NOTE

Attachments are not support by all Camel components
The *Attachments API* is based on the Java Activation Framework and is generally only used by the Mail API. Since many of the other Camel components do not support attachments, the attachments could potentially be lost as they propagate along the route. The rule of thumb, therefore, is to add attachments just before sending a message to the mail endpoint.

The mail component supports attachments. In the sample below, we send a mail message containing a plain text message with a logo file attachment.

38.13. SSL SAMPLE

In this sample, we want to poll our Google mail inbox for mails. To download mail onto a local mail client, Google mail requires you to enable and configure SSL. This is done by logging into your Google mail account and changing your settings to allow IMAP access. Google have extensive documentation on how to do this.

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&delay=60000").to("log:newmail");
```

The preceding route polls the Google mail inbox for new mails once every minute and logs the received messages to the **newmail** logger category.
Running the sample with **DEBUG** logging enabled, we can monitor the progress in the logs:

```
2008-05-08 06:32:09,640 DEBUG MailConsumer - Connecting to MailStore
imaps//imap.gmail.com:993 (SSL enabled), folder=INBOX
2008-05-08 06:32:11,203 DEBUG MailConsumer - Polling mailfolder: imaps//imap.gmail.com:993
(SSL enabled), folder=INBOX
2008-05-08 06:32:11,640 DEBUG MailConsumer - Fetching 1 messages. Total 1 messages.
2008-05-08 06:32:12,171 DEBUG MailConsumer - Processing message: messageNumber=[332],
from=[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
2008-05-08 06:32:12,187 INFO  newmail - Exchange[MailMessage: messageNumber=[332], from=
[James Bond <007@mi5.co.uk>], to=YOUR_USERNAME@gmail.com], subject=[...
```

38.14. CONSUMING MAILS WITH ATTACHMENT SAMPLE

In this sample we poll a mailbox and store all attachments from the mails as files. First, we define a route to poll the mailbox. As this sample is based on google mail, it uses the same route as shown in the SSL sample:

```
from("imaps://imap.gmail.com?
username=YOUR_USERNAME@gmail.com&password=YOUR_PASSWORD"
    + "&delete=false&unseen=true&delay=60000").process(new MyMailProcessor());
```

Instead of logging the mail we use a processor where we can process the mail from java code:

```
public void process(Exchange exchange) throws Exception {
    // the API is a bit clunky so we need to loop
    AttachmentMessage attachmentMessage = exchange.getMessage(AttachmentMessage.class);
    Map<String, DataHandler> attachments = attachmentMessage.getAttachments();
    if (attachments.size() > 0) {
        for (String name : attachments.keySet()) {
            DataHandler dh = attachments.get(name);
            // get the file name
            String filename = dh.getName();

            // get the content and convert it to byte[]
            byte[] data = exchange.getContext().getTypeConverter()
                            .convertTo(byte[].class, dh.getInputStream());

            // write the data to a file
            FileOutputStream out = new FileOutputStream(filename);
            out.write(data);
            out.flush();
            out.close();
        }
    }
}
```

As you can see the API to handle attachments is a bit clunky but it's there so you can get the **javax.activation.DataHandler** so you can handle the attachments using standard API.

38.15. HOW TO SPLIT A MAIL MESSAGE WITH ATTACHMENTS

In this example we consume mail messages which may have a number of attachments. What we want to do is to use the Splitter EIP per individual attachment, to process the attachments separately. For example if the mail message has 5 attachments, we want the Splitter to process five messages, each having a single attachment. To do this we need to provide a custom Expression to the Splitter where we provide a List<Message> that contains the five messages with the single attachment.

The code is provided out of the box in Camel 2.10 onwards in the **camel-mail** component. The code is in the class: **org.apache.camel.component.mail.SplitAttachmentsExpression**, which you can find in the source code here.

In the Camel route you then need to use this Expression in the route as shown below:

If you use XML DSL then you need to declare a method call expression in the Splitter as shown below

```
<split>
  <method beanType="org.apache.camel.component.mail.SplitAttachmentsExpression"/>
  <to uri="mock:split"/>
</split>
```

You can also split the attachments as byte[] to be stored as the message body. This is done by creating the expression with boolean true

```
SplitAttachmentsExpression split = SplitAttachmentsExpression(true);
```

And then use the expression with the splitter EIP.

38.16. USING CUSTOM SEARCHTERM

You can configure a **searchTerm** on the **MailEndpoint** which allows you to filter out unwanted mails.

For example to filter mails to contain Camel in either Subject or Text you can do as follows:

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subjectOrBody=Camel"/>
  <to uri="bean:myBean"/>
</route>
```

Notice we use the **"searchTerm.subjectOrBody"** as parameter key to indicate that we want to search on mail subject or body, to contain the word "Camel".
The class **org.apache.camel.component.mail.SimpleSearchTerm** has a number of options you can configure:

Or to get the new unseen emails going 24 hours back in time you can do. Notice the "now-24h" syntax. See the table below for more details.

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.fromSentDate=now-24h"/>
  <to uri="bean:myBean"/>
</route>
```

You can have multiple searchTerm in the endpoint uri configuration. They would then be combined together using AND operator, eg so both conditions must match. For example to get the last unseen emails going back 24 hours which has Camel in the mail subject you can do:

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm.subject=Camel&searchTerm.fromSentDate=now-
24h"/>
  <to uri="bean:myBean"/>
</route>
```

The **SimpleSearchTerm** is designed to be easily configurable from a POJO, so you can also configure it using a <bean> style in XML

```
<bean id="mySearchTerm" class="org.apache.camel.component.mail.SimpleSearchTerm">
```

```
  <property name="subject" value="Order"/>
  <property name="to" value="acme-order@acme.com"/>
  <property name="fromSentDate" value="now"/>
 </bean>
```

You can then refer to this bean, using #beanId in your Camel route as shown:

```
<route>
  <from uri="imaps://mymailseerver?
username=foo&password=secret&searchTerm=#mySearchTerm"/>
  <to uri="bean:myBean"/>
</route>
```

In Java there is a builder class to build compound **SearchTerms** using the **org.apache.camel.component.mail.SearchTermBuilder** class. This allows you to build complex terms such as:

```
// we just want the unseen mails which is not spam
SearchTermBuilder builder = new SearchTermBuilder();

builder.unseen().body(Op.not, "Spam").subject(Op.not, "Spam")
  // which was sent from either foo or bar
  .from("foo@somewhere.com").from(Op.or, "bar@somewhere.com");
  // .. and we could continue building the terms

SearchTerm term = builder.build();
```

38.17. POLLING OPTIMIZATION

The parameter maxMessagePerPoll and fetchSize allow you to restrict the number message that should be processed for each poll. These parameters should help to prevent bad performance when working with folders that contain a lot of messages. In previous versions these parameters have been evaluated too late, so that big mailboxes could still cause performance problems. With Camel 3.1 these parameters are evaluated earlier during the poll to avoid these problems.

38.18. USING HEADERS WITH ADDITIONAL JAVA MAIL SENDER PROPERTIES

When sending mails, then you can provide dynamic java mail properties for the **JavaMailSender** from the Exchange as message headers with keys starting with **java.smtp.**.

You can set any of the **java.smtp** properties which you can find in the Java Mail documentation.

For example to provide a dynamic uuid in **java.smtp.from** (SMTP MAIL command):

```
  .setHeader("from", constant("reply2me@foo.com"));
  .setHeader("java.smtp.from", method(UUID.class, "randomUUID"));
  .to("smtp://mymailserver:1234");
```

NOTE

This is only supported when **not** using a custom **JavaMailSender**.

38.19. SPRING BOOT AUTO-CONFIGURATION

When using imap with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-mail-starter</artifactId>
</dependency>
```

The component supports 50 options, which are listed below.

Name	Description	Default	Type
camel.component.mail.additional-java-mail-properties	Sets additional java mail properties, that will append/override any default properties that is set based on all the other options. This is useful if you need to add some special options but want to keep the others as is. The option is a java.util.Properties type.		Properties
camel.component.mail.alternative-body-header	Specifies the key to an IN message header that contains an alternative email body. For example, if you send emails in text/html format and want to provide an alternative mail body for non-HTML email clients, set the alternative mail body with this key as a header.	CamelMailAlternativeBody	String
camel.component.mail.attachments-content-transfer-encoding-resolver	To use a custom AttachmentsContentTransferEncodingResolver to resolve what content-type-encoding to use for attachments. The option is a org.apache.camel.component.mail.AttachmentsContentTransferEncodingResolver type.		AttachmentsContentTransferEncodingResolver
camel.component.mail.authenticator	The authenticator for login. If set then the password and username are ignored. Can be used for tokens which can expire and therefore must be read dynamically. The option is a org.apache.camel.component.mail.MailAuthenticator type.		MailAuthenticator
camel.component.mail.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.mail.bcc	Sets the BCC email address. Separate multiple email addresses with comma.		String
camel.component.mail.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.mail.cc	Sets the CC email address. Separate multiple email addresses with comma.		String
camel.component.mail.close-folder	Whether the consumer should close the folder after polling. Setting this option to false and having disconnect=false as well, then the consumer keep the folder open between polls.	true	Boolean
camel.component.mail.configuration	Sets the Mail configuration. The option is a org.apache.camel.component.mail.MailConfiguration type.		MailConfiguration
camel.component.mail.connection-timeout	The connection timeout in milliseconds.	30000	Integer
camel.component.mail.content-type	The mail message content type. Use text/html for HTML mails.	text/plain	String
camel.component.mail.content-type-resolver	Resolver to determine Content-Type for file attachments. The option is a org.apache.camel.component.mail.ContentTypeResolver type.		ContentTypeResolver
camel.component.mail.copy-to	After processing a mail message, it can be copied to a mail folder with the given name. You can override this configuration value, with a header with the key copyTo, allowing you to copy messages to folder names configured at runtime.		String
camel.component.mail.debug-mode	Enable debug mode on the underlying mail framework. The SUN Mail framework logs the debug messages to System.out by default.	false	Boolean

Name	Description	Default	Type
camel.component.mail.decode-filename	If set to true, the MimeUtility.decodeText method will be used to decode the filename. This is similar to setting JVM system property mail.mime.encodefilename.	false	Boolean
camel.component.mail.delete	Deletes the messages after they have been processed. This is done by setting the DELETED flag on the mail message. If false, the SEEN flag is set instead. As of Camel 2.10 you can override this configuration option by setting a header with the key delete to determine if the mail should be deleted or not.	false	Boolean
camel.component.mail.disconnect	Whether the consumer should disconnect after polling. If enabled this forces Camel to connect on each poll.	false	Boolean
camel.component.mail.enabled	Whether to enable auto configuration of the mail component. This is enabled by default.		Boolean
camel.component.mail.fetch-size	Sets the maximum number of messages to consume during a poll. This can be used to avoid overloading a mail server, if a mailbox folder contains a lot of messages. Default value of -1 means no fetch size and all messages will be consumed. Setting the value to 0 is a special corner case, where Camel will not consume any messages at all.	-1	Integer
camel.component.mail.folder-name	The folder to poll.	INBOX	String
camel.component.mail.from	The from email address.	camel@localhost	String
camel.component.mail.handle-failed-message	If the mail consumer cannot retrieve a given mail message, then this option allows to handle the caused exception by the consumer's error handler. By enable the bridge error handler on the consumer, then the Camel routing error handler can handle the exception instead. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	Boolean

Name	Description	Default	Type
camel.component.mail.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.mail.ignore-unsupported-charset	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	Boolean
camel.component.mail.ignore-uri-scheme	Option to let Camel ignore unsupported charset in the local JVM when sending mails. If the charset is unsupported then charset=XXX (where XXX represents the unsupported charset) is removed from the content-type and it relies on the platform default instead.	false	Boolean
camel.component.mail.java-mail-properties	Sets the java mail options. Will clear any default properties and only use the properties provided for this method. The option is a java.util.Properties type.		Properties
camel.component.mail.java-mail-sender	To use a custom org.apache.camel.component.mail.JavaMailSender for sending emails. The option is a org.apache.camel.component.mail.JavaMailSender type.		JavaMailSender
camel.component.mail.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.mail.map-mail-message	Specifies whether Camel should map the received mail message to Camel body/headers/attachments. If set to true, the body of the mail message is mapped to the body of the Camel IN message, the mail headers are mapped to IN headers, and the attachments to Camel IN attachment message. If this option is set to false then the IN message contains a raw javax.mail.Message. You can retrieve this raw message by calling exchange.getIn().getBody(javax.mail.Message.class).	true	Boolean
camel.component.mail.mime-decode-headers	This option enables transparent MIME decoding and unfolding for mail headers.	false	Boolean
camel.component.mail.move-to	After processing a mail message, it can be moved to a mail folder with the given name. You can override this configuration value, with a header with the key moveTo, allowing you to move messages to folder names configured at runtime.		String
camel.component.mail.password	The password for login. See also setAuthenticator(MailAuthenticator).		String
camel.component.mail.peek	Will mark the javax.mail.Message as peeked before processing the mail message. This applies to IMAPMessage messages types only. By using peek the mail will not be eager marked as SEEN on the mail server, which allows us to rollback the mail message if there is an error processing in Camel.	true	Boolean
camel.component.mail.reply-to	The Reply-To recipients (the receivers of the response mail). Separate multiple email addresses with a comma.		String
camel.component.mail.session	Specifies the mail session that camel should use for all mail interactions. Useful in scenarios where mail sessions are created and managed by some other resource, such as a JavaEE container. When using a custom mail session, then the hostname and port from the mail session will be used (if configured on the session). The option is a javax.mail.Session type.		Session

Name	Description	Default	Type
camel.component.mail.skip-failed-message	If the mail consumer cannot retrieve a given mail message, then this option allows to skip the message and move on to retrieve the next mail message. The default behavior would be the consumer throws an exception and no mails from the batch would be able to be routed by Camel.	false	Boolean
camel.component.mail.ssl-context-parameters	To configure security using SSLContextParameters. The option is a org.apache.camel.support.jsse.SSLContextParameters type.		SSLContextParameters
camel.component.mail.subject	The Subject of the message being sent. Note: Setting the subject in the header takes precedence over this option.		String
camel.component.mail.to	Sets the To email address. Separate multiple email addresses with comma.		String
camel.component.mail.unseen	Whether to limit by unseen mails only.	true	Boolean
camel.component.mail.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.mail.use-inline-attachments	Whether to use disposition inline or attachment.	false	Boolean
camel.component.mail.username	The username for login. See also setAuthenticator(MailAuthenticator).		String
camel.dataformat.mime-multipart.binary-content	Defines whether the content of binary parts in the MIME multipart is binary (true) or Base-64 encoded (false) Default is false.	false	Boolean
camel.dataformat.mime-multipart.enabled	Whether to enable auto configuration of the mime-multipart data format. This is enabled by default.		Boolean
camel.dataformat.mime-multipart.headers-inline	Defines whether the MIME-Multipart headers are part of the message body (true) or are set as Camel headers (false). Default is false.	false	Boolean

Name	Description	Default	Type
camel.dataformat.mime-multipart.include-headers	A regex that defines which Camel headers are also included as MIME headers into the MIME multipart. This will only work if headersInline is set to true. Default is to include no headers.		String
camel.dataformat.mime-multipart.multipart-sub-type	Specify the subtype of the MIME Multipart. Default is mixed.	mixed	String
camel.dataformat.mime-multipart.multipart-without-attachment	Defines whether a message without attachment is also marshaled into a MIME Multipart (with only one body part). Default is false.	false	Boolean

CHAPTER 39. MAIL MICROSOFT OAUTH

Since Camel 3.18.4.

The Mail Microsoft OAuth2 provides an implementation of **org.apache.camel.component.mail.MailAuthenticator** to authenticate IMAP/POP/SMTP connections and access to Email via Spring's Mail support and the underlying JavaMail system.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mail-microsoft-oauth</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Importing **camel-mail-microsoft-oauth** will automatically import camel-mail component.

39.1. MICROSOFT EXCHANGE ONLINE OAUTH2 MAIL AUTHENTICATOR IMAP SAMPLE

To use OAuth, an application must be registered with Azure Active Directory. Follow the instructions to register a new application.

Procedure

1. Enable the application to access Exchange mailboxes via client credentials flow. For more information, see Authenticate an IMAP, POP or SMTP connection using OAuth

2. Once everything is set up, declare and register in the registry, an instance of **org.apache.camel.component.mail.MicrosoftExchangeOnlineOAuth2MailAuthenticator**.

3. For Example, in a Spring Boot application:

```
@BindToRegistry("auth")
public MicrosoftExchangeOnlineOAuth2MailAuthenticator exchangeAuthenticator(){
   return new MicrosoftExchangeOnlineOAuth2MailAuthenticator(tenantId, clientId, clientSecret,
"jon@doe.com");
}
```

1. Then reference it in the Camel URI as follows:

```
 from("imaps://outlook.office365.com:993"
                + "?authenticator=#auth"
                + "&mail.imaps.auth.mechanisms=XOAUTH2"
                + "&debugMode=true"
                + "&delete=false")
```

CHAPTER 40. MAPSTRUCT

Since Camel 3.19

Only producer is supported.

The camel-mapstruct component is used for converting POJOs using .

40.1. URI FORMAT

```
mapstruct:className[?options]
```

Where **className** is the fully qualified class name of the POJO to convert to.

40.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

40.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

40.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

40.3. COMPONENT OPTIONS

The MapStruct component supports 4 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
mapperPackageName (producer)	**Required** Package name(s) where Camel should discover Mapstruct mapping classes. Multiple package names can be separated by comma.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
mapStructConverter (advanced)	**Autowired** To use a custom MapStructConverter such as adapting to a special runtime.		MapStructMapperFinder

40.4. ENDPOINT OPTIONS

The MapStruct endpoint is configured using URI syntax:

```
mapstruct:className
```

with the following path and query parameters:

40.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
className (producer)	**Required** The fully qualified class name of the POJO that mapstruct should convert to (target).		String

40.4.2. Query Parameters (2 parameters)

Name	Description	Default	Type
mandatory (producer)	Whether there must exist a mapstruct converter to convert to the POJO.	true	boolean
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

40.5. SETTING UP MAPSTRUCT

The camel-mapstruct component must be configured with one or more package names, for classpath scanning MapStruct **Mapper** classes. This is needed because the **Mapper** classes are to be used for converting POJOs with MapStruct.

For example, to set up two packages you can do as following:

```
MapstructComponent mc = context.getComponent("mapstruct", MapstructComponent.class);
mc.setMapperPackageName("com.foo.mapper,com.bar.mapper");
```

This can also be configured in **application.properties**:

```
camel.component.mapstruct.mapper-package-name = com.foo.mapper,com.bar.mapper
```

Camel will on startup scan these packages for classes which names ends with **Mapper**. These classes are then introspected to discover the mapping methods. These mapping methods are then registered into the Camel registry. This means that you can also use type converter to convert the POJOs with MapStruct, such as:

```
from("direct:foo")
  .convertBodyTo(MyFooDto.class);
```

Where **MyFooDto** is a POJO that MapStruct is able to convert to/from.

40.6. SPRING BOOT AUTO-CONFIGURATION

When using mapstruct with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-mapstruct-starter</artifactId>
```

```
</dependency>
```

The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.component.mapstruct.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.mapstruct.enabled	Whether to enable auto configuration of the mapstruct component. This is enabled by default.		Boolean
camel.component.mapstruct.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.mapstruct.map-struct-converter	To use a custom MapStructConverter such as adapting to a special runtime. The option is a org.apache.camel.component.mapstruct.MapStructMapperFinder type.		MapStructMapperFinder
camel.component.mapstruct.mapper-package-name	Package name(s) where Camel should discover Mapstruct mapping classes. Multiple package names can be separated by comma.		String

CHAPTER 41. MASTER

Only consumer is supported

The Camel-Master endpoint provides a way to ensure only a single consumer in a cluster consumes from a given endpoint; with automatic failover if that JVM dies.

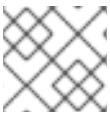
This can be very useful if you need to consume from some legacy back end which either doesn't support concurrent consumption or due to commercial or stability reasons you can only have a single connection at any point in time.

41.1. USING THE MASTER ENDPOINT

Just prefix any camel endpoint with **master:someName:** where *someName* is a logical name and is used to acquire the master lock. e.g.

```
from("master:cheese:jms:foo").to("activemq:wine");
```

In this example, there master component ensures that the route is only active in one node, at any given time, in the cluster. So if there are 8 nodes in the cluster, then the master component will elect one route to be the leader, and only this route will be active, and hence only this route will consume messages from **jms:foo**. In case this route is stopped or unexpected terminated, then the master component will detect this, and re-elect another node to be active, which will then become active and start consuming messages from **jms:foo**.



NOTE

Apache ActiveMQ 5.x has such feature out of the box called Exclusive Consumers.

41.2. URI FORMAT

```
master:namespace:endpoint[?options]
```

Where endpoint is any Camel endpoint you want to run in master/slave mode.

41.3. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

41.3.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

41.3.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

41.4. COMPONENT OPTIONS

The Master component supports 4 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
service (advanced)	Inject the service to use.		CamelClusterService
serviceSelector (advanced)	Inject the service selector used to lookup the CamelClusterService to use.		Selector

41.5. ENDPOINT OPTIONS

The Master endpoint is configured using URI syntax:

```
master:namespace:delegateUri
```

with the following path and query parameters:

41.5.1. Path Parameters (2 parameters)

Name	Description	Default	Type
namespace (consumer)	**Required** The name of the cluster namespace to use.		String
delegateUri (consumer)	**Required** The endpoint uri to use in master/slave mode.		String

41.5.2. Query Parameters (3 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

41.6. EXAMPLE

You can protect a clustered Camel application to only consume files from one active node.

```
// the file endpoint we want to consume from
String url = "file:target/inbox?delete=true";

// use the camel master component in the clustered group named myGroup
// to run a master/slave mode in the following Camel url
from("master:myGroup:" + url)
    .log(name + " - Received file: ${file:name}")
    .delay(delay)
    .log(name + " - Done file:     ${file:name}")
    .to("file:target/outbox");
```

The master component leverages CamelClusterService you can configure using

- **Java**

```
ZooKeeperClusterService service = new ZooKeeperClusterService();
service.setId("camel-node-1");
service.setNodes("myzk:2181");
service.setBasePath("/camel/cluster");

context.addService(service)
```

- **Xml (Spring/Blueprint)**

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


  <bean id="cluster"
class="org.apache.camel.component.zookeeper.cluster.ZooKeeperClusterService">
    <property name="id" value="camel-node-1"/>
    <property name="basePath" value="/camel/cluster"/>
    <property name="nodes" value="myzk:2181"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/spring" autoStartup="false">
    ...
  </camelContext>

</beans>
```

- **Spring boot**

```
camel.component.zookeeper.cluster.service.enabled   = true
camel.component.zookeeper.cluster.service.id        = camel-node-1
camel.component.zookeeper.cluster.service.base-path = /camel/cluster
camel.component.zookeeper.cluster.service.nodes     = myzk:2181
```

41.7. IMPLEMENTATIONS

Camel provides the following ClusterService implementations:

- camel-consul
- camel-file
- camel-infinispan
- camel-jgroups-raft
- camel-jgroups
- camel-kubernetes
- camel-zookeeper

41.8. SPRING BOOT AUTO-CONFIGURATION

When using master with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-master-starter</artifactId>
</dependency>
```

The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.component.master.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.master.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.master.enabled	Whether to enable auto configuration of the master component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.master.service	Inject the service to use. The option is a org.apache.camel.cluster.CamelClusterService type.		CamelClusterService
camel.component.master.service-selector	Inject the service selector used to lookup the CamelClusterService to use. The option is a org.apache.camel.cluster.CamelClusterService.Selector type.		CamelClusterService$Selector

CHAPTER 42. MINIO

Since Camel 3.5

Both producer and consumer are supported

The Minio component supports storing and retrieving objects from/to Minio service.

42.1. PREREQUISITES

You must have valid credentials for authorized access to the buckets/folders. More information is available at Minio.

42.2. URI FORMAT

```
minio://bucketName[?options]
```

The bucket will be created if it doesn't already exist. You can append query options to the URI in the following format,

```
?options=value&option2=value&…
```

For example in order to read file **hello.txt** from the bucket **helloBucket**, use the following snippet:

```
from("minio://helloBucket?accessKey=yourAccessKey&secretKey=yourSecretKey&prefix=hello.txt")
  .to("file:/var/downloaded");
```

42.3. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

42.3.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

42.3.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

42.4. COMPONENT OPTIONS

The Minio component supports 47 options, which are listed below.

Name	Description	Default	Type
autoCreateBucket (common)	Setting the autocreation of the bucket if bucket name not exist.	true	boolean
configuration (common)	The component configuration.		MinioConfiguration
customHttpClient (common)	Set custom HTTP client for authenticated access.		OkHttpClient
endpoint (common)	Endpoint can be an URL, domain name, IPv4 address or IPv6 address.		String
minioClient (common)	**Autowired** Reference to a Minio Client object in the registry.		MinioClient
objectLock (common)	Set when creating new bucket.	false	boolean
policy (common)	The policy for this queue to set in the method.		String
proxyPort (common)	TCP/IP port number. 80 and 443 are used as defaults for HTTP and HTTPS.		Integer
region (common)	The region in which Minio client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1). You'll need to use the name Region.EU_WEST_1.id().		String
secure (common)	Flag to indicate to use secure connection to minio service or not.	false	boolean

Name	Description	Default	Type
serverSideEncryption (common)	Server-side encryption.		ServerSideEncryption
serverSideEncryptionCustomerKey (common)	Server-side encryption for source object while copy/move objects.		ServerSideEncryptionCustomerKey
autoCloseBody (consumer)	If this option is true and includeBody is true, then the MinioObject.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
bypassGovernanceMode (consumer)	Set this flag if you want to bypassGovernanceMode when deleting a particular object.	false	boolean
deleteAfterRead (consumer)	Delete objects from Minio after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the MinioConstants#BUCKET_NAME and MinioConstants#OBJECT_NAME headers, or only the MinioConstants#OBJECT_NAME header.	true	boolean
delimiter (consumer)	The delimiter which is used in the ListObjectsRequest to only consume objects we are interested in.		String
destinationBucketName (consumer)	Source bucket name.		String

Name	Description	Default	Type
destinationObjectName (consumer)	Source object name.		String
includeBody (consumer)	If it is true, the exchange body will be set to a stream to the contents of the file. If false, the headers will be set with the Minio object metadata, but the body will be null. This option is strongly related to autocloseBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	boolean
includeFolders (consumer)	The flag which is used in the ListObjectsRequest to set include folders.	false	boolean
includeUserMetadata (consumer)	The flag which is used in the ListObjectsRequest to get objects with user meta data.	false	boolean
includeVersions (consumer)	The flag which is used in the ListObjectsRequest to get objects with versioning.	false	boolean
length (consumer)	Number of bytes of object data from offset.		long
matchETag (consumer)	Set match ETag parameter for get object(s).		String
maxConnections (consumer)	Set the maxConnections parameter in the minio client configuration.	60	int
maxMessagesPerPoll (consumer)	Gets the maximum number of messages as a limit to poll at each polling. Gets the maximum number of messages as a limit to poll at each polling. The default value is 10. Use 0 or a negative number to set it as unlimited.	10	int
modifiedSince (consumer)	Set modified since parameter for get object(s).		ZonedDateTime
moveAfterRead (consumer)	Move objects from bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	boolean

Name	Description	Default	Type
notMatchETag (consumer)	Set not match ETag parameter for get object(s).		String
objectName (consumer)	To get the object from the bucket with the given object name.		String
offset (consumer)	Start byte position of object data.		long
prefix (consumer)	Object name starts with prefix.		String
recursive (consumer)	List recursively than directory structure emulation.	false	boolean
startAfter (consumer)	list objects in bucket after this object name.		String
unModifiedSince (consumer)	Set un modified since parameter for get object(s).		ZonedDateTime
useVersion1 (consumer)	when true, version 1 of REST API is used.	false	boolean
versionId (consumer)	Set specific version_ID of a object when deleting the object.		String
deleteAfterWrite (producer)	Delete file object after the Minio file has been uploaded.	false	boolean
keyName (producer)	Setting the key name for an element in the bucket through endpoint parameter.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
operation (producer)	The operation to do in case the user don't want to do only an upload. Enum values: • copyObject • listObjects • deleteObject • deleteObjects • deleteBucket • listBuckets • getObject • getObjectRange		MinioOperations
pojoRequest (producer)	If we want to use a POJO request as body or not.	false	boolean
storageClass (producer)	The storage class to set in the request.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
accessKey (security)	Amazon AWS Secret Access Key or Minio Access Key. If not set camel will connect to service for anonymous access.		String
secretKey (security)	Amazon AWS Access Key Id or Minio Secret Key. If not set camel will connect to service for anonymous access.		String

42.5. ENDPOINT OPTIONS

The Minio endpoint is configured using URI syntax:

```
minio:bucketName
```

with the following path and query parameters:

42.5.1. Path Parameters (1 parameters)

Name	Description	Default	Type
bucketName (common)	**Required** Bucket name.		String

42.5.2. Query Parameters (63 parameters)

Name	Description	Default	Type
autoCreateBucket (common)	Setting the autocreation of the bucket if bucket name not exist.	true	boolean
customHttpClient (common)	Set custom HTTP client for authenticated access.		OkHttpClient
endpoint (common)	Endpoint can be an URL, domain name, IPv4 address or IPv6 address.		String
minioClient (common)	**Autowired** Reference to a Minio Client object in the registry.		MinioClient
objectLock (common)	Set when creating new bucket.	false	boolean
policy (common)	The policy for this queue to set in the method.		String
proxyPort (common)	TCP/IP port number. 80 and 443 are used as defaults for HTTP and HTTPS.		Integer
region (common)	The region in which Minio client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1). You'll need to use the name Region.EU_WEST_1.id().		String
secure (common)	Flag to indicate to use secure connection to minio service or not.	false	boolean
serverSideEncryption (common)	Server-side encryption.		ServerSideEncryption

Name	Description	Default	Type
serverSideEncryptionCustomerKey (common)	Server-side encryption for source object while copy/move objects.		ServerSideEncryptionCustomerKey
autoCloseBody (consumer)	If this option is true and includeBody is true, then the MinioObject.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	boolean
bypassGovernanceMode (consumer)	Set this flag if you want to bypassGovernanceMode when deleting a particular object.	false	boolean
deleteAfterRead (consumer)	Delete objects from Minio after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the MinioConstants#BUCKET_NAME and MinioConstants#OBJECT_NAME headers, or only the MinioConstants#OBJECT_NAME header.	true	boolean
delimiter (consumer)	The delimiter which is used in the ListObjectsRequest to only consume objects we are interested in.		String
destinationBucketName (consumer)	Source bucket name.		String
destinationObjectName (consumer)	Source object name.		String

Name	Description	Default	Type
includeBody (consumer)	If it is true, the exchange body will be set to a stream to the contents of the file. If false, the headers will be set with the Minio object metadata, but the body will be null. This option is strongly related to autocloseBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	boolean
includeFolders (consumer)	The flag which is used in the ListObjectsRequest to set include folders.	false	boolean
includeUserMetadata (consumer)	The flag which is used in the ListObjectsRequest to get objects with user meta data.	false	boolean
includeVersions (consumer)	The flag which is used in the ListObjectsRequest to get objects with versioning.	false	boolean
length (consumer)	Number of bytes of object data from offset.		long
matchETag (consumer)	Set match ETag parameter for get object(s).		String
maxConnections (consumer)	Set the maxConnections parameter in the minio client configuration.	60	int
maxMessagesPerPoll (consumer)	Gets the maximum number of messages as a limit to poll at each polling. Gets the maximum number of messages as a limit to poll at each polling. The default value is 10. Use 0 or a negative number to set it as unlimited.	10	int
modifiedSince (consumer)	Set modified since parameter for get object(s).		ZonedDateTime
moveAfterRead (consumer)	Move objects from bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	boolean
notMatchETag (consumer)	Set not match ETag parameter for get object(s).		String

Name	Description	Default	Type
objectName (consumer)	To get the object from the bucket with the given object name.		String
offset (consumer)	Start byte position of object data.		long
prefix (consumer)	Object name starts with prefix.		String
recursive (consumer)	List recursively than directory structure emulation.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
startAfter (consumer)	list objects in bucket after this object name.		String
unModifiedSince (consumer)	Set un modified since parameter for get object(s).		ZonedDateTime
useVersion1 (consumer)	when true, version 1 of REST API is used.	false	boolean
versionId (consumer)	Set specific version_ID of a object when deleting the object.		String
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
deleteAfterWrite (producer)	Delete file object after the Minio file has been uploaded.	false	boolean
keyName (producer)	Setting the key name for an element in the bucket through endpoint parameter.		String
operation (producer)	The operation to do in case the user don't want to do only an upload. Enum values: • copyObject • listObjects • deleteObject • deleteObjects • deleteBucket • listBuckets • getObject • getObjectRange		MinioOperations
pojoRequest (producer)	If we want to use a POJO request as body or not.	false	boolean
storageClass (producer)	The storage class to set in the request.		String

Name	Description	Default	Type
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
accessKey (security)	Amazon AWS Secret Access Key or Minio Access Key. If not set camel will connect to service for anonymous access.		String
secretKey (security)	Amazon AWS Access Key Id or Minio Secret Key. If not set camel will connect to service for anonymous access.		String

You have to provide the minioClient in the Registry or your accessKey and secretKey to access the Minio.

42.6. BATCH CONSUMER

This component implements the Batch Consumer.

This allows you for instance to know how many messages exists in this batch and for instance let the Aggregator aggregate this number of messages.

42.7. MESSAGE HEADERS

The Minio component supports 21 message header(s), which is/are listed below:

Name	Description	Default	Type
CamelMinioBucketName (common) Constant: BUCKET_NAME	Producer: The bucket Name which this object will be stored or which will be used for the current operation. Consumer: The name of the bucket in which this object is contained.		String
CamelMinioDestinationBucketName (producer) Constant: DESTINATION_BUCKET_NAME	The bucket Destination Name which will be used for the current operation.		String

Name	Description	Default	Type
CamelMinioContentControl (common) Constant: CACHE_CONTROL	Producer: The content control of this object. Consumer: The optional Cache-Control HTTP header which allows the user to specify caching behavior along the HTTP request/reply chain.		String
CamelMinioContentDisposition (common) Constant: CONTENT_DISPOSITION	Producer: The content disposition of this object. Consumer: The optional Content-Disposition HTTP header, which specifies presentational information such as the recommended filename for the object to be saved as.		String
CamelMinioContentEncoding (common) Constant: CONTENT_ENCODING	Producer: The content encoding of this object. Consumer: The optional Content-Encoding HTTP header specifying what content encodings have been applied to the object and what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type field.		String
CamelMinioContentLength (common) Constant: CONTENT_LENGTH	Producer: The content length of this object. Consumer: The Content-Length HTTP header indicating the size of the associated object in bytes.		Long
CamelMinioContentMD5 (common) Constant: CONTENT_MD5	Producer: The md5 checksum of this object. Consumer: The base64 encoded 128-bit MD5 digest of the associated object (content - not including headers) according to RFC 1864. This data is used as a message integrity check to verify that the data received by Minio is the same data that the caller sent.		String
CamelMinioContentType (common) Constant: CONTENT_TYPE	Producer: The content type of this object. Consumer: The Content-Type HTTP header, which indicates the type of content stored in the associated object. The value of this header is a standard MIME type.		String

Name	Description	Default	Type
CamelMinioETag (common) Constant: E_TAG	Producer: The ETag value for the newly uploaded object. Consumer: The hex encoded 128-bit MD5 digest of the associated object according to RFC 1864. This data is used as an integrity check to verify that the data received by the caller is the same data that was sent by Minio.		String
CamelMinioObjectName (common) Constant: OBJECT_NAME	Producer: The key under which this object will be stored or which will be used for the current operation. Consumer: The key under which this object is stored.		String
CamelMinioDestinationObjectName (producer) Constant: DESTINATION_OBJECT_NAME	The Destination key which will be used for the current operation.		String
CamelMinioLastModified (common) Constant: LAST_MODIFIED	Producer: The last modified timestamp of this object. Consumer: The value of the Last-Modified header, indicating the date and time at which Minio last recorded a modification to the associated object.		Date
CamelMinioStorageClass (producer) Constant: STORAGE_CLASS	The storage class of this object.		String
CamelMinioVersionId (common) Constant: VERSION_ID	Producer: The version Id of the object to be stored or returned from the current operation. Consumer: The version ID of the associated Minio object if available. Version IDs are only assigned to objects when an object is uploaded to an Minio bucket that has object versioning enabled.		String
CamelMinioCannedAcl (producer) Constant: CANNED_ACL	The canned acl that will be applied to the object. see com.amazonaws.services.s3.model.CannedAccessControlList for allowed values.		String

Name	Description	Default	Type
CamelMinioOperation (producer) Constant: MINIO_OPERATION	The operation to perform. Enum values: • copyObject • listObjects • deleteObject • deleteObjects • deleteBucket • listBuckets • getObject • getPartialObject		MinioOperations
CamelMinioServerSideEncryption (common) Constant: SERVER_SIDE_ENCRYPTION	Producer: Sets the server-side encryption algorithm when encrypting the object using Minio-managed keys. For example use AES256. Consumer: The server-side encryption algorithm when encrypting the object using Minio-managed keys.		String
CamelMinioExpirationTime (common) Constant: EXPIRATION_TIME	The expiration time.		String
CamelMinioReplicationStatus (common) Constant: REPLICATION_STATUS	The replication status.		String
CamelMinioOffset (producer) Constant: OFFSET	The offset.		String

Name	Description	Default	Type
CamelMinioLength (producer) Constant: LENGTH	The length.		String

42.7.1. Minio Producer operations

Camel-Minio component provides the following operation on the producer side:

- copyObject
- deleteObject
- deleteObjects
- listBuckets
- deleteBucket
- listObjects
- getObject (this will return a MinioObject instance)
- getObjectRange (this will return a MinioObject instance)

42.7.2. Advanced Minio configuration

If your Camel Application is running behind a firewall or if you need to have more control over the **MinioClient** instance configuration, you can create your own instance and refer to it in your Camel minio component configuration:

```
from("minio://MyBucket?minioClient=#client&delay=5000&maxMessagesPerPoll=5")
.to("mock:result");
```

42.7.3. Minio Producer Operation examples

- CopyObject: this operation copy an object from one bucket to a different one

```
 from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(MinioConstants.DESTINATION_BUCKET_NAME,
"camelDestinationBucket");
        exchange.getIn().setHeader(MinioConstants.OBJECT_NAME, "camelKey");
        exchange.getIn().setHeader(MinioConstants.DESTINATION_OBJECT_NAME,
"camelDestinationKey");
    }
```

```
})
.to("minio://mycamelbucket?minioClient=#minioClient&operation=copyObject")
.to("mock:result");
```

This operation will copy the object with the name expressed in the header camelDestinationKey to the camelDestinationBucket bucket, from the bucket mycamelbucket.

- DeleteObject: this operation deletes an object from a bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(MinioConstants.OBJECT_NAME, "camelKey");
    }
})
.to("minio://mycamelbucket?minioClient=#minioClient&operation=deleteObject")
.to("mock:result");
```

This operation will delete the object camelKey from the bucket mycamelbucket.

- ListBuckets: this operation list the buckets for this account in this region

```
from("direct:start")
.to("minio://mycamelbucket?minioClient=#minioClient&operation=listBuckets")
.to("mock:result");
```

This operation will list the buckets for this account

- DeleteBucket: this operation delete the bucket specified as URI parameter or header

```
from("direct:start")
.to("minio://mycamelbucket?minioClient=#minioClient&operation=deleteBucket")
.to("mock:result");
```

This operation will delete the bucket mycamelbucket

- ListObjects: this operation list object in a specific bucket

```
from("direct:start")
.to("minio://mycamelbucket?minioClient=#minioClient&operation=listObjects")
.to("mock:result");
```

This operation will list the objects in the mycamelbucket bucket

- GetObject: this operation get a single object in a specific bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(MinioConstants.OBJECT_NAME, "camelKey");
    }
```

```
    })
    .to("minio://mycamelbucket?minioClient=#minioClient&operation=getObject")
    .to("mock:result");
```

This operation will return an MinioObject instance related to the camelKey object in mycamelbucket bucket.

- GetObjectRange: this operation get a single object range in a specific bucket

```
from("direct:start").process(new Processor() {

    @Override
    public void process(Exchange exchange) throws Exception {
        exchange.getIn().setHeader(MinioConstants.OBJECT_NAME, "camelKey");
        exchange.getIn().setHeader(MinioConstants.OFFSET, "0");
        exchange.getIn().setHeader(MinioConstants.LENGTH, "9");
    }
})
.to("minio://mycamelbucket?minioClient=#minioClient&operation=getObjectRange")
.to("mock:result");
```

This operation will return an MinioObject instance related to the camelKey object in mycamelbucket bucket, containing bytes from 0 to 9.

42.8. BUCKET AUTOCREATION

With the option **autoCreateBucket** users are able to avoid the autocreation of a Minio Bucket in case it doesn't exist. The default for this option is **true**. If set to false any operation on a not-existent bucket in Minio won't be successful, and an error will be returned.

42.9. AUTOMATIC DETECTION OF MINIO CLIENT IN REGISTRY

The component is capable of detecting the presence of a Minio bean into the registry. If it's the only instance of that type it will be used as client, and you won't have to define it as uri parameter, like the example above. This may be really useful for smarter configuration of the endpoint.

42.10. MOVING STUFF BETWEEN A BUCKET AND ANOTHER BUCKET

Some users like to consume stuff from a bucket and move the content in a different one without using the copyObject feature of this component. If this is case for you, don't forget to remove the bucketName header from the incoming exchange of the consumer, otherwise the file will always be overwritten on the same original bucket.

42.11. MOVEAFTERREAD CONSUMER OPTION

In addition to deleteAfterRead it has been added another option, moveAfterRead. With this option enabled the consumed object will be moved to a target destinationBucket instead of being only deleted. This will require specifying the destinationBucket option. As example:

```
  from("minio://mycamelbucket?
minioClient=#minioClient&moveAfterRead=true&destinationBucketName=myothercamelbucket")
  .to("mock:result");
```

In this case the objects consumed will be moved to myothercamelbucket bucket and deleted from the original one (because of deleteAfterRead set to true as default).

42.12. USING A POJO AS BODY

Sometimes build a Minio Request can be complex, because of multiple options. We introduce the possibility to use a POJO as body. In Minio there are multiple operations you can submit, as an example for List brokers request, you can do something like:

```
from("direct:minio")
    .setBody(ListObjectsArgs.builder()
                .bucket(bucketName)
                .recursive(getConfiguration().isRecursive())))
    .to("minio://test?minioClient=#minioClient&operation=listObjects&pojoRequest=true")
```

In this way you'll pass the request directly without the need of passing headers and options specifically related to this operation.

42.13. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-minio</artifactId>
    <version>${camel-version}</version>
</dependency>
```

where **$3.18.3** must be replaced by the actual version of Camel.

42.14. SPRING BOOT AUTO-CONFIGURATION

When using minio with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-minio-starter</artifactId>
</dependency>
```

The component supports 48 options, which are listed below.

Name	Description	Default	Type
camel.component.minio.access-key	Amazon AWS Secret Access Key or Minio Access Key. If not set camel will connect to service for anonymous access.		String

Name	Description	Default	Type
camel.component.minio.auto-close-body	If this option is true and includeBody is true, then the MinioObject.close() method will be called on exchange completion. This option is strongly related to includeBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	Boolean
camel.component.minio.auto-create-bucket	Setting the autocreation of the bucket if bucket name not exist.	true	Boolean
camel.component.minio.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.minio.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.minio.bypass-governance-mode	Set this flag if you want to bypassGovernanceMode when deleting a particular object.	false	Boolean
camel.component.minio.configuration	The component configuration. The option is a org.apache.camel.component.minio.MinioConfiguration type.		MinioConfiguration
camel.component.minio.custom-http-client	Set custom HTTP client for authenticated access. The option is a okhttp3.OkHttpClient type.		OkHttpClient

Name	Description	Default	Type
camel.component.minio.delete-after-read	Delete objects from Minio after they have been retrieved. The delete is only performed if the Exchange is committed. If a rollback occurs, the object is not deleted. If this option is false, then the same objects will be retrieve over and over again on the polls. Therefore you need to use the Idempotent Consumer EIP in the route to filter out duplicates. You can filter using the MinioConstants#BUCKET_NAME and MinioConstants#OBJECT_NAME headers, or only the MinioConstants#OBJECT_NAME header.	true	Boolean
camel.component.minio.delete-after-write	Delete file object after the Minio file has been uploaded.	false	Boolean
camel.component.minio.delimiter	The delimiter which is used in the ListObjectsRequest to only consume objects we are interested in.		String
camel.component.minio.destination-bucket-name	Source bucket name.		String
camel.component.minio.destination-object-name	Source object name.		String
camel.component.minio.enabled	Whether to enable auto configuration of the minio component. This is enabled by default.		Boolean
camel.component.minio.endpoint	Endpoint can be an URL, domain name, IPv4 address or IPv6 address.		String
camel.component.minio.include-body	If it is true, the exchange body will be set to a stream to the contents of the file. If false, the headers will be set with the Minio object metadata, but the body will be null. This option is strongly related to autocloseBody option. In case of setting includeBody to true and autocloseBody to false, it will be up to the caller to close the MinioObject stream. Setting autocloseBody to true, will close the MinioObject stream automatically.	true	Boolean
camel.component.minio.include-folders	The flag which is used in the ListObjectsRequest to set include folders.	false	Boolean

Name	Description	Default	Type
camel.component.minio.include-user-metadata	The flag which is used in the ListObjectsRequest to get objects with user meta data.	false	Boolean
camel.component.minio.include-versions	The flag which is used in the ListObjectsRequest to get objects with versioning.	false	Boolean
camel.component.minio.key-name	Setting the key name for an element in the bucket through endpoint parameter.		String
camel.component.minio.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.minio.length	Number of bytes of object data from offset.		Long
camel.component.minio.match-e-tag	Set match ETag parameter for get object(s).		String
camel.component.minio.max-connections	Set the maxConnections parameter in the minio client configuration.	60	Integer
camel.component.minio.max-messages-per-poll	Gets the maximum number of messages as a limit to poll at each polling. Gets the maximum number of messages as a limit to poll at each polling. The default value is 10. Use 0 or a negative number to set it as unlimited.	10	Integer
camel.component.minio.minio-client	Reference to a Minio Client object in the registry. The option is a io.minio.MinioClient type.		MinioClient
camel.component.minio.modified-since	Set modified since parameter for get object(s). The option is a java.time.ZonedDateTime type.		ZonedDateTime

Name	Description	Default	Type
camel.component.minio.move-after-read	Move objects from bucket to a different bucket after they have been retrieved. To accomplish the operation the destinationBucket option must be set. The copy bucket operation is only performed if the Exchange is committed. If a rollback occurs, the object is not moved.	false	Boolean
camel.component.minio.not-match-e-tag	Set not match ETag parameter for get object(s).		String
camel.component.minio.object-lock	Set when creating new bucket.	false	Boolean
camel.component.minio.object-name	To get the object from the bucket with the given object name.		String
camel.component.minio.offset	Start byte position of object data.		Long
camel.component.minio.operation	The operation to do in case the user don't want to do only an upload.		MinioOperations
camel.component.minio.pojo-request	If we want to use a POJO request as body or not.	false	Boolean
camel.component.minio.policy	The policy for this queue to set in the method.		String
camel.component.minio.prefix	Object name starts with prefix.		String
camel.component.minio.proxy-port	TCP/IP port number. 80 and 443 are used as defaults for HTTP and HTTPS.		Integer
camel.component.minio.recursive	List recursively than directory structure emulation.	false	Boolean
camel.component.minio.region	The region in which Minio client needs to work. When using this parameter, the configuration will expect the lowercase name of the region (for example ap-east-1). You'll need to use the name Region.EU_WEST_1.id().		String

Name	Description	Default	Type
camel.component.minio.secret-key	Amazon AWS Access Key Id or Minio Secret Key. If not set camel will connect to service for anonymous access.		String
camel.component.minio.secure	Flag to indicate to use secure connection to minio service or not.	false	Boolean
camel.component.minio.server-side-encryption	Server-side encryption. The option is a io.minio.ServerSideEncryption type.		ServerSideEncryption
camel.component.minio.server-side-encryption-customer-key	Server-side encryption for source object while copy/move objects. The option is a io.minio.ServerSideEncryptionCustomerKey type.		ServerSideEncryptionCustomerKey
camel.component.minio.start-after	list objects in bucket after this object name.		String
camel.component.minio.storage-class	The storage class to set in the request.		String
camel.component.minio.un-modified-since	Set un modified since parameter for get object(s). The option is a java.time.ZonedDateTime type.		ZonedDateTime
camel.component.minio.use-version1	when true, version 1 of REST API is used.	false	Boolean
camel.component.minio.version-id	Set specific version_ID of a object when deleting the object.		String

CHAPTER 43. MLLP

Both producer and consumer are supported

The MLLP component is specifically designed to handle the nuances of the MLLP protocol and provide the functionality required by Healthcare providers to communicate with other systems using the MLLP protocol.

The MLLP component provides a simple configuration URI, automated HL7 acknowledgment generation and automatic acknowledgment interrogation.

The MLLP protocol does not typically use a large number of concurrent TCP connections - a single active TCP connection is the normal case. Therefore, the MLLP component uses a simple thread-per-connection model based on standard Java Sockets. This keeps the implementation simple and eliminates the dependencies on only Camel itself.

The component supports the following:

- A Camel consumer using a TCP Server
- A Camel producer using a TCP Client

The MLLP component use **byte[]** payloads, and relies on Camel type conversion to convert **byte[]** to other types.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mllp</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

43.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

43.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

43.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

43.2. COMPONENT OPTIONS

The MLLP component supports 30 options, which are listed below.

Name	Description	Default	Type
autoAck (common)	Enable/Disable the automatic generation of a MLLP Acknowledgement MLLP Consumers only.	true	boolean
charsetName (common)	Sets the default charset to use.		String
configuration (common)	Sets the default configuration to use when creating MLLP endpoints.		MllpConfiguration
hl7Headers (common)	Enable/Disable the automatic generation of message headers from the HL7 Message MLLP Consumers only.	true	boolean
requireEndOfData (common)	Enable/Disable strict compliance to the MLLP standard. The MLLP standard specifies START_OF_BLOCKhl7 payloadEND_OF_BLOCKEND_OF_DATA, however, some systems do not send the final END_OF_DATA byte. This setting controls whether or not the final END_OF_DATA byte is required or optional.	true	boolean
stringPayload (common)	Enable/Disable converting the payload to a String. If enabled, HL7 Payloads received from external systems will be validated converted to a String. If the charsetName property is set, that character set will be used for the conversion. If the charsetName property is not set, the value of MSH-18 will be used to determine th appropriate character set. If MSH-18 is not set, then the default ISO-8859-1 character set will be use.	true	boolean

Name	Description	Default	Type
validatePayload (common)	Enable/Disable the validation of HL7 Payloads If enabled, HL7 Payloads received from external systems will be validated (see Hl7Util.generateInvalidPayloadExceptionMessage for details on the validation). If and invalid payload is detected, a MllpInvalidMessageException (for consumers) or a MllpInvalidAcknowledgementException will be thrown.	false	boolean
acceptTimeout (consumer)	Timeout (in milliseconds) while waiting for a TCP connection TCP Server Only.	60000	int
backlog (consumer)	The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.	5	Integer
bindRetryInterval (consumer)	TCP Server Only - The number of milliseconds to wait between bind attempts.	5000	int
bindTimeout (consumer)	TCP Server Only - The number of milliseconds to retry binding to a server port.	30000	int
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to receive incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. If disabled, the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions by logging them at WARN or ERROR level and ignored.	true	boolean
lenientBind (consumer)	TCP Server Only - Allow the endpoint to start before the TCP ServerSocket is bound. In some environments, it may be desirable to allow the endpoint to start before the TCP ServerSocket is bound.	false	boolean
maxConcurrentConsumers (consumer)	The maximum number of concurrent MLLP Consumer connections that will be allowed. If a new connection is received and the maximum is number are already established, the new connection will be reset immediately.	5	int
reuseAddress (consumer)	Enable/disable the SO_REUSEADDR socket option.	false	Boolean

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut	InOut	ExchangePattern
connectTimeout (producer)	Timeout (in milliseconds) for establishing for a TCP connection TCP Client only.	30000	int
idleTimeoutStrategy (producer)	decide what action to take when idle timeout occurs. Possible values are : RESET: set SO_LINGER to 0 and reset the socket CLOSE: close the socket gracefully default is RESET. Enum values: • RESET • CLOSE	RESET	MllpIdleTimeoutStrategy
keepAlive (producer)	Enable/disable the SO_KEEPALIVE socket option.	true	Boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
tcpNoDelay (producer)	Enable/disable the TCP_NODELAY socket option.	true	Boolean

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
defaultCharset (advanced)	Set the default character set to use for byte to/from String conversions.	ISO-8859-1	String
logPhi (advanced)	Whether to log PHI.	true	Boolean
logPhiMaxBytes (advanced)	Set the maximum number of bytes of PHI that will be logged in a log entry.	5120	Integer
readTimeout (advanced)	The SO_TIMEOUT value (in milliseconds) used after the start of an MLLP frame has been received.	5000	int
receiveBufferSize (advanced)	Sets the SO_RCVBUF option to the specified value (in bytes).	8192	Integer
receiveTimeout (advanced)	The SO_TIMEOUT value (in milliseconds) used when waiting for the start of an MLLP frame.	15000	int
sendBufferSize (advanced)	Sets the SO_SNDBUF option to the specified value (in bytes).	8192	Integer
idleTimeout (tcp)	The approximate idle time allowed before the Client TCP Connection will be reset. A null value or a value less than or equal to zero will disable the idle timeout.		Integer

43.3. ENDPOINT OPTIONS

The MLLP endpoint is configured using URI syntax:

```
mllp:hostname:port
```

with the following path and query parameters:

43.3.1. Path Parameters (2 parameters)

Name	Description	Default	Type
hostname (common)	**Required** Hostname or IP for connection for the TCP connection. The default value is null, which means any local IP address.		String
port (common)	**Required** Port number for the TCP connection.		int

43.3.2. Query Parameters (26 parameters)

Name	Description	Default	Type
autoAck (common)	Enable/Disable the automatic generation of a MLLP Acknowledgement MLLP Consumers only.	true	boolean
charsetName (common)	Sets the default charset to use.		String
hl7Headers (common)	Enable/Disable the automatic generation of message headers from the HL7 Message MLLP Consumers only.	true	boolean
requireEndOfData (common)	Enable/Disable strict compliance to the MLLP standard. The MLLP standard specifies START_OF_BLOCKhl7 payloadEND_OF_BLOCKEND_OF_DATA, however, some systems do not send the final END_OF_DATA byte. This setting controls whether or not the final END_OF_DATA byte is required or optional.	true	boolean
stringPayload (common)	Enable/Disable converting the payload to a String. If enabled, HL7 Payloads received from external systems will be validated converted to a String. If the charsetName property is set, that character set will be used for the conversion. If the charsetName property is not set, the value of MSH-18 will be used to determine th appropriate character set. If MSH-18 is not set, then the default ISO-8859-1 character set will be use.	true	boolean

Name	Description	Default	Type
validatePayload (common)	Enable/Disable the validation of HL7 Payloads If enabled, HL7 Payloads received from external systems will be validated (see Hl7Util.generateInvalidPayloadExceptionMessage for details on the validation). If and invalid payload is detected, a MllpInvalidMessageException (for consumers) or a MllpInvalidAcknowledgementException will be thrown.	false	boolean
acceptTimeout (consumer)	Timeout (in milliseconds) while waiting for a TCP connection TCP Server Only.	60000	int
backlog (consumer)	The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.	5	Integer
bindRetryInterval (consumer)	TCP Server Only - The number of milliseconds to wait between bind attempts.	5000	int
bindTimeout (consumer)	TCP Server Only - The number of milliseconds to retry binding to a server port.	30000	int
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to receive incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. If disabled, the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions by logging them at WARN or ERROR level and ignored.	true	boolean
lenientBind (consumer)	TCP Server Only - Allow the endpoint to start before the TCP ServerSocket is bound. In some environments, it may be desirable to allow the endpoint to start before the TCP ServerSocket is bound.	false	boolean
maxConcurrentConsumers (consumer)	The maximum number of concurrent MLLP Consumer connections that will be allowed. If a new connection is received and the maximum is number are already established, the new connection will be reset immediately.	5	int

Name	Description	Default	Type
reuseAddress (consumer)	Enable/disable the SO_REUSEADDR socket option.	false	Boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut	InOut	ExchangePattern
connectTimeout (producer)	Timeout (in milliseconds) for establishing for a TCP connection TCP Client only.	30000	int
idleTimeoutStrategy (producer)	decide what action to take when idle timeout occurs. Possible values are : RESET: set SO_LINGER to 0 and reset the socket CLOSE: close the socket gracefully default is RESET. Enum values: • RESET • CLOSE	RESET	MllpIdleTimeoutStrategy
keepAlive (producer)	Enable/disable the SO_KEEPALIVE socket option.	true	Boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
tcpNoDelay (producer)	Enable/disable the TCP_NODELAY socket option.	true	Boolean
readTimeout (advanced)	The SO_TIMEOUT value (in milliseconds) used after the start of an MLLP frame has been received.	5000	int
receiveBufferSize (advanced)	Sets the SO_RCVBUF option to the specified value (in bytes).	8192	Integer
receiveTimeout (advanced)	The SO_TIMEOUT value (in milliseconds) used when waiting for the start of an MLLP frame.	15000	int
sendBufferSize (advanced)	Sets the SO_SNDBUF option to the specified value (in bytes).	8192	Integer
idleTimeout (tcp)	The approximate idle time allowed before the Client TCP Connection will be reset. A null value or a value less than or equal to zero will disable the idle timeout.		Integer

43.4. MLLP CONSUMER

The MLLP Consumer supports receiving MLLP-framed messages and sending HL7 Acknowledgements. The MLLP Consumer can automatically generate the HL7 Acknowledgement (HL7 Application Acknowledgements only - AA, AE and AR), or the acknowledgement can be specified using the CamelMllpAcknowledgement exchange property. Additionally, the type of acknowledgement that will be generated can be controlled by setting the CamelMllpAcknowledgementType exchange property. The MLLP Consumer can read messages without sending any HL7 Acknowledgement if the automatic acknowledgement is disabled and exchange pattern is InOnly.

43.4.1. Message Headers

The MLLP Consumer adds these headers on the Camel message:

Key	Description
CamelMllpLocalAddress	The local TCP Address of the Socket
CamelMllpRemoteAddress	The local TCP Address of the Socket
CamelMllpSendingApplication	MSH-3 value
CamelMllpSendingFacility	MSH-4 value
CamelMllpReceivingApplication	MSH-5 value

CamelMllpReceivingFacility	MSH-6 value
CamelMllpTimestamp	MSH-7 value
CamelMllpSecurity	MSH-8 value
CamelMllpMessageType	MSH-9 value
CamelMllpEventType	MSH-9-1 value
CamelMllpTriggerEvent	MSH-9-2 value
CamelMllpMessageControlId	MSH-10 value
CamelMllpProcessingId	MSH-11 value
CamelMllpVersionId	MSH-12 value
CamelMllpCharset	MSH-18 value

All headers are String types. If a header value is missing, its value is null.

43.4.2. Exchange Properties

The type of acknowledgment the MLLP Consumer generates and state of the TCP Socket can be controlled by these properties on the Camel exchange:

Key	Type	Description
CamelMllpAcknowledgement	byte[]	If present, this property will we sent to client as the MLLP Acknowledgement
CamelMllpAcknowledgementString	String	If present and CamelMllpAcknowledgement is not present, this property will we sent to client as the MLLP Acknowledgement
CamelMllpAcknowledgementMsaText	String	If neither CamelMllpAcknowledgement or CamelMllpAcknowledgementString are present and autoAck is true, this property can be used to specify the contents of MSA-3 in the generated HL7 acknowledgement
CamelMllpAcknowledgementType	String	If neither CamelMllpAcknowledgement or CamelMllpAcknowledgementString are present and autoAck is true, this property can be used to specify the HL7 acknowledgement type (i.e. AA, AE, AR)
CamelMllpAutoAcknowledge	Boolean	Overrides the autoAck query parameter

Key	Type	Description
CamelMllpCloseConnectionBeforeSend	Boolean	If true, the Socket will be closed before sending data
CamelMllpResetConnectionBeforeSend	Boolean	If true, the Socket will be reset before sending data
CamelMllpCloseConnectionAfterSend	Boolean	If true, the Socket will be closed immediately after sending data
CamelMllpResetConnectionAfterSend	Boolean	If true, the Socket will be reset immediately after sending any data

43.5. MLLP PRODUCER

The MLLP Producer supports sending MLLP-framed messages and receiving HL7 Acknowledgements. The MLLP Producer interrogates the HL7 Acknowledgments and raises exceptions if a negative acknowledgement is received. The received acknowledgement is interrogated and an exception is raised in the event of a negative acknowledgement. The MLLP Producer can ignore acknowledgements when configured with InOnly exchange pattern.

43.5.1. Message Headers

The MLLP Producer adds these headers on the Camel message:

Key	Description
CamelMllpLocalAddress	The local TCP Address of the Socket
CamelMllpRemoteAddress	The remote TCP Address of the Socket
CamelMllpAcknowledgement	The HL7 Acknowledgment byte[] received
CamelMllpAcknowledgementString	The HL7 Acknowledgment received, converted to a String

43.5.2. Exchange Properties

The state of the TCP Socket can be controlled by these properties on the Camel exchange:

Key	Type	Description
CamelMllpCloseConnectionBeforeSend	Boolean	If true, the Socket will be closed before sending data

Key	Type	Description
CamelMllpResetConnectionBeforeSend	Boolean	If true, the Socket will be reset before sending data
CamelMllpCloseConnectionAfterSend	Boolean	If true, the Socket will be closed immediately after sending data
CamelMllpResetConnectionAfterSend	Boolean	If true, the Socket will be reset immediately after sending any data

43.6. SPRING BOOT AUTO-CONFIGURATION

When using mllp with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-mllp-starter</artifactId>
</dependency>
```

The component supports 31 options, which are listed below.

Name	Description	Default	Type
camel.component.mllp.accept-timeout	Timeout (in milliseconds) while waiting for a TCP connection TCP Server Only.	60000	Integer
camel.component.mllp.auto-ack	Enable/Disable the automatic generation of a MLLP Acknowledgement MLLP Consumers only.	true	Boolean
camel.component.mllp.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.mllp.backlog	The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.	5	Integer
camel.component.mllp.bind-retry-interval	TCP Server Only - The number of milliseconds to wait between bind attempts.	5000	Integer

Name	Description	Default	Type
camel.component.mllp.bind-timeout	TCP Server Only - The number of milliseconds to retry binding to a server port.	30000	Integer
camel.component.mllp.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to receive incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. If disabled, the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions by logging them at WARN or ERROR level and ignored.	true	Boolean
camel.component.mllp.charset-name	Sets the default charset to use.		String
camel.component.mllp.configuration	Sets the default configuration to use when creating MLLP endpoints. The option is a org.apache.camel.component.mllp.MllpConfiguration type.		MllpConfiguration
camel.component.mllp.connect-timeout	Timeout (in milliseconds) for establishing for a TCP connection TCP Client only.	30000	Integer
camel.component.mllp.default-charset	Set the default character set to use for byte to/from String conversions.	ISO-8859-1	String
camel.component.mllp.enabled	Whether to enable auto configuration of the mllp component. This is enabled by default.		Boolean
camel.component.mllp.exchange-pattern	Sets the exchange pattern when the consumer creates an exchange.		ExchangePattern
camel.component.mllp.hl7-headers	Enable/Disable the automatic generation of message headers from the HL7 Message MLLP Consumers only.	true	Boolean
camel.component.mllp.idle-timeout	The approximate idle time allowed before the Client TCP Connection will be reset. A null value or a value less than or equal to zero will disable the idle timeout.		Integer

Name	Description	Default	Type
camel.component.mllp.idle-timeout-strategy	decide what action to take when idle timeout occurs. Possible values are : RESET: set SO_LINGER to 0 and reset the socket CLOSE: close the socket gracefully default is RESET.		MllpIdleTimeoutStrategy
camel.component.mllp.keep-alive	Enable/disable the SO_KEEPALIVE socket option.	true	Boolean
camel.component.mllp.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.mllp.lenient-bind	TCP Server Only - Allow the endpoint to start before the TCP ServerSocket is bound. In some environments, it may be desirable to allow the endpoint to start before the TCP ServerSocket is bound.	false	Boolean
camel.component.mllp.log-phi	Whether to log PHI.	true	Boolean
camel.component.mllp.log-phi-max-bytes	Set the maximum number of bytes of PHI that will be logged in a log entry.	5120	Integer
camel.component.mllp.max-concurrent-consumers	The maximum number of concurrent MLLP Consumer connections that will be allowed. If a new connection is received and the maximum is number are already established, the new connection will be reset immediately.	5	Integer
camel.component.mllp.read-timeout	The SO_TIMEOUT value (in milliseconds) used after the start of an MLLP frame has been received.	5000	Integer
camel.component.mllp.receive-buffer-size	Sets the SO_RCVBUF option to the specified value (in bytes).	8192	Integer

Name	Description	Default	Type
camel.component.mllp.receive-timeout	The SO_TIMEOUT value (in milliseconds) used when waiting for the start of an MLLP frame.	15000	Integer
camel.component.mllp.require-end-of-data	Enable/Disable strict compliance to the MLLP standard. The MLLP standard specifies START_OF_BLOCKhl7 payloadEND_OF_BLOCKEND_OF_DATA, however, some systems do not send the final END_OF_DATA byte. This setting controls whether or not the final END_OF_DATA byte is required or optional.	true	Boolean
camel.component.mllp.reuse-address	Enable/disable the SO_REUSEADDR socket option.	false	Boolean
camel.component.mllp.send-buffer-size	Sets the SO_SNDBUF option to the specified value (in bytes).	8192	Integer
camel.component.mllp.string-payload	Enable/Disable converting the payload to a String. If enabled, HL7 Payloads received from external systems will be validated converted to a String. If the charsetName property is set, that character set will be used for the conversion. If the charsetName property is not set, the value of MSH-18 will be used to determine th appropriate character set. If MSH-18 is not set, then the default ISO-8859-1 character set will be use.	true	Boolean
camel.component.mllp.tcp-no-delay	Enable/disable the TCP_NODELAY socket option.	true	Boolean
camel.component.mllp.validate-payload	Enable/Disable the validation of HL7 Payloads If enabled, HL7 Payloads received from external systems will be validated (see Hl7Util.generateInvalidPayloadExceptionMessage for details on the validation). If and invalid payload is detected, a MllpInvalidMessageException (for consumers) or a MllpInvalidAcknowledgementException will be thrown.	false	Boolean

CHAPTER 44. MOCK

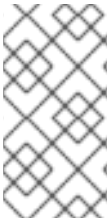
Only producer is supported

Testing of distributed and asynchronous processing is notoriously difficult. The Mock, Test and Dataset endpoints work great with the Camel Testing Framework to simplify your unit and integration testing using Enterprise Integration Patterns and Camel's large range of Components together with the powerful Bean Integration.

The Mock component provides a powerful declarative testing mechanism, which is similar to jMock in that it allows declarative expectations to be created on any Mock endpoint before a test begins. Then the test is run, which typically fires messages to one or more endpoints, and finally the expectations can be asserted in a test case to ensure the system worked as expected.

This allows you to test various things like:

- The correct number of messages are received on each endpoint,
- The correct payloads are received, in the right order,
- Messages arrive on an endpoint in order, using some Expression to create an order testing function,
- Messages arrive match some kind of Predicate such as that specific headers have certain values, or that messages match some predicate, such as by evaluating an XPath or XQuery Expression.



NOTE

There is also the Test endpoint which is a Mock endpoint, but which uses a second endpoint to provide the list of expected message bodies and automatically sets up the Mock endpoint assertions. In other words, it's a Mock endpoint that automatically sets up its assertions from some sample messages in a File or database, for example.

NOTE

Mock endpoints keep received Exchanges in memory indefinitely.
Remember that Mock is designed for testing. When you add Mock endpoints to a route, each Exchange sent to the endpoint will be stored (to allow for later validation) in memory until explicitly reset or the JVM is restarted. If you are sending high volume and/or large messages, this may cause excessive memory use. If your goal is to test deployable routes inline, consider using NotifyBuilder or AdviceWith in your tests instead of adding Mock endpoints to routes directly. There are two new options retainFirst, and retainLast that can be used to limit the number of messages the Mock endpoints keep in memory.

44.1. URI FORMAT

```
mock:someName[?options]
```

Where **someName** can be any string that uniquely identifies the endpoint.

44.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

44.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

44.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

44.3. COMPONENT OPTIONS

The Mock component supports 4 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
log (producer)	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
exchangeFormatter (advanced)	**Autowired** Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter.		ExchangeFormatter

44.4. ENDPOINT OPTIONS

The Mock endpoint is configured using URI syntax:

```
mock:name
```

with the following path and query parameters:

44.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (producer)	**Required** Name of mock endpoint.		String

44.4.2. Query Parameters (12 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
assertPeriod (producer)	Sets a grace period after which the mock endpoint will re-assert to ensure the preliminary assertion is still valid. This is used for example to assert that exactly a number of messages arrives. For example if expectedMessageCount(int) was set to 5, then the assertion is satisfied when 5 or more message arrives. To ensure that exactly 5 messages arrives, then you would need to wait a little period to ensure no further message arrives. This is what you can use this method for. By default this period is disabled.		long
expectedCount (producer)	Specifies the expected number of message exchanges that should be received by this endpoint. Beware: If you want to expect that 0 messages, then take extra care, as 0 matches when the tests starts, so you need to set a assert period time to let the test run for a while to make sure there are still no messages arrived; for that use setAssertPeriod(long). An alternative is to use NotifyBuilder, and use the notifier to know when Camel is done routing some messages, before you call the assertIsSatisfied() method on the mocks. This allows you to not use a fixed assert period, to speedup testing times. If you want to assert that exactly n'th message arrives to this mock endpoint, then see also the setAssertPeriod(long) method for further details.	-1	int
failFast (producer)	Sets whether assertIsSatisfied() should fail fast at the first detected failed expectation while it may otherwise wait for all expected messages to arrive before performing expectations verifications. Is by default true. Set to false to use behavior as in Camel 2.x.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
log (producer)	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	boolean
reportGroup (producer)	A number that is used to turn on throughput logging based on groups of the size.		int
resultMinimumWaitTime (producer)	Sets the minimum expected amount of time (in millis) the assertIsSatisfied() will wait on a latch until it is satisfied.		long
resultWaitTime (producer)	Sets the maximum amount of time (in millis) the assertIsSatisfied() will wait on a latch until it is satisfied.		long
retainFirst (producer)	Specifies to only retain the first n'th number of received Exchanges. This is used when testing with big data, to reduce memory consumption by not storing copies of every Exchange this mock endpoint receives. Important: When using this limitation, then the getReceivedCounter() will still return the actual number of received Exchanges. For example if we have received 5000 Exchanges, and have configured to only retain the first 10 Exchanges, then the getReceivedCounter() will still return 5000 but there is only the first 10 Exchanges in the getExchanges() and getReceivedExchanges() methods. When using this method, then some of the other expectation methods is not supported, for example the expectedBodiesReceived(Object...) sets a expectation on the first number of bodies received. You can configure both setRetainFirst(int) and setRetainLast(int) methods, to limit both the first and last received.	-1	int

Name	Description	Default	Type
retainLast (producer)	Specifies to only retain the last n'th number of received Exchanges. This is used when testing with big data, to reduce memory consumption by not storing copies of every Exchange this mock endpoint receives. Important: When using this limitation, then the getReceivedCounter() will still return the actual number of received Exchanges. For example if we have received 5000 Exchanges, and have configured to only retain the last 20 Exchanges, then the getReceivedCounter() will still return 5000 but there is only the last 20 Exchanges in the getExchanges() and getReceivedExchanges() methods. When using this method, then some of the other expectation methods is not supported, for example the expectedBodiesReceived(Object...) sets a expectation on the first number of bodies received. You can configure both setRetainFirst(int) and setRetainLast(int) methods, to limit both the first and last received.	-1	int
sleepForEmptyTest (producer)	Allows a sleep to be specified to wait to check that this endpoint really is empty when expectedMessageCount(int) is called with zero.		long
copyOnExchange (producer (advanced))	Sets whether to make a deep copy of the incoming Exchange when received at this mock endpoint. Is by default true.	true	boolean

44.5. SIMPLE EXAMPLE

Here's a simple example of Mock endpoint in use. First, the endpoint is resolved on the context. Then we set an expectation, and then, after the test has run, we assert that our expectations have been met:

```
MockEndpoint resultEndpoint = context.getEndpoint("mock:foo", MockEndpoint.class);

// set expectations
resultEndpoint.expectedMessageCount(2);

// send some messages

// now lets assert that the mock:foo endpoint received 2 messages
resultEndpoint.assertIsSatisfied();
```

You typically always call the method to test that the expectations were met after running a test.

Camel will by default wait 10 seconds when the **assertIsSatisfied()** is invoked. This can be configured by setting the **setResultWaitTime(millis)** method.

44.6. USING ASSERTPERIOD

When the assertion is satisfied then Camel will stop waiting and continue from the **assertIsSatisfied** method. That means if a new message arrives on the mock endpoint, just a bit later, that arrival will not affect the outcome of the assertion. Suppose you do want to test that no new messages arrives after a period thereafter, then you can do that by setting the **setAssertPeriod** method, for example:

```
MockEndpoint resultEndpoint = context.getEndpoint("mock:foo", MockEndpoint.class);
resultEndpoint.setAssertPeriod(5000);
resultEndpoint.expectedMessageCount(2);

// send some messages

// now lets assert that the mock:foo endpoint received 2 messages
resultEndpoint.assertIsSatisfied();
```

44.7. SETTING EXPECTATIONS

You can see from the Javadoc of MockEndpoint the various helper methods you can use to set expectations. The main methods are as follows:

Method	Description
expectedMessageCount(int)	To define the expected message count on the endpoint.
expectedMinimumMessageCount(int)	To define the minimum number of expected messages on the endpoint.
expectedBodiesReceived(...)	To define the expected bodies that should be received (in order).
expectedHeaderReceived(...)	To define the expected header that should be received
expectsAscending(Expression)	To add an expectation that messages are received in order, using the given Expression to compare messages.
expectsDescending(Expression)	To add an expectation that messages are received in order, using the given Expression to compare messages.
expectsNoDuplicates(Expression)	To add an expectation that no duplicate messages are received; using an Expression to calculate a unique identifier for each message. This could be something like the **JMSMessageID** if using JMS, or some unique reference number within the message.

Here's another example:

```
resultEndpoint.expectedBodiesReceived("firstMessageBody", "secondMessageBody",
"thirdMessageBody");
```

44.8. ADDING EXPECTATIONS TO SPECIFIC MESSAGES

In addition, you can use the **message(int messageIndex)** method to add assertions about a specific message that is received.

For example, to add expectations of the headers or body of the first message (using zero-based indexing like **java.util.List**), you can use the following code:

```
resultEndpoint.message(0).header("foo").isEqualTo("bar");
```

There are some examples of the Mock endpoint in use in the **camel-core** processor tests.

44.9. MOCKING EXISTING ENDPOINTS

Camel now allows you to automatically mock existing endpoints in your Camel routes.

NOTE

How it works
The endpoints are still in action. What happens differently is that a Mock endpoint is injected and receives the message first and then delegates the message to the target endpoint. You can view this as a kind of intercept and delegate or endpoint listener.

Suppose you have the given route below:

Route

```
@Override
protected RouteBuilder createRouteBuilder() throws Exception {
    return new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from("direct:start").routeId("start")
                    .to("direct:foo").to("log:foo").to("mock:result");

            from("direct:foo").routeId("foo")
                    .transform(constant("Bye World"));
        }
    };
}
```

You can then use the **adviceWith** feature in Camel to mock all the endpoints in a given route from your unit test, as shown below:

adviceWith mocking all endpoints

```
@Test
public void testAdvisedMockEndpoints() throws Exception {
    // advice the start route using the inlined AdviceWith lambda style route builder
    // which has extended capabilities than the regular route builder
    AdviceWith.adviceWith(context, "start", a ->
    // mock all endpoints
```

```
        a.mockEndpoints());

        getMockEndpoint("mock:direct:start").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:direct:foo").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:log:foo").expectedBodiesReceived("Bye World");
        getMockEndpoint("mock:result").expectedBodiesReceived("Bye World");

        template.sendBody("direct:start", "Hello World");

        assertMockEndpointsSatisfied();

        // additional test to ensure correct endpoints in registry
        assertNotNull(context.hasEndpoint("direct:start"));
        assertNotNull(context.hasEndpoint("direct:foo"));
        assertNotNull(context.hasEndpoint("log:foo"));
        assertNotNull(context.hasEndpoint("mock:result"));
        // all the endpoints was mocked
        assertNotNull(context.hasEndpoint("mock:direct:start"));
        assertNotNull(context.hasEndpoint("mock:direct:foo"));
        assertNotNull(context.hasEndpoint("mock:log:foo"));
    }
```

Notice that the mock endpoints is given the URI **mock:<endpoint>**, for example **mock:direct:foo**. Camel logs at **INFO** level the endpoints being mocked:

```
INFO  Adviced endpoint [direct://foo] with mock endpoint [mock:direct:foo]
```

NOTE

Mocked endpoints are without parameters
Endpoints which are mocked will have their parameters stripped off. For example the endpoint **log:foo?showAll=true** will be mocked to the following endpoint **mock:log:foo**. Notice the parameters have been removed.

Its also possible to only mock certain endpoints using a pattern. For example to mock all **log** endpoints you do as shown:

adviceWith mocking only log endpoints using a pattern

```
    @Test
    public void testAdvisedMockEndpointsWithPattern() throws Exception {
        // advice the start route using the inlined AdviceWith lambda style route builder
        // which has extended capabilities than the regular route builder
        AdviceWith.adviceWith(context, "start", a ->
        // mock only log endpoints
        a.mockEndpoints("log*"));

        // now we can refer to log:foo as a mock and set our expectations
        getMockEndpoint("mock:log:foo").expectedBodiesReceived("Bye World");

        getMockEndpoint("mock:result").expectedBodiesReceived("Bye World");

        template.sendBody("direct:start", "Hello World");
```

```
        assertMockEndpointsSatisfied();

        // additional test to ensure correct endpoints in registry
        assertNotNull(context.hasEndpoint("direct:start"));
        assertNotNull(context.hasEndpoint("direct:foo"));
        assertNotNull(context.hasEndpoint("log:foo"));
        assertNotNull(context.hasEndpoint("mock:result"));
        // only the log:foo endpoint was mocked
        assertNotNull(context.hasEndpoint("mock:log:foo"));
        assertNull(context.hasEndpoint("mock:direct:start"));
        assertNull(context.hasEndpoint("mock:direct:foo"));
    }
```

The pattern supported can be a wildcard or a regular expression. See more details about this at Intercept as its the same matching function used by Camel.

NOTE

Mind that mocking endpoints causes the messages to be copied when they arrive on the mock.
That means Camel will use more memory. This may not be suitable when you send in a lot of messages.

44.10. MOCKING EXISTING ENDPOINTS USING THE CAMEL-TEST COMPONENT

Instead of using the **adviceWith** to instruct Camel to mock endpoints, you can easily enable this behavior when using the **camel-test** Test Kit.

The same route can be tested as follows. Notice that we return **"*"** from the **isMockEndpoints** method, which tells Camel to mock all endpoints.

If you only want to mock all **log** endpoints you can return **"log*"** instead.

isMockEndpoints using camel-test kit

```
public class IsMockEndpointsJUnit4Test extends CamelTestSupport {

    @Override
    public String isMockEndpoints() {
        // override this method and return the pattern for which endpoints to mock.
        // use * to indicate all
        return "*";
    }

    @Test
    public void testMockAllEndpoints() throws Exception {
        // notice we have automatic mocked all endpoints and the name of the endpoints is "mock:uri"
        getMockEndpoint("mock:direct:start").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:direct:foo").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:log:foo").expectedBodiesReceived("Bye World");
        getMockEndpoint("mock:result").expectedBodiesReceived("Bye World");

        template.sendBody("direct:start", "Hello World");
```

```
        assertMockEndpointsSatisfied();

        // additional test to ensure correct endpoints in registry
        assertNotNull(context.hasEndpoint("direct:start"));
        assertNotNull(context.hasEndpoint("direct:foo"));
        assertNotNull(context.hasEndpoint("log:foo"));
        assertNotNull(context.hasEndpoint("mock:result"));
        // all the endpoints was mocked
        assertNotNull(context.hasEndpoint("mock:direct:start"));
        assertNotNull(context.hasEndpoint("mock:direct:foo"));
        assertNotNull(context.hasEndpoint("mock:log:foo"));
    }

    @Override
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from("direct:start").to("direct:foo").to("log:foo").to("mock:result");

                from("direct:foo").transform(constant("Bye World"));
            }
        };
    }
}
```

44.11. MOCKING EXISTING ENDPOINTS WITH XML DSL

If you do not use the **camel-test** component for unit testing (as shown above) you can use a different approach when using XML files for routes.
The solution is to create a new XML file used by the unit test and then include the intended XML file which has the route you want to test.

Suppose we have the route in the **camel-route.xml** file:

camel-route.xml

```
<!-- this camel route is in the camel-route.xml file -->
<camelContext xmlns="http://camel.apache.org/schema/spring">

    <route>
        <from uri="direct:start"/>
        <to uri="direct:foo"/>
        <to uri="log:foo"/>
        <to uri="mock:result"/>
    </route>

    <route>
        <from uri="direct:foo"/>
        <transform>
            <constant>Bye World</constant>
        </transform>
    </route>

</camelContext>
```

Then we create a new XML file as follows, where we include the **camel-route.xml** file and define a spring bean with the class **org.apache.camel.impl.InterceptSendToMockEndpointStrategy** which tells Camel to mock all endpoints:

test-camel-route.xml

```
    <!-- the Camel route is defined in another XML file -->
    <import resource="camel-route.xml"/>

    <!-- bean which enables mocking all endpoints -->
    <bean id="mockAllEndpoints"
class="org.apache.camel.component.mock.InterceptSendToMockEndpointStrategy"/>
```

Then in your unit test you load the new XML file (**test-camel-route.xml**) instead of **camel-route.xml**.

To only mock all Log endpoints you can define the pattern in the constructor for the bean:

```
<bean id="mockAllEndpoints"
class="org.apache.camel.impl.InterceptSendToMockEndpointStrategy">
    <constructor-arg index="0" value="log*"/>
</bean>
```

44.12. MOCKING ENDPOINTS AND SKIP SENDING TO ORIGINAL ENDPOINT

Sometimes you want to easily mock and skip sending to a certain endpoints. So the message is detoured and send to the mock endpoint only. You can now use the **mockEndpointsAndSkip** method using AdviceWith. The example below will skip sending to the two endpoints **"direct:foo"**, and **"direct:bar"**.

adviceWith mock and skip sending to endpoints

```
    @Test
    public void testAdvisedMockEndpointsWithSkip() throws Exception {
        // advice the first route using the inlined AdviceWith route builder
        // which has extended capabilities than the regular route builder
        AdviceWith.adviceWith(context.getRouteDefinitions().get(0), context, new
AdviceWithRouteBuilder() {
            @Override
            public void configure() throws Exception {
                // mock sending to direct:foo and direct:bar and skip send to it
                mockEndpointsAndSkip("direct:foo", "direct:bar");
            }
        });

        getMockEndpoint("mock:result").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:direct:foo").expectedMessageCount(1);
        getMockEndpoint("mock:direct:bar").expectedMessageCount(1);

        template.sendBody("direct:start", "Hello World");

        assertMockEndpointsSatisfied();

        // the message was not send to the direct:foo route and thus not sent to
        // the seda endpoint
```

```
        SedaEndpoint seda = context.getEndpoint("seda:foo", SedaEndpoint.class);
        assertEquals(0, seda.getCurrentQueueSize());
    }
```

The same example using the Test Kit

isMockEndpointsAndSkip using camel-test kit

```
public class IsMockEndpointsAndSkipJUnit4Test extends CamelTestSupport {

    @Override
    public String isMockEndpointsAndSkip() {
        // override this method and return the pattern for which endpoints to mock,
        // and skip sending to the original endpoint.
        return "direct:foo";
    }

    @Test
    public void testMockEndpointAndSkip() throws Exception {
        // notice we have automatic mocked the direct:foo endpoints and the name of the endpoints is
"mock:uri"
        getMockEndpoint("mock:result").expectedBodiesReceived("Hello World");
        getMockEndpoint("mock:direct:foo").expectedMessageCount(1);

        template.sendBody("direct:start", "Hello World");

        assertMockEndpointsSatisfied();

        // the message was not send to the direct:foo route and thus not sent to the seda endpoint
        SedaEndpoint seda = context.getEndpoint("seda:foo", SedaEndpoint.class);
        assertEquals(0, seda.getCurrentQueueSize());
    }

    @Override
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from("direct:start").to("direct:foo").to("mock:result");

                from("direct:foo").transform(constant("Bye World")).to("seda:foo");
            }
        };
    }
}
```

44.13. LIMITING THE NUMBER OF MESSAGES TO KEEP

The Mock endpoints will by default keep a copy of every Exchange that it received. So if you test with a lot of messages, then it will consume memory.
We have introduced two options **retainFirst** and **retainLast** that can be used to specify to only keep N'th of the first and/or last Exchanges.

For example in the code below, we only want to retain a copy of the first 5 and last 5 Exchanges the mock receives.

```
MockEndpoint mock = getMockEndpoint("mock:data");
mock.setRetainFirst(5);
mock.setRetainLast(5);
mock.expectedMessageCount(2000);

mock.assertIsSatisfied();
```

Using this has some limitations. The **getExchanges()** and **getReceivedExchanges()** methods on the **MockEndpoint** will return only the retained copies of the Exchanges. So in the example above, the list will contain 10 Exchanges; the first five, and the last five.
The **retainFirst** and **retainLast** options also have limitations on which expectation methods you can use. For example the **expectedXXX** methods that work on message bodies, headers, etc. will only operate on the retained messages. In the example above they can test only the expectations on the 10 retained messages.

44.14. TESTING WITH ARRIVAL TIMES

The Mock endpoint stores the arrival time of the message as a property on the Exchange

```
Date time = exchange.getProperty(Exchange.RECEIVED_TIMESTAMP, Date.class);
```

You can use this information to know when the message arrived on the mock. But it also provides foundation to know the time interval between the previous and next message arrived on the mock. You can use this to set expectations using the **arrives** DSL on the Mock endpoint.

For example to say that the first message should arrive between 0-2 seconds before the next you can do:

```
mock.message(0).arrives().noLaterThan(2).seconds().beforeNext();
```

You can also define this as that 2nd message (0 index based) should arrive no later than 0-2 seconds after the previous:

```
mock.message(1).arrives().noLaterThan(2).seconds().afterPrevious();
```

You can also use between to set a lower bound. For example suppose that it should be between 1-4 seconds:

```
mock.message(1).arrives().between(1, 4).seconds().afterPrevious();
```

You can also set the expectation on all messages, for example to say that the gap between them should be at most 1 second:

```
mock.allMessages().arrives().noLaterThan(1).seconds().beforeNext();
```

NOTE

Time units
In the example above we use **seconds** as the time unit, but Camel offers **milliseconds**, and **minutes** as well.

44.15. SPRING BOOT AUTO-CONFIGURATION

When using mock with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-mock-starter</artifactId>
</dependency>
```

The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.component.mock.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.mock.enabled	Whether to enable auto configuration of the mock component. This is enabled by default.		Boolean
camel.component.mock.exchange-formatter	Sets a custom ExchangeFormatter to convert the Exchange to a String suitable for logging. If not specified, we default to DefaultExchangeFormatter. The option is a org.apache.camel.spi.ExchangeFormatter type.		ExchangeFormatter
camel.component.mock.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.mock.log	To turn on logging when the mock receives an incoming message. This will log only one time at INFO level for the incoming message. For more detailed logging then set the logger to DEBUG level for the org.apache.camel.component.mock.MockEndpoint class.	false	Boolean

CHAPTER 45. MONGODB

Both producer and consumer are supported

According to Wikipedia: "NoSQL is a movement promoting a loosely defined class of non-relational data stores that break with a long history of relational databases and ACID guarantees." NoSQL solutions have grown in popularity in the last few years, and major extremely-used sites and services such as Facebook, LinkedIn, Twitter, etc. are known to use them extensively to achieve scalability and agility.

Basically, NoSQL solutions differ from traditional RDBMS (Relational Database Management Systems) in that they don't use SQL as their query language and generally don't offer ACID-like transactional behaviour nor relational data. Instead, they are designed around the concept of flexible data structures and schemas (meaning that the traditional concept of a database table with a fixed schema is dropped), extreme scalability on commodity hardware and blazing-fast processing.

MongoDB is a very popular NoSQL solution and the camel-mongodb component integrates Camel with MongoDB allowing you to interact with MongoDB collections both as a producer (performing operations on the collection) and as a consumer (consuming documents from a MongoDB collection).

MongoDB revolves around the concepts of documents (not as is office documents, but rather hierarchical data defined in JSON/BSON) and collections. This component page will assume you are familiar with them. Otherwise, visit http://www.mongodb.org/.

NOTE

The MongoDB Camel component uses Mongo Java Driver 4.x.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-mongodb</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

45.1. URI FORMAT

```
mongodb:connectionBean?
database=databaseName&collection=collectionName&operation=operationName[&moreOptions...]
```

45.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

45.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

45.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

45.3. COMPONENT OPTIONS

The MongoDB component supports 4 options, which are listed below.

Name	Description	Default	Type
mongoConnection (common)	**Autowired** Shared client used for connection. All endpoints generated from the component will share this connection client.		MongoClient
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

45.4. ENDPOINT OPTIONS

The MongoDB endpoint is configured using URI syntax:

```
mongodb:connectionBean
```

with the following path and query parameters:

45.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
connectionBean (common)	**Required** Sets the connection bean reference used to lookup a client for connecting to a database.		String

45.4.2. Query Parameters (27 parameters)

Name	Description	Default	Type
collection (common)	Sets the name of the MongoDB collection to bind to this endpoint.		String
collectionIndex (common)	Sets the collection index (JSON FORMAT : \\{ field1 : order1, field2 : order2}).		String

Name	Description	Default	Type
createCollection (common)	Create collection during initialisation if it doesn't exist. Default is true.	true	boolean
database (common)	Sets the name of the MongoDB database to target.		String
hosts (common)	Host address of mongodb server in host:port format. It's possible also use more than one address, as comma separated list of hosts: host1:port1,host2:port2. If hosts parameter is specified, provided connectionBean is ignored.		String
mongoConnection (common)	Sets the connection bean used as a client for connecting to a database.		MongoClient
operation (common)	Sets the operation this endpoint will execute against MongoDB. Enum values: • findById • findOneByQuery • findAll • findDistinct • insert • save • update • remove • bulkWrite • aggregate • getDbStats • getColStats • count • command		MongoDbOperation

Name	Description	Default	Type
outputType (common)	Convert the output of the producer to the selected type : DocumentList Document or MongoIterable. DocumentList or MongoIterable applies to findAll and aggregate. Document applies to all other operations. Enum values: ● DocumentList ● Document ● MongoIterable		MongoDbOutputType
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
consumerType (consumer)	Consumer type.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
cursorRegenerationDelay (advanced)	MongoDB tailable cursors will block until new data arrives. If no new data is inserted, after some time the cursor will be automatically freed and closed by the MongoDB server. The client is expected to regenerate the cursor if needed. This value specifies the time to wait before attempting to fetch a new cursor, and if the attempt fails, how long before the next attempt is made. Default value is 1000ms.	1000	long
dynamicity (advanced)	Sets whether this endpoint will attempt to dynamically resolve the target database and collection from the incoming Exchange properties. Can be used to override at runtime the database and collection specified on the otherwise static endpoint URI. It is disabled by default to boost performance. Enabling it will take a minimal performance hit.	false	boolean
readPreference (advanced)	Configure how MongoDB clients route read operations to the members of a replica set. Possible values are PRIMARY, PRIMARY_PREFERRED, SECONDARY, SECONDARY_PREFERRED or NEAREST. Enum values: • PRIMARY • PRIMARY_PREFERRED • SECONDARY • SECONDARY_PREFERRED • NEAREST	PRIMARY	String

Name	Description	Default	Type
writeConcern (advanced)	Configure the connection bean with the level of acknowledgment requested from MongoDB for write operations to a standalone mongod, replicaset or cluster. Possible values are ACKNOWLEDGED, W1, W2, W3, UNACKNOWLEDGED, JOURNALED or MAJORITY. Enum values: • ACKNOWLEDGED • W1 • W2 • W3 • UNACKNOWLEDGED • JOURNALED • MAJORITY	ACKNOWLEDGED	String
writeResultAsHeader (advanced)	In write operations, it determines whether instead of returning WriteResult as the body of the OUT message, we transfer the IN message to the OUT and attach the WriteResult as a header.	false	boolean
streamFilter (changeStream)	Filter condition for change streams consumer.		String
password (security)	User password for mongodb connection.		String
username (security)	Username for mongodb connection.		String
persistentId (tail)	One tail tracking collection can host many trackers for several tailable consumers. To keep them separate, each tracker should have its own unique persistentId.		String
persistentTailTracking (tail)	Enable persistent tail tracking, which is a mechanism to keep track of the last consumed message across system restarts. The next time the system is up, the endpoint will recover the cursor from the point where it last stopped slurping records.	false	boolean

Name	Description	Default	Type
tailTrackCollection (tail)	Collection where tail tracking information will be persisted. If not specified, MongoDbTailTrackingConfig#DEFAULT_COLLECTION will be used by default.		String
tailTrackDb (tail)	Indicates what database the tail tracking mechanism will persist to. If not specified, the current database will be picked by default. Dynamicity will not be taken into account even if enabled, i.e. the tail tracking database will not vary past endpoint initialisation.		String
tailTrackField (tail)	Field where the last tracked value will be placed. If not specified, MongoDbTailTrackingConfig#DEFAULT_FIELD will be used by default.		String
tailTrackIncreasingField (tail)	Correlation field in the incoming record which is of increasing nature and will be used to position the tailing cursor every time it is generated. The cursor will be (re)created with a query of type: tailTrackIncreasingField greater than lastValue (possibly recovered from persistent tail tracking). Can be of type Integer, Date, String, etc. NOTE: No support for dot notation at the current time, so the field should be at the top level of the document.		String

45.5. CONFIGURATION OF DATABASE IN SPRING XML

The following Spring XML creates a bean defining the connection to a MongoDB instance.

Since mongo java driver 3, the WriteConcern and readPreference options are not dynamically modifiable. They are defined in the mongoClient object

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

  <mongo:mongo-client id="mongoBean" host="${mongo.url}" port="${mongo.port}"
credentials="${mongo.user}:${mongo.pass}@${mongo.dbname}">
    <mongo:client-options write-concern="NORMAL" />
  </mongo:mongo-client>
</beans>
```

45.6. SAMPLE ROUTE

The following route defined in Spring XML executes the operation **getDbStats** on a collection.

Get DB stats for specified collection

```
<route>
  <from uri="direct:start" />
  <!-- using bean 'mongoBean' defined above -->
  <to uri="mongodb:mongoBean?
database=${mongodb.database}&amp;collection=${mongodb.collection}&amp;operation=getDbStats"
/>
  <to uri="direct:result" />
</route>
```

45.7. MONGODB OPERATIONS - PRODUCER ENDPOINTS

45.7.1. Query operations

45.7.1.1. findById

This operation retrieves only one element from the collection whose _id field matches the content of the IN message body. The incoming object can be anything that has an equivalent to a **Bson** type. See http://bsonspec.org/spec.html and http://www.mongodb.org/display/DOCS/Java+Types.

```
from("direct:findById")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findById")
    .to("mock:resultFindById");
```

Please, note that the default _id is treated by Mongo as and **ObjectId** type, so you may need to convert it properly.

```
from("direct:findById")
    .convertBodyTo(ObjectId.class)
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findById")
    .to("mock:resultFindById");
```

NOTE

Supports optional parameters
This operation supports projection operators. See Specifying a fields filter (projection) .

45.7.1.2. findOneByQuery

Retrieve the first element from a collection matching a MongoDB query selector. **If the CamelMongoDbCriteria header is set, then its value is used as the query selector**If the **CamelMongoDbCriteria** header is *null*, then the IN message body is used as the query selector. In both cases, the query selector should be of type **Bson** or convertible to **Bson** (for instance, a JSON string or **HashMap**). See Type conversions for more info.

Create query selectors using the **Filters** provided by the MongoDB Driver.

45.7.1.3. Example without a query selector (returns the first document in a collection)

```
from("direct:findOneByQuery")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

45.7.1.4. Example with a query selector (returns the first matching document in a collection):

```
from("direct:findOneByQuery")
    .setHeader(MongoDbConstants.CRITERIA, constant(Filters.eq("name", "Raul Kripalani")))
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findOneByQuery")
    .to("mock:resultFindOneByQuery");
```

NOTE

Supports optional parameters
This operation supports projection operators and sort clauses. See Specifying a fields filter (projection), Specifying a sort clause.

45.7.1.5. findAll

The **findAll** operation returns all documents matching a query, or none at all, in which case all documents contained in the collection are returned. **The query object is extracted CamelMongoDbCriteria header**. if the CamelMongoDbCriteria header is null the query object is extracted message body, i.e. it should be of type **Bson** or convertible to **Bson**. It can be a JSON String or a Hashmap. See Type conversions for more info.

45.7.1.5.1. Example without a query selector (returns all documents in a collection)

```
from("direct:findAll")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

45.7.1.5.2. Example with a query selector (returns all matching documents in a collection)

```
from("direct:findAll")
    .setHeader(MongoDbConstants.CRITERIA, Filters.eq("name", "Raul Kripalani"))
    .to("mongodb:myDb?database=flights&collection=tickets&operation=findAll")
    .to("mock:resultFindAll");
```

Paging and efficient retrieval is supported via the following headers:

Header key	Quick constant	Description (extracted from MongoDB API doc)	Expected type
CamelMongoDbNumToSkip	**MongoDbConstants.NUM_TO_SKIP**	Discards a given number of elements at the beginning of the cursor.	int/Integer

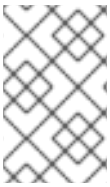
Header key	Quick constant	Description (extracted from MongoDB API doc)	Expected type
CamelMongoDbLimit	**MongoDbConstants.LIMIT**	Limits the number of elements returned.	int/Integer
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	Limits the number of elements returned in one batch. A cursor typically fetches a batch of result objects and store them locally. If batchSize is positive, it represents the size of each batch of objects retrieved. It can be adjusted to optimize performance and limit data transfer. If batchSize is negative, it will limit of number objects returned, that fit within the max batch size limit (usually 4MB), and cursor will be closed. For example if batchSize is -10, then the server will return a maximum of 10 documents and as many as can fit in 4MB, then close the cursor. Note that this feature is different from limit() in that documents must fit within a maximum size, and it removes the need to send a request to close the cursor server-side. The batch size can be changed even after a cursor is iterated, in which case the setting will apply on the next batch retrieval.	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	Sets allowDiskUse MongoDB flag. This is supported since MongoDB Server 4.3.1. Using this header with older MongoDB Server version can cause query to fail.	boolean/Boolean

45.7.1.5.3. Example with option *outputType=MongoIterable* and batch size

```
from("direct:findAll")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setHeader(MongoDbConstants.CRITERIA, constant(Filters.eq("name", "Raul Kripalani")))
    .to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll&outputType=MongoIterable")
    .to("mock:resultFindAll");
```

The **findAll** operation will also return the following OUT headers to enable you to iterate through result pages if you are using paging:

Header key	Quick constant	Description (extracted from MongoDB API doc)	Data type
CamelMongoDbResultTotalSize	**MongoDbConstants.RESULT_TOTAL_SIZE**	Number of objects matching the query. This does not take limit/skip into consideration.	int/Integer
CamelMongoDbResultPageSize	**MongoDbConstants.RESULT_PAGE_SIZE**	Number of objects matching the query. This does not take limit/skip into consideration.	int/Integer



NOTE

Supports optional parameters
This operation supports projection operators and sort clauses. See Specifying a fields filter (projection), Specifying a sort clause.

45.7.1.6. count

Returns the total number of objects in a collection, returning a Long as the OUT message body. The following example will count the number of records in the "dynamicCollectionName" collection. Notice how dynamicity is enabled, and as a result, the operation will not run against the "notableScientists" collection, but against the "dynamicCollectionName" collection.

```
// from("direct:count").to("mongodb:myDb?
database=tickets&collection=flights&operation=count&dynamicity=true");
Long result = template.requestBodyAndHeader("direct:count", "irrelevantBody",
MongoDbConstants.COLLECTION, "dynamicCollectionName");
assertTrue("Result is not of type Long", result instanceof Long);
```

You can provide a query **The query object is extracted CamelMongoDbCriteria header**. if the CamelMongoDbCriteria header is null the query object is extracted message body, i.e. it should be of type **Bson** or convertible to **Bson**., and operation will return the amount of documents matching this criteria.

```
Document query = ...
Long count = template.requestBodyAndHeader("direct:count", query,
MongoDbConstants.COLLECTION, "dynamicCollectionName");
```

45.7.1.7. Specifying a fields filter (projection)

Query operations will, by default, return the matching objects in their entirety (with all their fields). If your documents are large and you only require retrieving a subset of their fields, you can specify a field filter in all query operations, simply by setting the relevant **Bson** (or type convertible to **Bson**, such as a JSON String, Map, etc.) on the **CamelMongoDbFieldsProjection** header, constant shortcut: **MongoDbConstants.FIELDS_PROJECTION**.

Here is an example that uses MongoDB's **Projections** to simplify the creation of Bson. It retrieves all fields except **_id** and **boringField**:

```
// route: from("direct:findAll").to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

Here is an example that uses MongoDB's **Projections** to simplify the creation of Bson. It retrieves all fields except **_id** and **boringField**:

```
// route: from("direct:findAll").to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll")
Bson fieldProjection = Projection.exclude("_id", "boringField");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.FIELDS_PROJECTION, fieldProjection);
```

45.7.1.8. Specifying a sort clause

There is a often a requirement to fetch the min/max record from a collection based on sorting by a particular field that uses MongoDB's **Sorts** to simplify the creation of Bson. It retrieves all fields except **_id** and **boringField**:

```
// route: from("direct:findAll").to("mongodb:myDb?
database=flights&collection=tickets&operation=findAll")
Bson sorts = Sorts.descending("_id");
Object result = template.requestBodyAndHeader("direct:findAll", ObjectUtils.NULL,
MongoDbConstants.SORT_BY, sorts);
```

In a Camel route the SORT_BY header can be used with the findOneByQuery operation to achieve the same result. If the FIELDS_PROJECTION header is also specified the operation will return a single field/value pair that can be passed directly to another component (for example, a parameterized MyBatis SELECT query). This example demonstrates fetching the temporally newest document from a collection and reducing the result to a single field, based on the **documentTimestamp** field:

```
.from("direct:someTriggeringEvent")
.setHeader(MongoDbConstants.SORT_BY).constant(Sorts.descending("documentTimestamp"))
.setHeader(MongoDbConstants.FIELDS_PROJECTION).constant(Projection.include("documentTime
stamp"))
.setBody().constant("{}")
.to("mongodb:myDb?database=local&collection=myDemoCollection&operation=findOneByQuery")
.to("direct:aMyBatisParameterizedSelect");
```

45.7.2. Create/update operations

45.7.2.1. insert

Inserts an new object into the MongoDB collection, taken from the IN message body. Type conversion is attempted to turn it into **Document** or a **List**.
Two modes are supported: single insert and multiple insert. For multiple insert, the endpoint will expect a List, Array or Collections of objects of any type, as long as they are - or can be converted to - **Document**. Example:

```
from("direct:insert")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=insert");
```

The operation will return a WriteResult, and depending on the **WriteConcern** or the value of the **invokeGetLastError** option, **getLastError()** would have been called already or not. If you want to access the ultimate result of the write operation, you need to retrieve the **CommandResult** by calling **getLastError()** or **getCachedLastError()** on the **WriteResult**. Then you can verify the result by calling **CommandResult.ok()**, **CommandResult.getErrorMessage()** and/or **CommandResult.getException()**.

Note that the new object's **_id** must be unique in the collection. If you don't specify the value, MongoDB will automatically generate one for you. But if you do specify it and it is not unique, the insert operation will fail (and for Camel to notice, you will need to enable invokeGetLastError or set a WriteConcern that waits for the write result).

This is not a limitation of the component, but it is how things work in MongoDB for higher throughput. If you are using a custom **_id**, you are expected to ensure at the application level that is unique (and this is a good practice too).

OID(s) of the inserted record(s) is stored in the message header under **CamelMongoOid** key (**MongoDbConstants.OID** constant). The value stored is **org.bson.types.ObjectId** for single insert or **java.util.List<org.bson.types.ObjectId>** if multiple records have been inserted.

In MongoDB Java Driver 3.x the insertOne and insertMany operation return void. The Camel insert operation return the Document or List of Documents inserted. Note that each Documents are Updated by a new OID if need.

45.7.2.2. save

The save operation is equivalent to an *upsert* (UPdate, inSERT) operation, where the record will be updated, and if it doesn't exist, it will be inserted, all in one atomic operation. MongoDB will perform the matching based on the **_id** field.

Beware that in case of an update, the object is replaced entirely and the usage of MongoDB's $modifiers is not permitted. Therefore, if you want to manipulate the object if it already exists, you have two options:

1. perform a query to retrieve the entire object first along with all its fields (may not be efficient), alter it inside Camel and then save it.

2. use the update operation with $modifiers, which will execute the update at the server-side instead. You can enable the upsert flag, in which case if an insert is required, MongoDB will apply the $modifiers to the filter query object and insert the result.

If the document to be saved does not contain the **_id** attribute, the operation will be an insert, and the new **_id** created will be placed in the **CamelMongoOid** header.

For example:

```
from("direct:insert")
    .to("mongodb:myDb?database=flights&collection=tickets&operation=save");
```

```
// route: from("direct:insert").to("mongodb:myDb?
database=flights&collection=tickets&operation=save");
org.bson.Document docForSave = new org.bson.Document();
docForSave.put("key", "value");
Object result = template.requestBody("direct:insert", docForSave);
```

45.7.2.3. update

Update one or multiple records on the collection. Requires a filter query and a update rules.

You can define the filter using MongoDBConstants.CRITERIA header as **Bson** and define the update rules as **Bson** in Body.

NOTE

Update after enrich
While defining the filter by using MongoDBConstants.CRITERIA header as **Bson** to query mongodb before you do update, you should notice you need to remove it from the resulting camel exchange during aggregation if you use enrich pattern with a aggregation strategy and then apply mongodb update. If you don't remove this header during aggregation and/or redefine MongoDBConstants.CRITERIA header before sending camel exchange to mongodb producer endpoint, you may end up with invalid camel exchange payload while updating mongodb.

The second way Require a List<Bson> as the IN message body containing exactly 2 elements:

- Element 1 (index 0) ⇒ filter query ⇒ determines what objects will be affected, same as a typical query object
- Element 2 (index 1) ⇒ update rules ⇒ how matched objects will be updated. All modifier operations from MongoDB are supported.

NOTE

Multiupdates
By default, MongoDB will only update 1 object even if multiple objects match the filter query. To instruct MongoDB to update **all** matching records, set the **CamelMongoDbMultiUpdate** IN message header to **true**.

A header with key **CamelMongoDbRecordsAffected** will be returned (**MongoDbConstants.RECORDS_AFFECTED** constant) with the number of records updated (copied from **WriteResult.getN()**).

Supports the following IN message headers:

Header key	Quick constant	Description (extracted from MongoDB API doc)	Expected type
CamelMongoDbMultiUpdate	**MongoDbConstants.MULTIUPDATE**	If the update should be applied to all objects matching. See http://www.mongodb.org/display/DOCS/Atomic+Operations	boolean/Boolean
CamelMongoDbUpsert	**MongoDbConstants.UPSERT**	If the database should create the element if it does not exist	boolean/Boolean

For example, the following will update **all** records whose filterField field equals true by setting the value of the "scientist" field to "Darwin":

```
// route: from("direct:update").to("mongodb:myDb?database=science&collection=notableScientists&operation=update");
List<Bson> body = new ArrayList<>();
Bson filterField = Filters.eq("filterField", true);
body.add(filterField);
BsonDocument updateObj = new BsonDocument().append("$set", new BsonDocument("scientist", new BsonString("Darwin")));
body.add(updateObj);
Object result = template.requestBodyAndHeader("direct:update", body, MongoDbConstants.MULTIUPDATE, true);
```

```
// route: from("direct:update").to("mongodb:myDb?database=science&collection=notableScientists&operation=update");
Maps<String, Object> headers = new HashMap<>(2);
headers.add(MongoDbConstants.MULTIUPDATE, true);
headers.add(MongoDbConstants.FIELDS_FILTER, Filters.eq("filterField", true));
String updateObj = Updates.set("scientist", "Darwin");;
Object result = template.requestBodyAndHeaders("direct:update", updateObj, headers);
```

```
// route: from("direct:update").to("mongodb:myDb?database=science&collection=notableScientists&operation=update");
String updateObj = "[{\"filterField\": true}, {\"$set\", {\"scientist\", \"Darwin\"}}]";
Object result = template.requestBodyAndHeader("direct:update", updateObj, MongoDbConstants.MULTIUPDATE, true);
```

45.7.3. Delete operations

45.7.3.1. remove

Remove matching records from the collection. The IN message body will act as the removal filter query, and is expected to be of type **DBObject** or a type convertible to it.
The following example will remove all objects whose field 'conditionField' equals true, in the science database, notableScientists collection:

```
// route: from("direct:remove").to("mongodb:myDb?
database=science&collection=notableScientists&operation=remove");
Bson conditionField = Filters.eq("conditionField", true);
Object result = template.requestBody("direct:remove", conditionField);
```

A header with key **CamelMongoDbRecordsAffected** is returned (**MongoDbConstants.RECORDS_AFFECTED** constant) with type **int**, containing the number of records deleted (copied from **WriteResult.getN()**).

45.7.4. Bulk Write Operations

45.7.4.1. bulkWrite

Performs write operations in bulk with controls for order of execution. Requires a **List<WriteModel<Document>>** as the IN message body containing commands for insert, update, and delete operations.

The following example will insert a new scientist "Pierre Curie", update record with id "5" by setting the value of the "scientist" field to "Marie Curie" and delete record with id "3" :

```
// route: from("direct:bulkWrite").to("mongodb:myDb?
database=science&collection=notableScientists&operation=bulkWrite");
List<WriteModel<Document>> bulkOperations = Arrays.asList(
        new InsertOneModel<>(new Document("scientist", "Pierre Curie")),
        new UpdateOneModel<>(new Document("_id", "5"),
                new Document("$set", new Document("scientist", "Marie Curie"))),
        new DeleteOneModel<>(new Document("_id", "3")));

BulkWriteResult result = template.requestBody("direct:bulkWrite", bulkOperations,
BulkWriteResult.class);
```

By default, operations are executed in order and interrupted on the first write error without processing any remaining write operations in the list. To instruct MongoDB to continue to process remaining write operations in the list, set the **CamelMongoDbBulkOrdered** IN message header to **false**. Unordered operations are executed in parallel and this behavior is not guaranteed.

Header key	Quick constant	Description (extracted from MongoDB API doc)	Expected type
CamelMongoDbBulkOrdered	**MongoDbConstants.BULK_ORDERED**	Perform an ordered or unordered operation execution. Defaults to true.	boolean/Boolean

45.7.5. Other operations

45.7.5.1. aggregate

Perform a aggregation with the given pipeline contained in the body. **Aggregations could be long and heavy operations. Use with care.**

```
// route: from("direct:aggregate").to("mongodb:myDb?
```

```
database=science&collection=notableScientists&operation=aggregate");
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
        group("$scientist", sum("count", 1)));
from("direct:aggregate")
    .setBody().constant(aggregate)
    .to("mongodb:myDb?database=science&collection=notableScientists&operation=aggregate")
    .to("mock:resultAggregate");
```

Supports the following IN message headers:

Header key	Quick constant	Description (extracted from MongoDB API doc)	Expected type
CamelMongoDbBatchSize	**MongoDbConstants.BATCH_SIZE**	Sets the number of documents to return per batch.	int/Integer
CamelMongoDbAllowDiskUse	**MongoDbConstants.ALLOW_DISK_USE**	Enable aggregation pipeline stages to write data to temporary files.	boolean/Boolean

By default a List of all results is returned. This can be heavy on memory depending on the size of the results. A safer alternative is to set your outputType=MongoIterable. The next Processor will see an iterable in the message body allowing it to step through the results one by one. Thus setting a batch size and returning an iterable allows for efficient retrieval and processing of the result.

An example would look like:

```
List<Bson> aggregate = Arrays.asList(match(or(eq("scientist", "Darwin"), eq("scientist",
        group("$scientist", sum("count", 1)));
from("direct:aggregate")
    .setHeader(MongoDbConstants.BATCH_SIZE).constant(10)
    .setBody().constant(aggregate)
    .to("mongodb:myDb?
database=science&collection=notableScientists&operation=aggregate&outputType=MongoIterable")
    .split(body())
    .streaming()
    .to("mock:resultAggregate");
```

Note that calling **.split(body())** is enough to send the entries down the route one-by-one, however it would still load all the entries into memory first. Calling **.streaming()** is thus required to load data into memory by batches.

45.7.5.2. getDbStats

Equivalent of running the **db.stats()** command in the MongoDB shell, which displays useful statistic figures about the database.
For example:

```
> db.stats();
{
    "db" : "test",
```

```
    "collections" : 7,
    "objects" : 719,
    "avgObjSize" : 59.73296244784423,
    "dataSize" : 42948,
    "storageSize" : 1000058880,
    "numExtents" : 9,
    "indexes" : 4,
    "indexSize" : 32704,
    "fileSize" : 1275068416,
    "nsSizeMB" : 16,
    "ok" : 1
}
```

Usage example:

```
// from("direct:getDbStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getDbStats");
Object result = template.requestBody("direct:getDbStats", "irrelevantBody");
assertTrue("Result is not of type Document", result instanceof Document);
```

The operation will return a data structure similar to the one displayed in the shell, in the form of a **Document** in the OUT message body.

45.7.5.3. getColStats

Equivalent of running the **db.collection.stats()** command in the MongoDB shell, which displays useful statistic figures about the collection.
For example:

```
> db.camelTest.stats();
{
    "ns" : "test.camelTest",
    "count" : 100,
    "size" : 5792,
    "avgObjSize" : 57.92,
    "storageSize" : 20480,
    "numExtents" : 2,
    "nindexes" : 1,
    "lastExtentSize" : 16384,
    "paddingFactor" : 1,
    "flags" : 1,
    "totalIndexSize" : 8176,
    "indexSizes" : {
        "_id_" : 8176
    },
    "ok" : 1
}
```

Usage example:

```
// from("direct:getColStats").to("mongodb:myDb?
database=flights&collection=tickets&operation=getColStats");
Object result = template.requestBody("direct:getColStats", "irrelevantBody");
assertTrue("Result is not of type Document", result instanceof Document);
```

The operation will return a data structure similar to the one displayed in the shell, in the form of a **Document** in the OUT message body.

45.7.5.4. command

Run the body as a command on database. Useful for admin operation as getting host information, replication or sharding status.

Collection parameter is not use for this operation.

```
// route: from("command").to("mongodb:myDb?database=science&operation=command");
DBObject commandBody = new BasicDBObject("hostInfo", "1");
Object result = template.requestBody("direct:command", commandBody);
```

45.7.6. Dynamic operations

An Exchange can override the endpoint's fixed operation by setting the **CamelMongoDbOperation** header, defined by the **MongoDbConstants.OPERATION_HEADER** constant.
The values supported are determined by the MongoDbOperation enumeration and match the accepted values for the **operation** parameter on the endpoint URI.

For example:

```
// from("direct:insert").to("mongodb:myDb?database=flights&collection=tickets&operation=insert");
Object result = template.requestBodyAndHeader("direct:insert", "irrelevantBody",
MongoDbConstants.OPERATION_HEADER, "count");
assertTrue("Result is not of type Long", result instanceof Long);
```

45.8. CONSUMERS

There are several types of consumers:

1. Tailable Cursor Consumer
2. Change Streams Consumer

45.8.1. Tailable Cursor Consumer

MongoDB offers a mechanism to instantaneously consume ongoing data from a collection, by keeping the cursor open just like the **tail -f** command of *nix systems. This mechanism is significantly more efficient than a scheduled poll, due to the fact that the server pushes new data to the client as it becomes available, rather than making the client ping back at scheduled intervals to fetch new data. It also reduces otherwise redundant network traffic.

There is only one requisite to use tailable cursors: the collection must be a "capped collection", meaning that it will only hold N objects, and when the limit is reached, MongoDB flushes old objects in the same order they were originally inserted. For more information, please refer to http://www.mongodb.org/display/DOCS/Tailable+Cursors.

The Camel MongoDB component implements a tailable cursor consumer, making this feature available for you to use in your Camel routes. As new objects are inserted, MongoDB will push them as **Document** in natural order to your tailable cursor consumer, who will transform them to an Exchange and will trigger your route logic.

45.9. HOW THE TAILABLE CURSOR CONSUMER WORKS

To turn a cursor into a tailable cursor, a few special flags are to be signalled to MongoDB when first generating the cursor. Once created, the cursor will then stay open and will block upon calling the **MongoCursor.next()** method until new data arrives. However, the MongoDB server reserves itself the right to kill your cursor if new data doesn't appear after an indeterminate period. If you are interested to continue consuming new data, you have to regenerate the cursor. And to do so, you will have to remember the position where you left off or else you will start consuming from the top again.

The Camel MongoDB tailable cursor consumer takes care of all these tasks for you. You will just need to provide the key to some field in your data of increasing nature, which will act as a marker to position your cursor every time it is regenerated, e.g. a timestamp, a sequential ID, etc. It can be of any datatype supported by MongoDB. Date, Strings and Integers are found to work well. We call this mechanism "tail tracking" in the context of this component.

The consumer will remember the last value of this field and whenever the cursor is to be regenerated, it will run the query with a filter like: **increasingField > lastValue**, so that only unread data is consumed.

Setting the increasing field: Set the key of the increasing field on the endpoint URI **tailTrackingIncreasingField** option. In Camel 2.10, it must be a top-level field in your data, as nested navigation for this field is not yet supported. That is, the "timestamp" field is okay, but "nested.timestamp" will not work. Please open a ticket in the Camel JIRA if you do require support for nested increasing fields.

Cursor regeneration delay: One thing to note is that if new data is not already available upon initialisation, MongoDB will kill the cursor instantly. Since we don't want to overwhelm the server in this case, a **cursorRegenerationDelay** option has been introduced (with a default value of 1000ms.), which you can modify to suit your needs.

An example:

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime")
    .id("tailableCursorConsumer1")
    .autoStartup(false)
    .to("mock:test");
```

The above route will consume from the "flights.cancellations" capped collection, using "departureTime" as the increasing field, with a default regeneration cursor delay of 1000ms.

45.10. PERSISTENT TAIL TRACKING

Standard tail tracking is volatile and the last value is only kept in memory. However, in practice you will need to restart your Camel container every now and then, but your last value would then be lost and your tailable cursor consumer would start consuming from the top again, very likely sending duplicate records into your route.

To overcome this situation, you can enable the **persistent tail tracking** feature to keep track of the last consumed increasing value in a special collection inside your MongoDB database too. When the consumer initialises again, it will restore the last tracked value and continue as if nothing happened.

The last read value is persisted on two occasions: every time the cursor is regenerated and when the consumer shuts down. We may consider persisting at regular intervals too in the future (flush every 5 seconds) for added robustness if the demand is there. To request this feature, please open a ticket in the Camel JIRA.

45.11. ENABLING PERSISTENT TAIL TRACKING

To enable this function, set at least the following options on the endpoint URI:

- **persistentTailTracking** option to **true**
- **persistentId** option to a unique identifier for this consumer, so that the same collection can be reused across many consumers

Additionally, you can set the **tailTrackDb**, **tailTrackCollection** and **tailTrackField** options to customise where the runtime information will be stored. Refer to the endpoint options table at the top of this page for descriptions of each option.

For example, the following route will consume from the "flights.cancellations" capped collection, using "departureTime" as the increasing field, with a default regeneration cursor delay of 1000ms, with persistent tail tracking turned on, and persisting under the "cancellationsTracker" id on the "flights.camelTailTracking", storing the last processed value under the "lastTrackingValue" field (**camelTailTracking** and **lastTrackingValue** are defaults).

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTrack
ng=true" +
    "&persistentId=cancellationsTracker")
  .id("tailableCursorConsumer2")
  .autoStartup(false)
  .to("mock:test");
```

Below is another example identical to the one above, but where the persistent tail tracking runtime information will be stored under the "trackers.camelTrackers" collection, in the "lastProcessedDepartureTime" field:

```
from("mongodb:myDb?
database=flights&collection=cancellations&tailTrackIncreasingField=departureTime&persistentTailTrack
ng=true" +
    "&persistentId=cancellationsTracker&tailTrackDb=trackers&tailTrackCollection=camelTrackers" +
    "&tailTrackField=lastProcessedDepartureTime")
  .id("tailableCursorConsumer3")
  .autoStartup(false)
  .to("mock:test");
```

45.11.1. Change Streams Consumer

Change Streams allow applications to access real-time data changes without the complexity and risk of tailing the MongoDB oplog. Applications can use change streams to subscribe to all data changes on a collection and immediately react to them. Because change streams use the aggregation framework, applications can also filter for specific changes or transform the notifications at will. The exchange body will contain the full document of any change.

To configure Change Streams Consumer you need to specify **consumerType**, **database**, **collection** and optional JSON property **streamFilter** to filter events. That JSON property is standard MongoDB **$match** aggregation. It could be easily specified using XML DSL configuration:

```
<route id="filterConsumer">
  <from uri="mongodb:myDb?
consumerType=changeStreams&amp;database=flights&amp;collection=tickets&amp;streamFilter={
```

```
'$match':{'$or':[{'fullDocument.stringValue': 'specificValue'}]} }"/>
    <to uri="mock:test"/>
</route>
```

Java configuration:

```
from("mongodb:myDb?
consumerType=changeStreams&database=flights&collection=tickets&streamFilter={ '$match':{'$or':
[{'fullDocument.stringValue': 'specificValue'}]} }")
    .to("mock:test");
```

NOTE

You can externalize the streamFilter value into a property placeholder which allows the endpoint URI parameters to be *cleaner* and easier to read.

The **changeStreams** consumer type will also return the following OUT headers:

Header key	Quick constant	Description (extracted from MongoDB API doc)	Data type
CamelMongoDbStreamOperationType	**MongoDbConstants.STREAM_OPERATION_TYPE**	The type of operation that occurred. Can be any of the following values: insert, delete, replace, update, drop, rename, dropDatabase, invalidate.	String
_id	**MongoDbConstants.MONGO_ID**	A document that contains the _id of the document created or modified by the insert, replace, delete, update operations (i.e. CRUD operations). For sharded collections, also displays the full shard key for the document. The _id field is not repeated if it is already a part of the shard key.	ObjectId

45.12. TYPE CONVERSIONS

The **MongoDbBasicConverters** type converter included with the camel-mongodb component provides the following conversions:

Name	From type	To type	How?
fromMapToDocument	**Map**	**Document**	constructs a new **Document** via the **new Document(Map m)** constructor.
fromDocumentToMap	**Document**	**Map**	**Document** already implements **Map**.

Name	From type	To type	How?
fromStringToDocument	**String**	**Document**	uses **com.mongodb.Document.parse(String s)**.
fromStringToObjectId	**String**	**ObjectId**	constructs a new **ObjectId** via the **new ObjectId(s)**
fromFileToDocument	**File**	**Document**	uses **fromInputStreamToDocument** under the hood
fromInputStreamToDocument	**InputStream**	**Document**	converts the inputstream bytes to a **Document**
fromStringToList	**String**	**List<Bson>**	uses **org.bson.codecs.configuration.CodecRegistries** to convert to BsonArray then to List<Bson>.

This type converter is auto-discovered, so you don't need to configure anything manually.

45.13. SPRING BOOT AUTO-CONFIGURATION

When using mongodb with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-mongodb-starter</artifactId>
</dependency>
```

The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.component.mongodb.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.mongodb.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.mongodb.enabled	Whether to enable auto configuration of the mongodb component. This is enabled by default.		Boolean
camel.component.mongodb.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.mongodb.mongo-connection	Shared client used for connection. All endpoints generated from the component will share this connection client. The option is a com.mongodb.client.MongoClient type.		MongoClient

CHAPTER 46. NETTY

Both producer and consumer are supported

The Netty component in Camel is a socket communication component, based on the Netty project version 4.
Netty is a NIO client server framework which enables quick and easy development of networkServerInitializerFactory applications such as protocol servers and clients.
Netty greatly simplifies and streamlines network programming such as TCP and UDP socket server.

This camel component supports both producer and consumer endpoints.

The Netty component has several options and allows fine-grained control of a number of TCP/UDP communication parameters (buffer sizes, keepAlives, tcpNoDelay, etc) and facilitates both In-Only and In-Out communication on a Camel route.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-netty</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

46.1. URI FORMAT

The URI scheme for a netty component is as follows

```
netty:tcp://0.0.0.0:99999[?options]
netty:udp://remotehost:99999/[?options]
```

This component supports producer and consumer endpoints for both TCP and UDP.

46.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

46.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example, a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (**application.properties|yaml**), or directly with Java code.

46.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a *type safe* way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

46.3. COMPONENT OPTIONS

The Netty component supports 73 options, which are listed below.

Name	Description	Default	Type
configuration (common)	To use the NettyConfiguration as configuration when creating endpoints.		NettyConfiguration
disconnect (common)	Whether or not to disconnect(close) from Netty Channel right after use. Can be used for both consumer and producer.	false	boolean
keepAlive (common)	Setting to ensure socket is not closed due to inactivity.	true	boolean
reuseAddress (common)	Setting to facilitate socket multiplexing.	true	boolean
reuseChannel (common)	This option allows producers and consumers (in client mode) to reuse the same Netty Channel for the lifecycle of processing the Exchange. This is useful if you need to call a server multiple times in a Camel route and want to use the same network connection. When using this, the channel is not returned to the connection pool until the Exchange is done; or disconnected if the disconnect option is set to true. The reused Channel is stored on the Exchange as an exchange property with the key NettyConstants#NETTY_CHANNEL which allows you to obtain the channel during routing and use it as well.	false	boolean
sync (common)	Setting to set endpoint as one-way or request-response.	true	boolean

Name	Description	Default	Type
tcpNoDelay (common)	Setting to improve TCP protocol performance.	true	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
broadcast (consumer)	Setting to choose Multicast over UDP.	false	boolean
clientMode (consumer)	If the clientMode is true, netty consumer will connect the address as a TCP client.	false	boolean
reconnect (consumer)	Used only in clientMode in consumer, the consumer will attempt to reconnect on disconnection if this is enabled.	true	boolean
reconnectInterval (consumer)	Used if reconnect and clientMode is enabled. The interval in milli seconds to attempt reconnection.	10000	int
backlog (consumer (advanced))	Allows to configure a backlog for netty consumer (server). Note the backlog is just a best effort depending on the OS. Setting this option to a value such as 200, 500 or 1000, tells the TCP stack how long the accept queue can be If this option is not configured, then the backlog depends on OS setting.		int
bossCount (consumer (advanced))	When netty works on nio mode, it uses default bossCount parameter from Netty, which is 1. User can use this option to override the default bossCount from Netty.	1	int
bossGroup (consumer (advanced))	Set the BossGroup which could be used for handling the new connection of the server side across the NettyEndpoint.		EventLoopGroup
disconnectOnNoReply (consumer (advanced))	If sync is enabled then this option dictates NettyConsumer if it should disconnect where there is no reply to send back.	true	boolean

Name	Description	Default	Type
executorService (consumer (advanced))	To use the given EventExecutorGroup.		EventExecutorGroup
maximumPoolSize (consumer (advanced))	Sets a maximum thread pool size for the netty consumer ordered thread pool. The default size is 2 x cpu_core plus 1. Setting this value to eg 10 will then use 10 threads unless 2 x cpu_core plus 1 is a higher value, which then will override and be used. For example if there are 8 cores, then the consumer thread pool will be 17. This thread pool is used to route messages received from Netty by Camel. We use a separate thread pool to ensure ordering of messages and also in case some messages will block, then nettys worker threads (event loop) wont be affected.		int
nettyServerBootstrapFactory (consumer (advanced))	To use a custom NettyServerBootstrapFactory.		NettyServerBootstrapFactory
networkInterface (consumer (advanced))	When using UDP then this option can be used to specify a network interface by its name, such as eth0 to join a multicast group.		String
noReplyLogLevel (consumer (advanced))	If sync is enabled this option dictates NettyConsumer which logging level to use when logging a there is no reply to send back. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel

Name	Description	Default	Type
serverClosedChannelExceptionCaughtLogLevel (consumer (advanced))	If the server (NettyConsumer) catches an java.nio.channels.ClosedChannelException then its logged using this logging level. This is used to avoid logging the closed channel exceptions, as clients can disconnect abruptly and then cause a flood of closed exceptions in the Netty server. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	DEBUG	LoggingLevel
serverExceptionCaughtLogLevel (consumer (advanced))	If the server (NettyConsumer) catches an exception then its logged using this logging level. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
serverInitializerFactory (consumer (advanced))	To use a custom ServerInitializerFactory.		ServerInitializerFactory
usingExecutorService (consumer (advanced))	Whether to use ordered thread pool, to ensure events are processed orderly on the same channel.	true	boolean
connectTimeout (producer)	Time to wait for a socket connection to be available. Value is in milliseconds.	10000	int

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
requestTimeout (producer)	Allows to use a timeout for the Netty producer when calling a remote server. By default no timeout is in use. The value is in milli seconds, so eg 30000 is 30 seconds. The requestTimeout is using Netty's ReadTimeoutHandler to trigger the timeout.		long
clientInitializerFactory (producer (advanced))	To use a custom ClientInitializerFactory.		ClientInitializerFactory
correlationManager (producer (advanced))	To use a custom correlation manager to manage how request and reply messages are mapped when using request/reply with the netty producer. This should only be used if you have a way to map requests together with replies such as if there is correlation ids in both the request and reply messages. This can be used if you want to multiplex concurrent messages on the same channel (aka connection) in netty. When doing this you must have a way to correlate the request and reply messages so you can store the right reply on the inflight Camel Exchange before its continued routed. We recommend extending the TimeoutCorrelationManagerSupport when you build custom correlation managers. This provides support for timeout and other complexities you otherwise would need to implement as well. See also the producerPoolEnabled option for more details.		NettyCamelStateCorrelationManager
lazyChannelCreation (producer (advanced))	Channels can be lazily created to avoid exceptions, if the remote server is not up and running when the Camel producer is started.	true	boolean

Name	Description	Default	Type
producerPoolEnabled (producer (advanced))	Whether producer pool is enabled or not. Important: If you turn this off then a single shared connection is used for the producer, also if you are doing request/reply. That means there is a potential issue with interleaved responses if replies comes back out-of-order. Therefore you need to have a correlation id in both the request and reply messages so you can properly correlate the replies to the Camel callback that is responsible for continue processing the message in Camel. To do this you need to implement NettyCamelStateCorrelationManager as correlation manager and configure it via the correlationManager option. See also the correlationManager option for more details.	true	boolean
producerPoolMaxIdle (producer (advanced))	Sets the cap on the number of idle instances in the pool.	100	int
producerPoolMaxTotal (producer (advanced))	Sets the cap on the number of objects that can be allocated by the pool (checked out to clients, or idle awaiting checkout) at a given time. Use a negative value for no limit.	-1	int
producerPoolMinEvictableIdle (producer (advanced))	Sets the minimum amount of time (value in millis) an object may sit idle in the pool before it is eligible for eviction by the idle object evictor.	300000	long
producerPoolMinIdle (producer (advanced))	Sets the minimum number of instances allowed in the producer pool before the evictor thread (if active) spawns new objects.		int
udpConnectionlessSending (producer (advanced))	This option supports connection less udp sending which is a real fire and forget. A connected udp send receive the PortUnreachableException if no one is listen on the receiving port.	false	boolean
useByteBuf (producer (advanced))	If the useByteBuf is true, netty producer will turn the message body into ByteBuf before sending it out.	false	boolean
hostnameVerification (security)	To enable/disable hostname verification on SSLEngine.	false	boolean

Name	Description	Default	Type
allowSerializedHeaders (advanced)	Only used for TCP when transferExchange is true. When set to true, serializable objects in headers and properties will be added to the exchange. Otherwise Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
channelGroup (advanced)	To use a explicit ChannelGroup.		ChannelGroup
nativeTransport (advanced)	Whether to use native transport instead of NIO. Native transport takes advantage of the host operating system and is only supported on some platforms. You need to add the netty JAR for the host operating system you are using. See more details at: .	false	boolean
options (advanced)	Allows to configure additional netty options using option. as prefix. For example option.child.keepAlive=false to set the netty option child.keepAlive=false. See the Netty documentation for possible options that can be used.		Map
receiveBufferSize (advanced)	The TCP/UDP buffer sizes to be used during inbound communication. Size is bytes.	65536	int
receiveBufferSizePredictor (advanced)	Configures the buffer size predictor. See details at Jetty documentation and this mail thread.		int
sendBufferSize (advanced)	The TCP/UDP buffer sizes to be used during outbound communication. Size is bytes.	65536	int
transferExchange (advanced)	Only used for TCP. You can transfer the exchange over the wire instead of just the body. The following fields are transferred: In body, Out body, fault body, In headers, Out headers, fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean

Name	Description	Default	Type
udpByteArrayCodec (advanced)	For UDP only. If enabled the using byte array codec instead of Java serialization protocol.	false	boolean
workerCount (advanced)	When netty works on nio mode, it uses default workerCount parameter from Netty (which is cpu_core_threads x 2). User can use this option to override the default workerCount from Netty.		int
workerGroup (advanced)	To use a explicit EventLoopGroup as the boss thread pool. For example to share a thread pool with multiple consumers or producers. By default each consumer or producer has their own worker pool with 2 x cpu count core threads.		EventLoopGroup
allowDefaultCodec (codec)	The netty component installs a default codec if both, encoder/decoder is null and textline is false. Setting allowDefaultCodec to false prevents the netty component from installing a default codec as the first element in the filter chain.	true	boolean
autoAppendDelimiter (codec)	Whether or not to auto append missing end delimiter when sending using the textline codec.	true	boolean
decoderMaxLineLength (codec)	The max line length to use for the textline codec.	1024	int
decoders (codec)	A list of decoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		List
delimiter (codec)	The delimiter to use for the textline codec. Possible values are LINE and NULL. Enum values: • LINE • NULL	LINE	TextLineDelimiter
encoders (codec)	A list of encoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		List

Name	Description	Default	Type
encoding (codec)	The encoding (a charset name) to use for the textline codec. If not provided, Camel will use the JVM default Charset.		String
textline (codec)	Only used for TCP. If no codec is specified, you can use this flag to indicate a text line based codec; if not specified or the value is false, then Object Serialization is assumed over TCP - however only Strings are allowed to be serialized by default.	false	boolean
enabledProtocols (security)	Which protocols to enable when using SSL.	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	Client side certificate keystore to be used for encryption.		File
keyStoreFormat (security)	Keystore format to be used for payload encryption. Defaults to JKS if not set.		String
keyStoreResource (security)	Client side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
needClientAuth (security)	Configures whether the server needs client authentication when using SSL.	false	boolean
passphrase (security)	Password setting to use in order to encrypt/decrypt payloads sent using SSH.		String
securityProvider (security)	Security provider to be used for payload encryption. Defaults to SunX509 if not set.		String
ssl (security)	Setting to specify whether SSL encryption is applied to this endpoint.	false	boolean
sslClientCertHeaders (security)	When enabled and in SSL mode, then the Netty consumer will enrich the Camel Message with headers having information about the client certificate such as subject name, issuer name, serial number, and the valid date range.	false	boolean
sslContextParameters (security)	To configure security using SSLContextParameters.		SSLContextParameters

Name	Description	Default	Type
sslHandler (security)	Reference to a class that could be used to return an SSL Handler.		SslHandler
trustStoreFile (security)	Server side certificate keystore to be used for encryption.		File
trustStoreResource (security)	Server side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean

46.4. ENDPOINT OPTIONS

The Netty endpoint is configured using URI syntax:

```
netty:protocol://host:port
```

with the following path and query parameters:

46.4.1. Path Parameters (3 parameters)

Name	Description	Default	Type
protocol (common)	**Required** The protocol to use which can be tcp or udp. Enum values: • tcp • udp		String
host (common)	**Required** The hostname. For the consumer the hostname is localhost or 0.0.0.0. For the producer the hostname is the remote host to connect to.		String
port (common)	**Required** The host port number.		int

46.4.2. Query Parameters (71 parameters)

Name	Description	Default	Type
disconnect (common)	Whether or not to disconnect(close) from Netty Channel right after use. Can be used for both consumer and producer.	false	boolean
keepAlive (common)	Setting to ensure socket is not closed due to inactivity.	true	boolean
reuseAddress (common)	Setting to facilitate socket multiplexing.	true	boolean
reuseChannel (common)	This option allows producers and consumers (in client mode) to reuse the same Netty Channel for the lifecycle of processing the Exchange. This is useful if you need to call a server multiple times in a Camel route and want to use the same network connection. When using this, the channel is not returned to the connection pool until the Exchange is done; or disconnected if the disconnect option is set to true. The reused Channel is stored on the Exchange as an exchange property with the key NettyConstants#NETTY_CHANNEL which allows you to obtain the channel during routing and use it as well.	false	boolean
sync (common)	Setting to set endpoint as one-way or request-response.	true	boolean
tcpNoDelay (common)	Setting to improve TCP protocol performance.	true	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
broadcast (consumer)	Setting to choose Multicast over UDP.	false	boolean
clientMode (consumer)	If the clientMode is true, netty consumer will connect the address as a TCP client.	false	boolean

Name	Description	Default	Type
reconnect (consumer)	Used only in clientMode in consumer, the consumer will attempt to reconnect on disconnection if this is enabled.	true	boolean
reconnectInterval (consumer)	Used if reconnect and clientMode is enabled. The interval in milli seconds to attempt reconnection.	10000	int
backlog (consumer (advanced))	Allows to configure a backlog for netty consumer (server). Note the backlog is just a best effort depending on the OS. Setting this option to a value such as 200, 500 or 1000, tells the TCP stack how long the accept queue can be If this option is not configured, then the backlog depends on OS setting.		int
bossCount (consumer (advanced))	When netty works on nio mode, it uses default bossCount parameter from Netty, which is 1. User can use this option to override the default bossCount from Netty.	1	int
bossGroup (consumer (advanced))	Set the BossGroup which could be used for handling the new connection of the server side across the NettyEndpoint.		EventLoopGroup
disconnectOnNoReply (consumer (advanced))	If sync is enabled then this option dictates NettyConsumer if it should disconnect where there is no reply to send back.	true	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
nettyServerBootstrapFactory (consumer (advanced))	To use a custom NettyServerBootstrapFactory.		NettyServerBootstrapFactory

Name	Description	Default	Type
networkInterface (consumer (advanced))	When using UDP then this option can be used to specify a network interface by its name, such as eth0 to join a multicast group.		String
noReplyLogLevel (consumer (advanced))	If sync is enabled this option dictates NettyConsumer which logging level to use when logging a there is no reply to send back. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
serverClosedChannelExceptionCaughtLogLevel (consumer (advanced))	If the server (NettyConsumer) catches an java.nio.channels.ClosedChannelException then its logged using this logging level. This is used to avoid logging the closed channel exceptions, as clients can disconnect abruptly and then cause a flood of closed exceptions in the Netty server. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	DEBUG	LoggingLevel

Name	Description	Default	Type
serverExceptionCaughtLogLevel (consumer (advanced))	If the server (NettyConsumer) catches an exception then its logged using this logging level. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	WARN	LoggingLevel
serverInitializerFactory (consumer (advanced))	To use a custom ServerInitializerFactory.		ServerInitializerFactory
usingExecutorService (consumer (advanced))	Whether to use ordered thread pool, to ensure events are processed orderly on the same channel.	true	boolean
connectTimeout (producer)	Time to wait for a socket connection to be available. Value is in milliseconds.	10000	int
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
requestTimeout (producer)	Allows to use a timeout for the Netty producer when calling a remote server. By default no timeout is in use. The value is in milli seconds, so eg 30000 is 30 seconds. The requestTimeout is using Netty's ReadTimeoutHandler to trigger the timeout.		long
clientInitializerFactory (producer (advanced))	To use a custom ClientInitializerFactory.		ClientInitializerFactory

Name	Description	Default	Type
correlationManager (producer (advanced))	To use a custom correlation manager to manage how request and reply messages are mapped when using request/reply with the netty producer. This should only be used if you have a way to map requests together with replies such as if there is correlation ids in both the request and reply messages. This can be used if you want to multiplex concurrent messages on the same channel (aka connection) in netty. When doing this you must have a way to correlate the request and reply messages so you can store the right reply on the inflight Camel Exchange before its continued routed. We recommend extending the TimeoutCorrelationManagerSupport when you build custom correlation managers. This provides support for timeout and other complexities you otherwise would need to implement as well. See also the producerPoolEnabled option for more details.		NettyCamelStateCorrelationManager
lazyChannelCreation (producer (advanced))	Channels can be lazily created to avoid exceptions, if the remote server is not up and running when the Camel producer is started.	true	boolean
producerPoolEnabled (producer (advanced))	Whether producer pool is enabled or not. Important: If you turn this off then a single shared connection is used for the producer, also if you are doing request/reply. That means there is a potential issue with interleaved responses if replies comes back out-of-order. Therefore you need to have a correlation id in both the request and reply messages so you can properly correlate the replies to the Camel callback that is responsible for continue processing the message in Camel. To do this you need to implement NettyCamelStateCorrelationManager as correlation manager and configure it via the correlationManager option. See also the correlationManager option for more details.	true	boolean
producerPoolMaxIdle (producer (advanced))	Sets the cap on the number of idle instances in the pool.	100	int
producerPoolMaxTotal (producer (advanced))	Sets the cap on the number of objects that can be allocated by the pool (checked out to clients, or idle awaiting checkout) at a given time. Use a negative value for no limit.	-1	int

Name	Description	Default	Type
producerPoolMinEvictableIdle (producer (advanced))	Sets the minimum amount of time (value in millis) an object may sit idle in the pool before it is eligible for eviction by the idle object evictor.	300000	long
producerPoolMinIdle (producer (advanced))	Sets the minimum number of instances allowed in the producer pool before the evictor thread (if active) spawns new objects.		int
udpConnectionlessSending (producer (advanced))	This option supports connection less udp sending which is a real fire and forget. A connected udp send receive the PortUnreachableException if no one is listen on the receiving port.	false	boolean
useByteBuf (producer (advanced))	If the useByteBuf is true, netty producer will turn the message body into ByteBuf before sending it out.	false	boolean
hostnameVerification (security)	To enable/disable hostname verification on SSLEngine.	false	boolean
allowSerializedHeaders (advanced)	Only used for TCP when transferExchange is true. When set to true, serializable objects in headers and properties will be added to the exchange. Otherwise Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
channelGroup (advanced)	To use a explicit ChannelGroup.		ChannelGroup
nativeTransport (advanced)	Whether to use native transport instead of NIO. Native transport takes advantage of the host operating system and is only supported on some platforms. You need to add the netty JAR for the host operating system you are using. See more details at: .	false	boolean
options (advanced)	Allows to configure additional netty options using option. as prefix. For example option.child.keepAlive=false to set the netty option child.keepAlive=false. See the Netty documentation for possible options that can be used.		Map
receiveBufferSize (advanced)	The TCP/UDP buffer sizes to be used during inbound communication. Size is bytes.	65536	int

Name	Description	Default	Type
receiveBufferSizePredictor (advanced)	Configures the buffer size predictor. See details at Jetty documentation and this mail thread.		int
sendBufferSize (advanced)	The TCP/UDP buffer sizes to be used during outbound communication. Size is bytes.	65536	int
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
transferExchange (advanced)	Only used for TCP. You can transfer the exchange over the wire instead of just the body. The following fields are transferred: In body, Out body, fault body, In headers, Out headers, fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	boolean
udpByteArrayCodec (advanced)	For UDP only. If enabled the using byte array codec instead of Java serialization protocol.	false	boolean
workerCount (advanced)	When netty works on nio mode, it uses default workerCount parameter from Netty (which is cpu_core_threads x 2). User can use this option to override the default workerCount from Netty.		int
workerGroup (advanced)	To use a explicit EventLoopGroup as the boss thread pool. For example to share a thread pool with multiple consumers or producers. By default each consumer or producer has their own worker pool with 2 x cpu count core threads.		EventLoopGroup
allowDefaultCodec (codec)	The netty component installs a default codec if both, encoder/decoder is null and textline is false. Setting allowDefaultCodec to false prevents the netty component from installing a default codec as the first element in the filter chain.	true	boolean
autoAppendDelimiter (codec)	Whether or not to auto append missing end delimiter when sending using the textline codec.	true	boolean
decoderMaxLineLength (codec)	The max line length to use for the textline codec.	1024	int

Name	Description	Default	Type
decoders (codec)	A list of decoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		List
delimiter (codec)	The delimiter to use for the textline codec. Possible values are LINE and NULL. Enum values: • LINE • NULL	LINE	TextLineDelimiter
encoders (codec)	A list of encoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		List
encoding (codec)	The encoding (a charset name) to use for the textline codec. If not provided, Camel will use the JVM default Charset.		String
textline (codec)	Only used for TCP. If no codec is specified, you can use this flag to indicate a text line based codec; if not specified or the value is false, then Object Serialization is assumed over TCP - however only Strings are allowed to be serialized by default.	false	boolean
enabledProtocols (security)	Which protocols to enable when using SSL.	TLSv1, TLSv1.1, TLSv1.2	String
keyStoreFile (security)	Client side certificate keystore to be used for encryption.		File
keyStoreFormat (security)	Keystore format to be used for payload encryption. Defaults to JKS if not set.		String
keyStoreResource (security)	Client side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
needClientAuth (security)	Configures whether the server needs client authentication when using SSL.	false	boolean

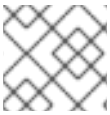
Name	Description	Default	Type
passphrase (security)	Password setting to use in order to encrypt/decrypt payloads sent using SSH.		String
securityProvider (security)	Security provider to be used for payload encryption. Defaults to SunX509 if not set.		String
ssl (security)	Setting to specify whether SSL encryption is applied to this endpoint.	false	boolean
sslClientCertHeaders (security)	When enabled and in SSL mode, then the Netty consumer will enrich the Camel Message with headers having information about the client certificate such as subject name, issuer name, serial number, and the valid date range.	false	boolean
sslContextParameters (security)	To configure security using SSLContextParameters.		SSLContextParameters
sslHandler (security)	Reference to a class that could be used to return an SSL Handler.		SslHandler
trustStoreFile (security)	Server side certificate keystore to be used for encryption.		File
trustStoreResource (security)	Server side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String

46.5. REGISTRY BASED OPTIONS

Codec Handlers and SSL Keystores can be enlisted in the Registry, such as in the Spring XML file. The values that could be passed in, are the following:

Name	Description
passphrase	password setting to use in order to encrypt/decrypt payloads sent using SSH
keyStoreFormat	keystore format to be used for payload encryption. Defaults to "JKS" if not set
securityProvider	Security provider to be used for payload encryption. Defaults to "SunX509" if not set.
keyStoreFile	**deprecated:** Client side certificate keystore to be used for encryption

Name	Description
trustStoreFile	**deprecated:** Server side certificate keystore to be used for encryption
keyStoreResource	Client side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with **"classpath:"**, **"file:"**, or **"http:"** to load the resource from different systems.
trustStoreResource	Server side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with **"classpath:"**, **"file:"**, or **"http:"** to load the resource from different systems.
sslHandler	Reference to a class that could be used to return an SSL Handler
encoder	A custom **ChannelHandler** class that can be used to perform special marshalling of outbound payloads. Must override io.netty.channel.ChannelInboundHandlerAdapter.
encoders	A list of encoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.
decoder	A custom **ChannelHandler** class that can be used to perform special marshalling of inbound payloads. Must override io.netty.channel.ChannelOutboundHandlerAdapter.
decoders	A list of decoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.



NOTE

Read below about using non shareable encoders/decoders.

46.5.1. Using non shareable encoders or decoders

If your encoders or decoders are not shareable (e.g. they don't have the @Shareable class annotation), then your encoder/decoder must implement the **org.apache.camel.component.netty.ChannelHandlerFactory** interface, and return a new instance in the **newChannelHandler** method. This is to ensure the encoder/decoder can safely be used. If this is not the case, then the Netty component will log a WARN when an endpoint is created.

The Netty component offers a **org.apache.camel.component.netty.ChannelHandlerFactories** factory class, that has a number of commonly used methods.

46.6. SENDING MESSAGES TO/FROM A NETTY ENDPOINT

46.6.1. Netty Producer

In Producer mode, the component provides the ability to send payloads to a socket endpoint using either TCP or UDP protocols (with optional SSL support).

The producer mode supports both one-way and request-response based operations.

46.6.2. Netty Consumer

In Consumer mode, the component provides the ability to:

- listen on a specified socket using either TCP or UDP protocols (with optional SSL support),
- receive requests on the socket using text/xml, binary and serialized object based payloads and
- send them along on a route as message exchanges.

The consumer mode supports both one-way and request-response based operations.

46.7. EXAMPLES

46.7.1. A UDP Netty endpoint using Request-Reply and serialized object payload

Note that Object serialization is not allowed by default, and so a decoder must be configured.

```
@BindToRegistry("decoder")
public ChannelHandler getDecoder() throws Exception {
    return new DefaultChannelHandlerFactory() {
        @Override
        public ChannelHandler newChannelHandler() {
            return new DatagramPacketObjectDecoder(ClassResolvers.weakCachingResolver(null));
        }
    };
}

RouteBuilder builder = new RouteBuilder() {
  public void configure() {
    from("netty:udp://0.0.0.0:5155?sync=true&decoders=#decoder")
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
          Poetry poetry = (Poetry) exchange.getIn().getBody();
          // Process poetry in some way
          exchange.getOut().setBody("Message received);
        }
      }
    }
};
```

46.7.2. A TCP based Netty consumer endpoint using One-way communication

```
RouteBuilder builder = new RouteBuilder() {
  public void configure() {
      from("netty:tcp://0.0.0.0:5150")
          .to("mock:result");
  }
};
```

46.7.3. An SSL/TCP based Netty consumer endpoint using Request-Reply communication

Using the JSSE Configuration Utility

The Netty component supports SSL/TLS configuration through the Camel JSSE Configuration Utility. This utility greatly decreases the amount of component specific code you need to write and is configurable at the endpoint and component levels. The following examples demonstrate how to use the utility with the Netty component.

Programmatic configuration of the component

```
KeyStoreParameters ksp = new KeyStoreParameters();
ksp.setResource("/users/home/server/keystore.jks");
ksp.setPassword("keystorePassword");

KeyManagersParameters kmp = new KeyManagersParameters();
kmp.setKeyStore(ksp);
kmp.setKeyPassword("keyPassword");

SSLContextParameters scp = new SSLContextParameters();
scp.setKeyManagers(kmp);

NettyComponent nettyComponent = getContext().getComponent("netty", NettyComponent.class);
nettyComponent.setSslContextParameters(scp);
```

Spring DSL based configuration of endpoint

```
...
  <camel:sslContextParameters
     id="sslContextParameters">
    <camel:keyManagers
       keyPassword="keyPassword">
      <camel:keyStore
         resource="/users/home/server/keystore.jks"
         password="keystorePassword"/>
    </camel:keyManagers>
  </camel:sslContextParameters>...
...
  <to uri="netty:tcp://0.0.0.0:5150?
sync=true&ssl=true&sslContextParameters=#sslContextParameters"/>
...
```

Using Basic SSL/TLS configuration on the Jetty Component

```
Registry registry = context.getRegistry();
registry.bind("password", "changeit");
registry.bind("ksf", new File("src/test/resources/keystore.jks"));
registry.bind("tsf", new File("src/test/resources/keystore.jks"));

context.addRoutes(new RouteBuilder() {
  public void configure() {
     String netty_ssl_endpoint =
       "netty:tcp://0.0.0.0:5150?sync=true&ssl=true&passphrase=#password"
       + "&keyStoreFile=#ksf&trustStoreFile=#tsf";
```

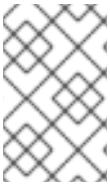
```
        String return_string =
          "When You Go Home, Tell Them Of Us And Say,"
          + "For Your Tomorrow, We Gave Our Today.";

        from(netty_ssl_endpoint)
         .process(new Processor() {
           public void process(Exchange exchange) throws Exception {
             exchange.getOut().setBody(return_string);
           }
         }
      }
    });
```

Getting access to SSLSession and the client certificate

You can get access to the **javax.net.ssl.SSLSession** if you eg need to get details about the client certificate. When **ssl=true** then the Netty component will store the **SSLSession** as a header on the Camel Message as shown below:

```
SSLSession session = exchange.getIn().getHeader(NettyConstants.NETTY_SSL_SESSION,
SSLSession.class);
// get the first certificate which is client certificate
javax.security.cert.X509Certificate cert = session.getPeerCertificateChain()[0];
Principal principal = cert.getSubjectDN();
```

Remember to set **needClientAuth=true** to authenticate the client, otherwise **SSLSession** cannot access information about the client certificate, and you may get an exception **javax.net.ssl.SSLPeerUnverifiedException: peer not authenticated**. You may also get this exception if the client certificate is expired or not valid etc.



NOTE

The option **sslClientCertHeaders** can be set to **true** which then enriches the Camel Message with headers having details about the client certificate. For example the subject name is readily available in the header **CamelNettySSLClientCertSubjectName**.

46.7.4. Using Multiple Codecs

In certain cases it may be necessary to add chains of encoders and decoders to the netty pipeline. To add multpile codecs to a camel netty endpoint the 'encoders' and 'decoders' uri parameters should be used. Like the 'encoder' and 'decoder' parameters they are used to supply references (lists of ChannelUpstreamHandlers and ChannelDownstreamHandlers) that should be added to the pipeline. Note that if encoders is specified then the encoder param will be ignored, similarly for decoders and the decoder param.

NOTE

Read further above about using non shareable encoders/decoders.

The lists of codecs need to be added to the Camel's registry so they can be resolved when the endpoint is created.

```
ChannelHandlerFactory lengthDecoder =
ChannelHandlerFactories.newLengthFieldBasedFrameDecoder(1048576, 0, 4, 0, 4);
```

```
StringDecoder stringDecoder = new StringDecoder();
registry.bind("length-decoder", lengthDecoder);
registry.bind("string-decoder", stringDecoder);

LengthFieldPrepender lengthEncoder = new LengthFieldPrepender(4);
StringEncoder stringEncoder = new StringEncoder();
registry.bind("length-encoder", lengthEncoder);
registry.bind("string-encoder", stringEncoder);

List<ChannelHandler> decoders = new ArrayList<ChannelHandler>();
decoders.add(lengthDecoder);
decoders.add(stringDecoder);

List<ChannelHandler> encoders = new ArrayList<ChannelHandler>();
encoders.add(lengthEncoder);
encoders.add(stringEncoder);

registry.bind("encoders", encoders);
registry.bind("decoders", decoders);
```

Spring's native collections support can be used to specify the codec lists in an application context

```
<util:list id="decoders" list-class="java.util.LinkedList">
        <bean class="org.apache.camel.component.netty.ChannelHandlerFactories" factory-
method="newLengthFieldBasedFrameDecoder">
            <constructor-arg value="1048576"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
            <constructor-arg value="0"/>
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringDecoder"/>
    </util:list>

    <util:list id="encoders" list-class="java.util.LinkedList">
        <bean class="io.netty.handler.codec.LengthFieldPrepender">
            <constructor-arg value="4"/>
        </bean>
        <bean class="io.netty.handler.codec.string.StringEncoder"/>
    </util:list>

    <bean id="length-encoder" class="io.netty.handler.codec.LengthFieldPrepender">
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-encoder" class="io.netty.handler.codec.string.StringEncoder"/>

    <bean id="length-decoder" class="org.apache.camel.component.netty.ChannelHandlerFactories"
factory-method="newLengthFieldBasedFrameDecoder">
        <constructor-arg value="1048576"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
        <constructor-arg value="0"/>
        <constructor-arg value="4"/>
    </bean>
    <bean id="string-decoder" class="io.netty.handler.codec.string.StringDecoder"/>
```

-

The bean names can then be used in netty endpoint definitions either as a comma separated list or contained in a List e.g.

```
 from("direct:multiple-codec").to("netty:tcp://0.0.0.0:{{port}}?encoders=#encoders&sync=false");

 from("netty:tcp://0.0.0.0:{{port}}?decoders=#length-decoder,#string-
decoder&sync=false").to("mock:multiple-codec");
```

or via XML.

```
<camelContext id="multiple-netty-codecs-context" xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:multiple-codec"/>
        <to uri="netty:tcp://0.0.0.0:5150?encoders=#encoders&amp;sync=false"/>
    </route>
    <route>
        <from uri="netty:tcp://0.0.0.0:5150?decoders=#length-decoder,#string-
decoder&amp;sync=false"/>
        <to uri="mock:multiple-codec"/>
    </route>
</camelContext>
```

46.8. CLOSING CHANNEL WHEN COMPLETE

When acting as a server you sometimes want to close the channel when, for example, a client conversion is finished.
You can do this by simply setting the endpoint option **disconnect=true**.

However you can also instruct Camel on a per message basis as follows.
To instruct Camel to close the channel, you should add a header with the key **CamelNettyCloseChannelWhenComplete** set to a boolean **true** value.
For instance, the example below will close the channel after it has written the bye message back to the client:

```
from("netty:tcp://0.0.0.0:8080").process(new Processor() {
    public void process(Exchange exchange) throws Exception {
        String body = exchange.getIn().getBody(String.class);
        exchange.getOut().setBody("Bye " + body);
        // some condition which determines if we should close
        if (close) {

exchange.getOut().setHeader(NettyConstants.NETTY_CLOSE_CHANNEL_WHEN_COMPLETE,
true);
        }
    }
});
```

Adding custom channel pipeline factories to gain complete control over a created pipeline.

46.9. CUSTOM PIPELINE

Custom channel pipelines provide complete control to the user over the handler/interceptor chain by inserting custom handler(s), encoder(s) & decoder(s) without having to specify them in the Netty Endpoint URL in a very simple way.

In order to add a custom pipeline, a custom channel pipeline factory must be created and registered with the context via the context registry (Registry, or the camel-spring ApplicationContextRegistry etc).

A custom pipeline factory must be constructed as follows

- A Producer linked channel pipeline factory must extend the abstract class **ClientPipelineFactory**.
- A Consumer linked channel pipeline factory must extend the abstract class **ServerInitializerFactory**.
- The classes should override the initChannel() method in order to insert custom handler(s), encoder(s) and decoder(s). Not overriding the **initChannel()** method creates a pipeline with no handlers, encoders or decoders wired to the pipeline.

The example below shows how ServerInitializerFactory factory may be created

46.9.1. Using custom pipeline factory

```
public class SampleServerInitializerFactory extends ServerInitializerFactory {
    private int maxLineSize = 1024;

    protected void initChannel(Channel ch) throws Exception {
        ChannelPipeline channelPipeline = ch.pipeline();

        channelPipeline.addLast("encoder-SD", new StringEncoder(CharsetUtil.UTF_8));
        channelPipeline.addLast("decoder-DELIM", new DelimiterBasedFrameDecoder(maxLineSize,
true, Delimiters.lineDelimiter()));
        channelPipeline.addLast("decoder-SD", new StringDecoder(CharsetUtil.UTF_8));
        // here we add the default Camel ServerChannelHandler for the consumer, to allow Camel to
route the message etc.
        channelPipeline.addLast("handler", new ServerChannelHandler(consumer));
    }
}
```

The custom channel pipeline factory can then be added to the registry and instantiated/utilized on a camel route in the following way

```
Registry registry = camelContext.getRegistry();
ServerInitializerFactory factory = new TestServerInitializerFactory();
registry.bind("spf", factory);
context.addRoutes(new RouteBuilder() {
  public void configure() {
     String netty_ssl_endpoint =
       "netty:tcp://0.0.0.0:5150?serverInitializerFactory=#spf"
     String return_string =
       "When You Go Home, Tell Them Of Us And Say,"
       + "For Your Tomorrow, We Gave Our Today.";

     from(netty_ssl_endpoint)
      .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
```

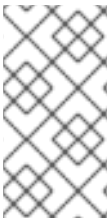
```
            exchange.getOut().setBody(return_string);
          }
        }
      }
    });
```

46.10. REUSING NETTY BOSS AND WORKER THREAD POOLS

Netty has two kind of thread pools: boss and worker. By default each Netty consumer and producer has their private thread pools. If you want to reuse these thread pools among multiple consumers or producers then the thread pools must be created and enlisted in the Registry.

For example using Spring XML we can create a shared worker thread pool using the **NettyWorkerPoolBuilder** with 2 worker threads as shown below:

```
<!-- use the worker pool builder to help create the shared thread pool -->
<bean id="poolBuilder" class="org.apache.camel.component.netty.NettyWorkerPoolBuilder">
  <property name="workerCount" value="2"/>
</bean>

<!-- the shared worker thread pool -->
<bean id="sharedPool" class="org.jboss.netty.channel.socket.nio.WorkerPool"
      factory-bean="poolBuilder" factory-method="build" destroy-method="shutdown">
</bean>
```



NOTE

For boss thread pool there is a **org.apache.camel.component.netty.NettyServerBossPoolBuilder** builder for Netty consumers, and a **org.apache.camel.component.netty.NettyClientBossPoolBuilder** for the Netty producers.

Then in the Camel routes we can refer to this worker pools by configuring the **workerPool** option in the URI as shown below:

```
<route>
  <from uri="netty:tcp://0.0.0.0:5021?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

And if we have another route we can refer to the shared worker pool:

```
<route>
  <from uri="netty:tcp://0.0.0.0:5022?
textline=true&amp;sync=true&amp;workerPool=#sharedPool&amp;usingExecutorService=false"/>
  <to uri="log:result"/>
  ...
</route>
```

and so forth.

46.11. MULTIPLEXING CONCURRENT MESSAGES OVER A SINGLE CONNECTION WITH REQUEST/REPLY

When using Netty for request/reply messaging via the netty producer then by default each message is sent via a non-shared connection (pooled). This ensures that replies are automatic being able to map to the correct request thread for further routing in Camel. In other words correlation between request/reply messages happens out-of-the-box because the replies comes back on the same connection that was used for sending the request; and this connection is not shared with others. When the response comes back, the connection is returned back to the connection pool, where it can be reused by others.

However if you want to multiplex concurrent request/responses on a single shared connection, then you need to turn off the connection pooling by setting **producerPoolEnabled=false**. Now this means there is a potential issue with interleaved responses if replies comes back out-of-order. Therefore you need to have a correlation id in both the request and reply messages so you can properly correlate the replies to the Camel callback that is responsible for continue processing the message in Camel. To do this you need to implement **NettyCamelStateCorrelationManager** as correlation manager and configure it via the **correlationManager=#myManager** option.

NOTE

We recommend extending the **TimeoutCorrelationManagerSupport** when you build custom correlation managers. This provides support for timeout and other complexities you otherwise would need to implement as well.

You can find an example with the Apache Camel source code in the examples directory under the **camel-example-netty-custom-correlation** directory.

46.12. SPRING BOOT AUTO-CONFIGURATION

When using netty with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-netty-starter</artifactId>
</dependency>
```

The component supports 74 options, which are listed below.

Name	Description	Default	Type
camel.component.netty.allow-default-codec	The netty component installs a default codec if both, encoder/decoder is null and textline is false. Setting allowDefaultCodec to false prevents the netty component from installing a default codec as the first element in the filter chain.	true	Boolean

Name	Description	Default	Type
camel.component.netty.allow-serialized-headers	Only used for TCP when transferExchange is true. When set to true, serializable objects in headers and properties will be added to the exchange. Otherwise Camel will exclude any non-serializable objects and log it at WARN level.	false	Boolean
camel.component.netty.auto-append-delimiter	Whether or not to auto append missing end delimiter when sending using the textline codec.	true	Boolean
camel.component.netty.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.netty.backlog	Allows to configure a backlog for netty consumer (server). Note the backlog is just a best effort depending on the OS. Setting this option to a value such as 200, 500 or 1000, tells the TCP stack how long the accept queue can be If this option is not configured, then the backlog depends on OS setting.		Integer
camel.component.netty.boss-count	When netty works on nio mode, it uses default bossCount parameter from Netty, which is 1. User can use this option to override the default bossCount from Netty.	1	Integer
camel.component.netty.boss-group	Set the BossGroup which could be used for handling the new connection of the server side across the NettyEndpoint. The option is a io.netty.channel.EventLoopGroup type.		EventLoopGroup
camel.component.netty.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.netty.broadcast	Setting to choose Multicast over UDP.	false	Boolean

Name	Description	Default	Type
camel.component.netty.channel-group	To use a explicit ChannelGroup. The option is a io.netty.channel.group.ChannelGroup type.		ChannelGroup
camel.component.netty.client-initializer-factory	To use a custom ClientInitializerFactory. The option is a org.apache.camel.component.netty.ClientInitializerFactory type.		ClientInitializerFactory
camel.component.netty.client-mode	If the clientMode is true, netty consumer will connect the address as a TCP client.	false	Boolean
camel.component.netty.configuration	To use the NettyConfiguration as configuration when creating endpoints. The option is a org.apache.camel.component.netty.NettyConfiguration type.		NettyConfiguration
camel.component.netty.connect-timeout	Time to wait for a socket connection to be available. Value is in milliseconds.	10000	Integer
camel.component.netty.correlation-manager	To use a custom correlation manager to manage how request and reply messages are mapped when using request/reply with the netty producer. This should only be used if you have a way to map requests together with replies such as if there is correlation ids in both the request and reply messages. This can be used if you want to multiplex concurrent messages on the same channel (aka connection) in netty. When doing this you must have a way to correlate the request and reply messages so you can store the right reply on the inflight Camel Exchange before its continued routed. We recommend extending the TimeoutCorrelationManagerSupport when you build custom correlation managers. This provides support for timeout and other complexities you otherwise would need to implement as well. See also the producerPoolEnabled option for more details. The option is a org.apache.camel.component.netty.NettyCamelStateCorrelationManager type.		NettyCamelStateCorrelationManager
camel.component.netty.decoder-max-line-length	The max line length to use for the textline codec.	1024	Integer

Name	Description	Default	Type
camel.component.netty.decoders	A list of decoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		String
camel.component.netty.delimiter	The delimiter to use for the textline codec. Possible values are LINE and NULL.		TextLineDelimiter
camel.component.netty.disconnect	Whether or not to disconnect(close) from Netty Channel right after use. Can be used for both consumer and producer.	false	Boolean
camel.component.netty.disconnect-on-no-reply	If sync is enabled then this option dictates NettyConsumer if it should disconnect where there is no reply to send back.	true	Boolean
camel.component.netty.enabled	Whether to enable auto configuration of the netty component. This is enabled by default.		Boolean
camel.component.netty.enabled-protocols	Which protocols to enable when using SSL.	TLSv1, TLSv1.1, TLSv1.2	String
camel.component.netty.encoders	A list of encoders to be used. You can use a String which have values separated by comma, and have the values be looked up in the Registry. Just remember to prefix the value with # so Camel knows it should lookup.		String
camel.component.netty.encoding	The encoding (a charset name) to use for the textline codec. If not provided, Camel will use the JVM default Charset.		String
camel.component.netty.executor-service	To use the given EventExecutorGroup. The option is a io.netty.util.concurrent.EventExecutorGroup type.		EventExecutorGroup
camel.component.netty.hostname-verification	To enable/disable hostname verification on SSLEngine.	false	Boolean
camel.component.netty.keep-alive	Setting to ensure socket is not closed due to inactivity.	true	Boolean
camel.component.netty.key-store-file	Client side certificate keystore to be used for encryption.		File

Name	Description	Default	Type
camel.component.netty.key-store-format	Keystore format to be used for payload encryption. Defaults to JKS if not set.		String
camel.component.netty.key-store-resource	Client side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
camel.component.netty.lazy-channel-creation	Channels can be lazily created to avoid exceptions, if the remote server is not up and running when the Camel producer is started.	true	Boolean
camel.component.netty.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.netty.maximum-pool-size	Sets a maximum thread pool size for the netty consumer ordered thread pool. The default size is 2 x cpu_core plus 1. Setting this value to eg 10 will then use 10 threads unless 2 x cpu_core plus 1 is a higher value, which then will override and be used. For example if there are 8 cores, then the consumer thread pool will be 17. This thread pool is used to route messages received from Netty by Camel. We use a separate thread pool to ensure ordering of messages and also in case some messages will block, then nettys worker threads (event loop) wont be affected.		Integer
camel.component.netty.native-transport	Whether to use native transport instead of NIO. Native transport takes advantage of the host operating system and is only supported on some platforms. You need to add the netty JAR for the host operating system you are using. See more details at: .	false	Boolean
camel.component.netty.need-client-auth	Configures whether the server needs client authentication when using SSL.	false	Boolean

Name	Description	Default	Type
camel.component.netty.netty-server-bootstrap-factory	To use a custom NettyServerBootstrapFactory. The option is a org.apache.camel.component.netty.NettyServerBootstrapFactory type.		NettyServerBootstrapFactory
camel.component.netty.network-interface	When using UDP then this option can be used to specify a network interface by its name, such as eth0 to join a multicast group.		String
camel.component.netty.no-reply-log-level	If sync is enabled this option dictates NettyConsumer which logging level to use when logging a there is no reply to send back.		LoggingLevel
camel.component.netty.options	Allows to configure additional netty options using option. as prefix. For example option.child.keepAlive=false to set the netty option child.keepAlive=false. See the Netty documentation for possible options that can be used.		Map
camel.component.netty.passphrase	Password setting to use in order to encrypt/decrypt payloads sent using SSH.		String
camel.component.netty.producer-pool-enabled	Whether producer pool is enabled or not. Important: If you turn this off then a single shared connection is used for the producer, also if you are doing request/reply. That means there is a potential issue with interleaved responses if replies comes back out-of-order. Therefore you need to have a correlation id in both the request and reply messages so you can properly correlate the replies to the Camel callback that is responsible for continue processing the message in Camel. To do this you need to implement NettyCamelStateCorrelationManager as correlation manager and configure it via the correlationManager option. See also the correlationManager option for more details.	true	Boolean
camel.component.netty.producer-pool-max-idle	Sets the cap on the number of idle instances in the pool.	100	Integer
camel.component.netty.producer-pool-max-total	Sets the cap on the number of objects that can be allocated by the pool (checked out to clients, or idle awaiting checkout) at a given time. Use a negative value for no limit.	-1	Integer

Name	Description	Default	Type
camel.component.netty.producer-pool-min-evictable-idle	Sets the minimum amount of time (value in millis) an object may sit idle in the pool before it is eligible for eviction by the idle object evictor.	300000	Long
camel.component.netty.producer-pool-min-idle	Sets the minimum number of instances allowed in the producer pool before the evictor thread (if active) spawns new objects.		Integer
camel.component.netty.receive-buffer-size	The TCP/UDP buffer sizes to be used during inbound communication. Size is bytes.	65536	Integer
camel.component.netty.receive-buffer-size-predictor	Configures the buffer size predictor. See details at Jetty documentation and this mail thread.		Integer
camel.component.netty.reconnect	Used only in clientMode in consumer, the consumer will attempt to reconnect on disconnection if this is enabled.	true	Boolean
camel.component.netty.reconnect-interval	Used if reconnect and clientMode is enabled. The interval in milli seconds to attempt reconnection.	10000	Integer
camel.component.netty.request-timeout	Allows to use a timeout for the Netty producer when calling a remote server. By default no timeout is in use. The value is in milli seconds, so eg 30000 is 30 seconds. The requestTimeout is using Netty's ReadTimeoutHandler to trigger the timeout.		Long
camel.component.netty.reuse-address	Setting to facilitate socket multiplexing.	true	Boolean
camel.component.netty.reuse-channel	This option allows producers and consumers (in client mode) to reuse the same Netty Channel for the lifecycle of processing the Exchange. This is useful if you need to call a server multiple times in a Camel route and want to use the same network connection. When using this, the channel is not returned to the connection pool until the Exchange is done; or disconnected if the disconnect option is set to true. The reused Channel is stored on the Exchange as an exchange property with the key NettyConstants#NETTY_CHANNEL which allows you to obtain the channel during routing and use it as well.	false	Boolean

Name	Description	Default	Type
camel.component.netty.security-provider	Security provider to be used for payload encryption. Defaults to SunX509 if not set.		String
camel.component.netty.send-buffer-size	The TCP/UDP buffer sizes to be used during outbound communication. Size is bytes.	65536	Integer
camel.component.netty.server-closed-channel-exception-caught-log-level	If the server (NettyConsumer) catches an java.nio.channels.ClosedChannelException then its logged using this logging level. This is used to avoid logging the closed channel exceptions, as clients can disconnect abruptly and then cause a flood of closed exceptions in the Netty server.		LoggingLevel
camel.component.netty.server-exception-caught-log-level	If the server (NettyConsumer) catches an exception then its logged using this logging level.		LoggingLevel
camel.component.netty.server-initializer-factory	To use a custom ServerInitializerFactory. The option is a org.apache.camel.component.netty.ServerInitializerFactory type.		ServerInitializerFactory
camel.component.netty.ssl	Setting to specify whether SSL encryption is applied to this endpoint.	false	Boolean
camel.component.netty.ssl-client-cert-headers	When enabled and in SSL mode, then the Netty consumer will enrich the Camel Message with headers having information about the client certificate such as subject name, issuer name, serial number, and the valid date range.	false	Boolean
camel.component.netty.ssl-context-parameters	To configure security using SSLContextParameters. The option is a org.apache.camel.support.jsse.SSLContextParameters type.		SSLContextParameters
camel.component.netty.ssl-handler	Reference to a class that could be used to return an SSL Handler. The option is a io.netty.handler.ssl.SslHandler type.		SslHandler
camel.component.netty.sync	Setting to set endpoint as one-way or request-response.	true	Boolean

Name	Description	Default	Type
camel.component.netty.tcp-no-delay	Setting to improve TCP protocol performance.	true	Boolean
camel.component.netty.textline	Only used for TCP. If no codec is specified, you can use this flag to indicate a text line based codec; if not specified or the value is false, then Object Serialization is assumed over TCP - however only Strings are allowed to be serialized by default.	false	Boolean
camel.component.netty.transfer-exchange	Only used for TCP. You can transfer the exchange over the wire instead of just the body. The following fields are transferred: In body, Out body, fault body, In headers, Out headers, fault headers, exchange properties, exchange exception. This requires that the objects are serializable. Camel will exclude any non-serializable objects and log it at WARN level.	false	Boolean
camel.component.netty.trust-store-file	Server side certificate keystore to be used for encryption.		File
camel.component.netty.trust-store-resource	Server side certificate keystore to be used for encryption. Is loaded by default from classpath, but you can prefix with classpath:, file:, or http: to load the resource from different systems.		String
camel.component.netty.udp-byte-array-codec	For UDP only. If enabled the using byte array codec instead of Java serialization protocol.	false	Boolean
camel.component.netty.udp-connectionless-sending	This option supports connection less udp sending which is a real fire and forget. A connected udp send receive the PortUnreachableException if no one is listen on the receiving port.	false	Boolean
camel.component.netty.use-byte-buf	If the useByteBuf is true, netty producer will turn the message body into ByteBuf before sending it out.	false	Boolean
camel.component.netty.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean

Name	Description	Default	Type
camel.component.netty.using-executor-service	Whether to use ordered thread pool, to ensure events are processed orderly on the same channel.	true	Boolean
camel.component.netty.worker-count	When netty works on nio mode, it uses default workerCount parameter from Netty (which is cpu_core_threads x 2). User can use this option to override the default workerCount from Netty.		Integer
camel.component.netty.worker-group	To use a explicit EventLoopGroup as the boss thread pool. For example to share a thread pool with multiple consumers or producers. By default each consumer or producer has their own worker pool with 2 x cpu count core threads. The option is a io.netty.channel.EventLoopGroup type.		EventLoopGroup

CHAPTER 47. PAHO

Both producer and consumer are supported

Paho component provides connector for the MQTT messaging protocol using the Eclipse Paho library. Paho is one of the most popular MQTT libraries, so if you would like to integrate it with your Java project - Camel Paho connector is a way to go.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paho</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

47.1. URI FORMAT

```
paho:topic[?options]
```

Where **topic** is the name of the topic.

47.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

47.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

47.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings.

In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

47.3. COMPONENT OPTIONS

The Paho component supports 31 options, which are listed below.

Name	Description	Default	Type
automaticReconnect (common)	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	boolean
brokerUrl (common)	The URL of the MQTT broker.	tcp://localhost:1883	String
cleanSession (common)	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	boolean
clientId (common)	MQTT client identifier. The identifier must be unique.		String
configuration (common)	To use the shared Paho configuration.		PahoConfiguration

Name	Description	Default	Type
connectionTimeout (common)	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	int
filePersistenceDirectory (common)	Base directory used by file persistence. Will by default use user directory.		String
keepAliveInterval (common)	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	int
maxInflight (common)	Sets the max inflight. please increase this value in a high traffic environment. The default value is 10.	10	int
maxReconnectDelay (common)	Get the maximum time (in millis) to wait between reconnects.	128000	int
mqttVersion (common)	Sets the MQTT version. The default action is to connect with version 3.1.1, and to fall back to 3.1 if that fails. Version 3.1.1 or 3.1 can be selected specifically, with no fall back, by using the MQTT_VERSION_3_1_1 or MQTT_VERSION_3_1 options respectively.		int
persistence (common)	Client persistence to be used - memory or file. Enum values: • FILE • MEMORY	MEMORY	PahoPersistence
qos (common)	Client quality of service level (0-2).	2	int
retained (common)	Retain option.	false	boolean

Name	Description	Default	Type
serverURIs (common)	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
willPayload (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String
willQos (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		int

Name	Description	Default	Type
willRetained (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.	false	boolean
willTopic (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
client (advanced)	To use a shared Paho client.		MqttClient
customWebSocketHeaders (advanced)	Sets the Custom WebSocket Headers for the WebSocket Connection.		Properties
executorServiceTimeout (advanced)	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	int
httpsHostnameVerificationEnabled (security)	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	boolean
password (security)	Password to be used for authentication against the MQTT broker.		String
socketFactory (security)	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings.		SocketFactory
sslClientProps (security)	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that		Properties

Name	Description	Default	Type
	contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509.		
sslHostnameVerifier (security)	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier.		HostnameVerifier
userName (security)	Username to be used for authentication against the MQTT broker.		String

47.4. ENDPOINT OPTIONS

The Paho endpoint is configured using URI syntax:

```
paho:topic
```

with the following path and query parameters:

47.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
topic (common)	**Required** Name of the topic.		String

47.4.2. Query Parameters (31 parameters)

Name	Description	Default	Type
automaticReconnect (common)	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	boolean
brokerUrl (common)	The URL of the MQTT broker.	tcp://localhost:1883	String
cleanSession (common)	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	boolean
clientId (common)	MQTT client identifier. The identifier must be unique.		String
connectionTimeout (common)	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	int

Name	Description	Default	Type
filePersistenceDirectory (common)	Base directory used by file persistence. Will by default use user directory.		String
keepAliveInterval (common)	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	int
maxInflight (common)	Sets the max inflight. please increase this value in a high traffic environment. The default value is 10.	10	int
maxReconnectDelay (common)	Get the maximum time (in millis) to wait between reconnects.	128000	int
mqttVersion (common)	Sets the MQTT version. The default action is to connect with version 3.1.1, and to fall back to 3.1 if that fails. Version 3.1.1 or 3.1 can be selected specifically, with no fall back, by using the MQTT_VERSION_3_1_1 or MQTT_VERSION_3_1 options respectively.		int
persistence (common)	Client persistence to be used - memory or file. Enum values: • FILE • MEMORY	MEMORY	PahoPersistence
qos (common)	Client quality of service level (0-2).	2	int
retained (common)	Retain option.	false	boolean

Name	Description	Default	Type
serverURIs (common)	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
willPayload (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String
willQos (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		int

Name	Description	Default	Type
willRetained (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.	false	boolean
willTopic (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
client (advanced)	To use an existing mqtt client.		MqttClient
customWebSocketHeaders (advanced)	Sets the Custom WebSocket Headers for the WebSocket Connection.		Properties
executorServiceTimeout (advanced)	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	int
httpsHostnameVerificationEnabled (security)	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	boolean
password (security)	Password to be used for authentication against the MQTT broker.		String
socketFactory (security)	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings.		SocketFactory
sslClientProps (security)	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to		Properties

Name	Description	Default	Type
	use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509.		
sslHostnameVerifier (security)	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier.		HostnameVerifier
userName (security)	Username to be used for authentication against the MQTT broker.		String

47.5. HEADERS

The following headers are recognized by the Paho component:

Header	Java constant	Endpoint type	Value type	Description
CamelMqttTopic	PahoConstants.MQTT_TOPIC	Consumer	String	The name of the topic
CamelMqttQoS	PahoConstants.MQTT_QOS	Consumer	Integer	QualityOfService of the incoming message
CamelPahoOverrideTopic	PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC	Producer	String	Name of topic to override and send to instead of topic specified on endpoint

47.6. DEFAULT PAYLOAD TYPE

By default Camel Paho component operates on the binary payloads extracted out of (or put into) the MQTT message:

```
// Receive payload
byte[] payload = (byte[]) consumerTemplate.receiveBody("paho:topic");

// Send payload
byte[] payload = "message".getBytes();
producerTemplate.sendBody("paho:topic", payload);
```

But of course Camel build-in type conversion API can perform the automatic data type transformations for you. In the example below Camel automatically converts binary payload into **String** (and conversely):

```
// Receive payload
String payload = consumerTemplate.receiveBody("paho:topic", String.class);

// Send payload
String payload = "message";
producerTemplate.sendBody("paho:topic", payload);
```

47.7. SAMPLES

For example the following snippet reads messages from the MQTT broker installed on the same host as the Camel router:

```
from("paho:some/queue")
    .to("mock:test");
```

While the snippet below sends message to the MQTT broker:

```
from("direct:test")
    .to("paho:some/target/queue");
```

For example this is how to read messages from the remote MQTT broker:

```
from("paho:some/queue?brokerUrl=tcp://iot.eclipse.org:1883")
    .to("mock:test");
```

And here we override the default topic and set to a dynamic topic

```
from("direct:test")
    .setHeader(PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC, simple("${header.customerId}"))
    .to("paho:some/target/queue");
```

47.8. SPRING BOOT AUTO-CONFIGURATION

When using paho with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-paho-starter</artifactId>
</dependency>
```

The component supports 32 options, which are listed below.

Name	Description	Default	Type
camel.component.paho.automatic-reconnect	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	Boolean
camel.component.paho.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.paho.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.paho.broker-url	The URL of the MQTT broker.	tcp://localhost:1883	String
camel.component.paho.clean-session	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	Boolean
camel.component.paho.client	To use a shared Paho client. The option is a org.eclipse.paho.client.mqttv3.MqttClient type.		MqttClient
camel.component.paho.client-id	MQTT client identifier. The identifier must be unique.		String
camel.component.paho.configuration	To use the shared Paho configuration. The option is a org.apache.camel.component.paho.PahoConfiguration type.		PahoConfiguration
camel.component.paho.connection-timeout	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	Integer

Name	Description	Default	Type
camel.component.paho.custom-web-socket-headers	Sets the Custom WebSocket Headers for the WebSocket Connection. The option is a java.util.Properties type.		Properties
camel.component.paho.enabled	Whether to enable auto configuration of the paho component. This is enabled by default.		Boolean
camel.component.paho.executor-service-timeout	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	Integer
camel.component.paho.file-persistence-directory	Base directory used by file persistence. Will by default use user directory.		String
camel.component.paho.https-hostname-verification-enabled	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	Boolean
camel.component.paho.keep-alive-interval	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	Integer

Name	Description	Default	Type
camel.component.paho.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.paho.max-inflight	Sets the max inflight. please increase this value in a high traffic environment. The default value is 10.	10	Integer
camel.component.paho.max-reconnect-delay	Get the maximum time (in millis) to wait between reconnects.	128000	Integer
camel.component.paho.mqtt-version	Sets the MQTT version. The default action is to connect with version 3.1.1, and to fall back to 3.1 if that fails. Version 3.1.1 or 3.1 can be selected specifically, with no fall back, by using the MQTT_VERSION_3_1_1 or MQTT_VERSION_3_1 options respectively.		Integer
camel.component.paho.password	Password to be used for authentication against the MQTT broker.		String
camel.component.paho.persistence	Client persistence to be used - memory or file.		PahoPersistence
camel.component.paho.qos	Client quality of service level (0-2).	2	Integer
camel.component.paho.retained	Retain option.	false	Boolean

Name	Description	Default	Type
camel.component.paho.server-u-r-is	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
camel.component.paho.socket-factory	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings. The option is a javax.net.SocketFactory type.		SocketFactory
camel.component.paho.ssl-client-props	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the		Properties

Name	Description	Default	Type
	KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509. The option is a java.util.Properties type.		
camel.component.paho.ssl-hostname-verifier	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier. The option is a javax.net.ssl.HostnameVerifier type.		HostnameVerifier

Name	Description	Default	Type
camel.component.paho.user-name	Username to be used for authentication against the MQTT broker.		String
camel.component.paho.will-payload	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String
camel.component.paho.will-qos	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		Integer
camel.component.paho.will-retained	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.	false	Boolean
camel.component.paho.will-topic	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to The byte payload for the message. The quality of service to publish the message at (0, 1 or 2). Whether or not the message should be retained.		String

CHAPTER 48. PAHO MQTT 5

Both producer and consumer are supported

Paho MQTT5 component provides connector for the MQTT messaging protocol using the Eclipse Paho library with MQTT v5. Paho is one of the most popular MQTT libraries, so if you would like to integrate it with your Java project - Camel Paho connector is a way to go.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-paho-mqtt5</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

48.1. URI FORMAT

```
paho-mqtt5:topic[?options]
```

Where **topic** is the name of the topic.

48.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

48.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

48.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

48.3. COMPONENT OPTIONS

The Paho MQTT 5 component supports 32 options, which are listed below.

Name	Description	Default	Type
automaticReconnect (common)	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	boolean
brokerUrl (common)	The URL of the MQTT broker.	tcp://localhost:1883	String
cleanStart (common)	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	boolean
clientId (common)	MQTT client identifier. The identifier must be unique.		String
configuration (common)	To use the shared Paho configuration.		PahoMqtt5Configuration

Name	Description	Default	Type
connectionTimeout (common)	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	int
filePersistenceDirectory (common)	Base directory used by file persistence. Will by default use user directory.		String
keepAliveInterval (common)	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	int
maxReconnectDelay (common)	Get the maximum time (in millis) to wait between reconnects.	128000	int
persistence (common)	Client persistence to be used - memory or file. Enum values: • FILE • MEMORY	MEMORY	PahoMqtt5Persistence
qos (common)	Client quality of service level (0-2).	2	int
receiveMaximum (common)	Sets the Receive Maximum. This value represents the limit of QoS 1 and QoS 2 publications that the client is willing to process concurrently. There is no mechanism to limit the number of QoS 0 publications that the Server might try to send. The default value is 65535.	65535	int
retained (common)	Retain option.	false	boolean

Name	Description	Default	Type
serverURIs (common)	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
sessionExpiryInterval (common)	Sets the Session Expiry Interval. This value, measured in seconds, defines the maximum time that the broker will maintain the session for once the client disconnects. Clients should only connect with a long Session Expiry interval if they intend to connect to the server at some later point in time. By default this value is -1 and so will not be sent, in this case, the session will not expire. If a 0 is sent, the session will end immediately once the Network Connection is closed. When the client has determined that it has no longer any use for the session, it should disconnect with a Session Expiry Interval set to 0.	-1	long

Name	Description	Default	Type
willMqttProperties (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The MQTT properties set for the message.		MqttProperties
willPayload (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The byte payload for the message.		String
willQos (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The quality of service to publish the message at (0, 1 or 2).	1	int
willRetained (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. Whether or not the message should be retained.	false	boolean
willTopic (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
client (advanced)	To use a shared Paho client.		MqttClient
customWebSocketHeaders (advanced)	Sets the Custom WebSocket Headers for the WebSocket Connection.		Map
executorServiceTimeout (advanced)	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	int
httpsHostnameVerificationEnabled (security)	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	boolean
password (security)	Password to be used for authentication against the MQTT broker.		String
socketFactory (security)	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings.		SocketFactory
sslClientProps (security)	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is		Properties

Name	Description	Default	Type
	available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509.		

Name	Description	Default	Type
sslHostnameVerifier (security)	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier.		HostnameVerifier
userName (security)	Username to be used for authentication against the MQTT broker.		String

48.4. ENDPOINT OPTIONS

The Paho MQTT 5 endpoint is configured using URI syntax:

```
paho-mqtt5:topic
```

with the following path and query parameters:

48.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
topic (common)	**Required** Name of the topic.		String

48.4.2. Query Parameters (32 parameters)

Name	Description	Default	Type
automaticReconnect (common)	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	boolean
brokerUrl (common)	The URL of the MQTT broker.	tcp://localhost:1883	String

Name	Description	Default	Type
cleanStart (common)	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	boolean
clientId (common)	MQTT client identifier. The identifier must be unique.		String
connectionTimeout (common)	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	int
filePersistenceDirectory (common)	Base directory used by file persistence. Will by default use user directory.		String
keepAliveInterval (common)	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	int
maxReconnectDelay (common)	Get the maximum time (in millis) to wait between reconnects.	128000	int

Name	Description	Default	Type
persistence (common)	Client persistence to be used - memory or file. Enum values: • FILE • MEMORY	MEMORY	PahoMqtt5Persistence
qos (common)	Client quality of service level (0-2).	2	int
receiveMaximum (common)	Sets the Receive Maximum. This value represents the limit of QoS 1 and QoS 2 publications that the client is willing to process concurrently. There is no mechanism to limit the number of QoS 0 publications that the Server might try to send. The default value is 65535.	65535	int
retained (common)	Retain option.	false	boolean

Name	Description	Default	Type
serverURIs (common)	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
sessionExpiryInterval (common)	Sets the Session Expiry Interval. This value, measured in seconds, defines the maximum time that the broker will maintain the session for once the client disconnects. Clients should only connect with a long Session Expiry interval if they intend to connect to the server at some later point in time. By default this value is -1 and so will not be sent, in this case, the session will not expire. If a 0 is sent, the session will end immediately once the Network Connection is closed. When the client has determined that it has no longer any use for the session, it should disconnect with a Session Expiry Interval set to 0.	-1	long

Name	Description	Default	Type
willMqttProperties (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The MQTT properties set for the message.		MqttProperties
willPayload (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The byte payload for the message.		String
willQos (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The quality of service to publish the message at (0, 1 or 2).	1	int
willRetained (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. Whether or not the message should be retained.	false	boolean
willTopic (common)	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
client (advanced)	To use an existing mqtt client.		MqttClient
customWebSocketHeaders (advanced)	Sets the Custom WebSocket Headers for the WebSocket Connection.		Map
executorServiceTimeout (advanced)	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	int
httpsHostnameVerificationEnabled (security)	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	boolean
password (security)	Password to be used for authentication against the MQTT broker.		String
socketFactory (security)	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings.		SocketFactory
sslClientProps (security)	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is		Properties

Name	Description	Default	Type
	available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509.		

Name	Description	Default	Type
sslHostnameVerifier (security)	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier.		HostnameVerifier
userName (security)	Username to be used for authentication against the MQTT broker.		String

48.5. HEADERS

The following headers are recognized by the Paho component:

Header	Java constant	Endpoint type	Value type	Description
CamelMqttTopic	PahoConstants.MQTT_TOPIC	Consumer	String	The name of the topic
CamelMqttQoS	PahoConstants.MQTT_QOS	Consumer	Integer	QualityOfService of the incoming message
CamelPahoOverrideTopic	PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC	Producer	String	Name of topic to override and send to instead of topic specified on endpoint

48.6. DEFAULT PAYLOAD TYPE

By default Camel Paho component operates on the binary payloads extracted out of (or put into) the MQTT message:

```
// Receive payload
byte[] payload = (byte[]) consumerTemplate.receiveBody("paho:topic");

// Send payload
byte[] payload = "message".getBytes();
producerTemplate.sendBody("paho:topic", payload);
```

But of course Camel build-in type conversion API can perform the automatic data type transformations for you. In the example below Camel automatically converts binary payload into **String** (and conversely):

```
// Receive payload
String payload = consumerTemplate.receiveBody("paho:topic", String.class);

// Send payload
String payload = "message";
producerTemplate.sendBody("paho:topic", payload);
```

-

48.7. SAMPLES

For example the following snippet reads messages from the MQTT broker installed on the same host as the Camel router:

```
from("paho:some/queue")
    .to("mock:test");
```

While the snippet below sends message to the MQTT broker:

```
from("direct:test")
    .to("paho:some/target/queue");
```

For example this is how to read messages from the remote MQTT broker:

```
from("paho:some/queue?brokerUrl=tcp://iot.eclipse.org:1883")
    .to("mock:test");
```

And here we override the default topic and set to a dynamic topic

```
from("direct:test")
    .setHeader(PahoConstants.CAMEL_PAHO_OVERRIDE_TOPIC, simple("${header.customerId}"))
    .to("paho:some/target/queue");
```

48.8. SPRING BOOT AUTO-CONFIGURATION

When using paho-mqtt5 with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-paho-mqtt5-starter</artifactId>
</dependency>
```

The component supports 33 options, which are listed below.

Name	Description	Default	Type
camel.component.paho-mqtt5.automatic-reconnect	Sets whether the client will automatically attempt to reconnect to the server if the connection is lost. If set to false, the client will not attempt to automatically reconnect to the server in the event that the connection is lost. If set to true, in the event that the connection is lost, the client will attempt to reconnect to the server. It will initially wait 1 second before it attempts to reconnect, for every failed reconnect attempt, the delay will double until it is at 2 minutes at which point the delay will stay at 2 minutes.	true	Boolean

Name	Description	Default	Type
camel.component.paho-mqtt5.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.paho-mqtt5.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.paho-mqtt5.broker-url	The URL of the MQTT broker.	tcp://localhost:1883	String
camel.component.paho-mqtt5.clean-start	Sets whether the client and server should remember state across restarts and reconnects. If set to false both the client and server will maintain state across restarts of the client, the server and the connection. As state is maintained: Message delivery will be reliable meeting the specified QOS even if the client, server or connection are restarted. The server will treat a subscription as durable. If set to true the client and server will not maintain state across restarts of the client, the server or the connection. This means Message delivery to the specified QOS cannot be maintained if the client, server or connection are restarted The server will treat a subscription as non-durable.	true	Boolean
camel.component.paho-mqtt5.client	To use a shared Paho client. The option is a org.eclipse.paho.mqttv5.client.MqttClient type.		MqttClient
camel.component.paho-mqtt5.client-id	MQTT client identifier. The identifier must be unique.		String
camel.component.paho-mqtt5.configuration	To use the shared Paho configuration. The option is a org.apache.camel.component.paho.mqtt5.PahoMqtt5Configuration type.		PahoMqtt5Configuration

Name	Description	Default	Type
camel.component.paho-mqtt5.connection-timeout	Sets the connection timeout value. This value, measured in seconds, defines the maximum time interval the client will wait for the network connection to the MQTT server to be established. The default timeout is 30 seconds. A value of 0 disables timeout processing meaning the client will wait until the network connection is made successfully or fails.	30	Integer
camel.component.paho-mqtt5.custom-web-socket-headers	Sets the Custom WebSocket Headers for the WebSocket Connection.		Map
camel.component.paho-mqtt5.enabled	Whether to enable auto configuration of the paho-mqtt5 component. This is enabled by default.		Boolean
camel.component.paho-mqtt5.executor-service-timeout	Set the time in seconds that the executor service should wait when terminating before forcefully terminating. It is not recommended to change this value unless you are absolutely sure that you need to.	1	Integer
camel.component.paho-mqtt5.file-persistence-directory	Base directory used by file persistence. Will by default use user directory.		String
camel.component.paho-mqtt5.https-hostname-verification-enabled	Whether SSL HostnameVerifier is enabled or not. The default value is true.	true	Boolean
camel.component.paho-mqtt5.keep-alive-interval	Sets the keep alive interval. This value, measured in seconds, defines the maximum time interval between messages sent or received. It enables the client to detect if the server is no longer available, without having to wait for the TCP/IP timeout. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small ping message, which the server will acknowledge. A value of 0 disables keepalive processing in the client. The default value is 60 seconds.	60	Integer

Name	Description	Default	Type
camel.component.paho-mqtt5.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.paho-mqtt5.max-reconnect-delay	Get the maximum time (in millis) to wait between reconnects.	128000	Integer
camel.component.paho-mqtt5.password	Password to be used for authentication against the MQTT broker.		String
camel.component.paho-mqtt5.persistence	Client persistence to be used - memory or file.		PahoMqtt5Persistence
camel.component.paho-mqtt5.qos	Client quality of service level (0-2).	2	Integer
camel.component.paho-mqtt5.receive-maximum	Sets the Receive Maximum. This value represents the limit of QoS 1 and QoS 2 publications that the client is willing to process concurrently. There is no mechanism to limit the number of QoS 0 publications that the Server might try to send. The default value is 65535.	65535	Integer
camel.component.paho-mqtt5.retained	Retain option.	false	Boolean

Name	Description	Default	Type
camel.component.paho-mqtt5.server-u-r-is	Set a list of one or more serverURIs the client may connect to. Multiple servers can be separated by comma. Each serverURI specifies the address of a server that the client may connect to. Two types of connection are supported tcp:// for a TCP connection and ssl:// for a TCP connection secured by SSL/TLS. For example: tcp://localhost:1883 ssl://localhost:8883 If the port is not specified, it will default to 1883 for tcp:// URIs, and 8883 for ssl:// URIs. If serverURIs is set then it overrides the serverURI parameter passed in on the constructor of the MQTT client. When an attempt to connect is initiated the client will start with the first serverURI in the list and work through the list until a connection is established with a server. If a connection cannot be made to any of the servers then the connect attempt fails. Specifying a list of servers that a client may connect to has several uses: High Availability and reliable message delivery Some MQTT servers support a high availability feature where two or more equal MQTT servers share state. An MQTT client can connect to any of the equal servers and be assured that messages are reliably delivered and durable subscriptions are maintained no matter which server the client connects to. The cleansession flag must be set to false if durable subscriptions and/or reliable message delivery is required. Hunt List A set of servers may be specified that are not equal (as in the high availability option). As no state is shared across the servers reliable message delivery and durable subscriptions are not valid. The cleansession flag must be set to true if the hunt list mode is used.		String
camel.component.paho-mqtt5.session-expiry-interval	Sets the Session Expiry Interval. This value, measured in seconds, defines the maximum time that the broker will maintain the session for once the client disconnects. Clients should only connect with a long Session Expiry interval if they intend to connect to the server at some later point in time. By default this value is -1 and so will not be sent, in this case, the session will not expire. If a 0 is sent, the session will end immediately once the Network Connection is closed. When the client has determined that it has no longer any use for the session, it should disconnect with a Session Expiry Interval set to 0.	-1	Long

Name	Description	Default	Type
camel.component.paho-mqtt5.socket-factory	Sets the SocketFactory to use. This allows an application to apply its own policies around the creation of network sockets. If using an SSL connection, an SSLSocketFactory can be used to supply application-specific security settings. The option is a javax.net.SocketFactory type.		SocketFactory
camel.component.paho-mqtt5.ssl-client-props	Sets the SSL properties for the connection. Note that these properties are only valid if an implementation of the Java Secure Socket Extensions (JSSE) is available. These properties are not used if a custom SocketFactory has been set. The following properties can be used: com.ibm.ssl.protocol One of: SSL, SSLv3, TLS, TLSv1, SSL_TLS. com.ibm.ssl.contextProvider Underlying JSSE provider. For example IBMJSSE2 or SunJSSE com.ibm.ssl.keyStore The name of the file that contains the KeyStore object that you want the KeyManager to use. For example /mydir/etc/key.p12 com.ibm.ssl.keyStorePassword The password for the KeyStore object that you want the KeyManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.keyStoreType Type of key store, for example PKCS12, JKS, or JCEKS. com.ibm.ssl.keyStoreProvider Key store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.trustStore The name of the file that contains the KeyStore object that you want the TrustManager to use. com.ibm.ssl.trustStorePassword The password for the TrustStore object that you want the TrustManager to use. The password can either be in plain-text, or may be obfuscated using the static method: com.ibm.micro.security.Password.obfuscate(char password). This obfuscates the password using a simple and insecure XOR and Base64 encoding mechanism. Note that this is only a simple scrambler to obfuscate clear-text passwords. com.ibm.ssl.trustStoreType The type of KeyStore object that you want the default TrustManager to use. Same possible values as keyStoreType. com.ibm.ssl.trustStoreProvider Trust store provider, for example IBMJCE or IBMJCEFIPS. com.ibm.ssl.enabledCipherSuites A list of which ciphers are enabled. Values are dependent on the		Properties

Name	Description	Default	Type
	provider, for example: SSL_RSA_WITH_AES_128_CBC_SHA;SSL_RSA_WITH_3DES_EDE_CBC_SHA. com.ibm.ssl.keyManager Sets the algorithm that will be used to instantiate a KeyManagerFactory object instead of using the default algorithm available in the platform. Example values: IbmX509 or IBMJ9X509. com.ibm.ssl.trustManager Sets the algorithm that will be used to instantiate a TrustManagerFactory object instead of using the default algorithm available in the platform. Example values: PKIX or IBMJ9X509. The option is a java.util.Properties type.		
camel.component.paho-mqtt5.ssl-hostname-verifier	Sets the HostnameVerifier for the SSL connection. Note that it will be used after handshake on a connection and you should do actions by yourself when hostname is verified error. There is no default HostnameVerifier. The option is a javax.net.ssl.HostnameVerifier type.		HostnameVerifier
camel.component.paho-mqtt5.user-name	Username to be used for authentication against the MQTT broker.		String
camel.component.paho-mqtt5.will-mqtt-properties	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The MQTT properties set for the message. The option is a org.eclipse.paho.mqttv5.common.packet.MqttProperties type.		MqttProperties
camel.component.paho-mqtt5.will-payload	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The byte payload for the message.		String
camel.component.paho-mqtt5.will-qos	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The quality of service to publish the message at (0, 1 or 2).	1	Integer
camel.component.paho-mqtt5.will-retained	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. Whether or not the message should be retained.	false	Boolean

Name	Description	Default	Type
camel.component.paho-mqtt5.will-topic	Sets the Last Will and Testament (LWT) for the connection. In the event that this client unexpectedly loses its connection to the server, the server will publish a message to itself using the supplied details. The topic to publish to.		String

CHAPTER 49. PLATFORM HTTP

Since Camel 3.0

Only consumer is supported

The Platform HTTP is used to allow Camel to use the existing HTTP server from the runtime. For example when running Camel on Spring Boot, Quarkus, or other runtimes.

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-platform-http</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

49.1. PLATFORM HTTP PROVIDER

To use Platform HTTP a provider (engine) is required to be available on the classpath. The purpose is to have drivers for different runtimes such as Quarkus, VertX, or Spring Boot.

At this moment there is only support for Quarkus and VertX by **camel-platform-http-vertx**. This JAR must be on the classpath otherwise the Platform HTTP component cannot be used and an exception will be thrown on startup.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-platform-http-vertx</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

49.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

49.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

49.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

49.2.3. Component Options

The Platform HTTP component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
engine (advanced)	An HTTP Server engine implementation to serve the requests.		PlatformHttpEngine

49.2.4. Endpoint Options

The Platform HTTP endpoint is configured using URI syntax:

```
platform-http:path
```

with the following path and query parameters:

49.2.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
path (consumer)	**Required** The path under which this endpoint serves the HTTP requests, for proxy use 'proxy'.		String

49.2.4.2. Query Parameters (11 parameters)

Name	Description	Default	Type
consumes (consumer)	The content type this endpoint accepts as an input, such as application/xml or application/json. null or / mean no restriction.		String
httpMethodRestrict (consumer)	A comma separated list of HTTP methods to serve, e.g. GET,POST . If no methods are specified, all methods will be served.		String
matchOnUriPrefix (consumer)	Whether or not the consumer should try to find a target consumer by matching the URI prefix if no exact match is found.	false	boolean
muteException (consumer)	If enabled and an Exchange failed processing on the consumer side the response's body won't contain the exception's stack trace.	true	boolean
produces (consumer)	The content type this endpoint produces, such as application/xml or application/json.		String
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
fileNameExtWhitelist (consumer (advanced))	A comma or whitespace separated list of file extensions. Uploads having these extensions will be stored locally. Null value or asterisk () will allow all files.		String
headerFilterStrategy (advanced)	To use a custom HeaderFilterStrategy to filter headers to and from Camel message.		HeaderFilterStrategy
platformHttpEngine (advanced)	An HTTP Server engine implementation to serve the requests of this endpoint.		PlatformHttpEngine

49.3. SPRING BOOT AUTO-CONFIGURATION

When using platform-http with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-platform-http-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.platform-http.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.platform-http.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.platform-http.enabled	Whether to enable auto configuration of the platform-http component. This is enabled by default.		Boolean
camel.component.platform-http.engine	An HTTP Server engine implementation to serve the requests. The option is a org.apache.camel.component.platform.http.spi.PlatformHttpEngine type.		PlatformHttpEngine

49.3.1. Implementing a reverse proxy

Platform HTTP component can act as a reverse proxy, in that case some headers are populated from the absolute URL received on the request line of the HTTP request. Those headers are specific to the underlining platform.

At this moment, this feature is only supported for Vert.x in **camel-platform-http-vertx** component.

CHAPTER 50. QUARTZ

Only consumer is supported

The Quartz component provides a scheduled delivery of messages using the Quartz Scheduler 2.x. Each endpoint represents a different timer (in Quartz terms, a Trigger and JobDetail).

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-quartz</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

50.1. URI FORMAT

```
quartz://timerName?options
quartz://groupName/timerName?options
quartz://groupName/timerName?cron=expression
quartz://timerName?cron=expression
```

The component uses either a **CronTrigger** or a **SimpleTrigger**. If no cron expression is provided, the component uses a simple trigger. If no **groupName** is provided, the quartz component uses the **Camel** group name.

50.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

50.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

50.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

50.3. COMPONENT OPTIONS

The Quartz component supports 13 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
enableJmx (consumer)	Whether to enable Quartz JMX which allows to manage the Quartz scheduler from JMX. This options is default true.	true	boolean
prefixInstanceName (consumer)	Whether to prefix the Quartz Scheduler instance name with the CamelContext name. This is enabled by default, to let each CamelContext use its own Quartz scheduler instance by default. You can set this option to false to reuse Quartz scheduler instances between multiple CamelContext's.	true	boolean
prefixJobNameWithEndpointId (consumer)	Whether to prefix the quartz job with the endpoint id. This option is default false.	false	boolean
properties (consumer)	Properties to configure the Quartz scheduler.		Map
propertiesFile (consumer)	File name of the properties to load from the classpath.		String
propertiesRef (consumer)	References to an existing Properties or Map to lookup in the registry to use for configuring quartz.		String

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
scheduler (advanced)	To use the custom configured Quartz scheduler, instead of creating a new Scheduler.		Scheduler
schedulerFactory (advanced)	To use the custom SchedulerFactory which is used to create the Scheduler.		SchedulerFactory
autoStartScheduler (scheduler)	Whether or not the scheduler should be auto started. This options is default true.	true	boolean
interruptJobsOnShutdown (scheduler)	Whether to interrupt jobs on shutdown which forces the scheduler to shutdown quicker and attempt to interrupt any running jobs. If this is enabled then any running jobs can fail due to being interrupted. When a job is interrupted then Camel will mark the exchange to stop continue routing and set java.util.concurrent.RejectedExecutionException as caused exception. Therefore use this with care, as its often better to allow Camel jobs to complete and shutdown gracefully.	false	boolean
startDelayedSeconds (scheduler)	Seconds to wait before starting the quartz scheduler.		int

50.4. ENDPOINT OPTIONS

The Quartz endpoint is configured using URI syntax:

```
quartz:groupName/triggerName
```

with the following path and query parameters:

50.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
groupName (consumer)	The quartz group name to use. The combination of group name and trigger name should be unique.	Camel	String

Name	Description	Default	Type
triggerName (consumer)	**Required** The quartz trigger name to use. The combination of group name and trigger name should be unique.		String

50.4.2. Query Parameters (17 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
cron (consumer)	Specifies a cron expression to define when to trigger.		String
deleteJob (consumer)	If set to true, then the trigger automatically delete when route stop. Else if set to false, it will remain in scheduler. When set to false, it will also mean user may reuse pre-configured trigger with camel Uri. Just ensure the names match. Notice you cannot have both deleteJob and pauseJob set to true.	true	boolean
durableJob (consumer)	Whether or not the job should remain stored after it is orphaned (no triggers point to it).	false	boolean
pauseJob (consumer)	If set to true, then the trigger automatically pauses when route stop. Else if set to false, it will remain in scheduler. When set to false, it will also mean user may reuse pre-configured trigger with camel Uri. Just ensure the names match. Notice you cannot have both deleteJob and pauseJob set to true.	false	boolean
recoverableJob (consumer)	Instructs the scheduler whether or not the job should be re-executed if a 'recovery' or 'fail-over' situation is encountered.	false	boolean
stateful (consumer)	Uses a Quartz PersistJobDataAfterExecution and DisallowConcurrentExecution instead of the default job.	false	boolean

Name	Description	Default	Type
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
customCalendar (advanced)	Specifies a custom calendar to avoid specific range of date.		Calendar
jobParameters (advanced)	To configure additional options on the job.		Map
prefixJobNameWithEndpointId (advanced)	Whether the job name should be prefixed with endpoint id.	false	boolean
triggerParameters (advanced)	To configure additional options on the trigger.		Map
usingFixedCamelContextName (advanced)	If it is true, JobDataMap uses the CamelContext name directly to reference the CamelContext, if it is false, JobDataMap uses use the CamelContext management name which could be changed during the deploy time.	false	boolean
autoStartScheduler (scheduler)	Whether or not the scheduler should be auto started.	true	boolean
startDelayedSeconds (scheduler)	Seconds to wait before starting the quartz scheduler.		int
triggerStartDelay (scheduler)	In case of scheduler has already started, we want the trigger start slightly after current time to ensure endpoint is fully started before the job kicks in. Negative value shifts trigger start time in the past.	500	long

50.4.3. Configuring quartz.properties file

By default Quartz will look for a **quartz.properties** file in the **org/quartz** directory of the classpath. If you are using WAR deployments this means just drop the quartz.properties in **WEB-INF/classes/org/quartz**.

However the Camel Quartz component also allows you to configure properties:

Parameter	Default	Type	Description
properties	**null**	**Properties**	You can configure a **java.util.Properties** instance.
propertiesFile	**null**	**String**	File name of the properties to load from the classpath

To do this you can configure this in Spring XML as follows

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
   <property name="propertiesFile" value="com/mycompany/myquartz.properties"/>
</bean>
```

50.5. ENABLING QUARTZ SCHEDULER IN JMX

You need to configure the quartz scheduler properties to enable JMX.
That is typically setting the option **"org.quartz.scheduler.jmx.export"** to a **true** value in the configuration file.

This option is set to true by default, unless explicitly disabled.

50.6. STARTING THE QUARTZ SCHEDULER

The Quartz component offers an option to let the Quartz scheduler be started delayed, or not auto started at all.

This is an example:

```
<bean id="quartz" class="org.apache.camel.component.quartz.QuartzComponent">
   <property name="startDelayedSeconds" value="5"/>
</bean>
```

50.7. CLUSTERING

If you use Quartz in clustered mode, e.g. the **JobStore** is clustered. Then the Quartz component will not pause/remove triggers when a node is being stopped/shutdown. This allows the trigger to keep running on the other nodes in the cluster.

NOTE

When running in clustered node no checking is done to ensure unique job name/group for endpoints.

50.8. MESSAGE HEADERS

Camel adds the getters from the Quartz Execution Context as header values. The following headers are added:
calendar, **fireTime**, **jobDetail**, **jobInstance**, **jobRuntTime**, **mergedJobDataMap**, **nextFireTime**, **previousFireTime**, **refireCount**, **result**, **scheduledFireTime**, **scheduler**, **trigger**, **triggerName**, **triggerGroup**.

The **fireTime** header contains the **java.util.Date** of when the exchange was fired.

50.9. USING CRON TRIGGERS

Quartz supports Cron-like expressions for specifying timers in a handy format. You can use these expressions in the **cron** URI parameter; though to preserve valid URI encoding we allow **+** to be used instead of spaces.

For example, the following will fire a message every five minutes starting at 12pm (noon) to 6pm on weekdays:

```
from("quartz://myGroup/myTimerName?cron=0+0/5+12-18+?+*+MON-FRI")
    .to("activemq:Totally.Rocks");
```

which is equivalent to using the cron expression

```
0 0/5 12-18 ? * MON-FRI
```

The following table shows the URI character encodings we use to preserve valid URI syntax:

URI Character	Cron character
+	*Space*

50.10. SPECIFYING TIME ZONE

The Quartz Scheduler allows you to configure time zone per trigger. For example to use a timezone of your country, then you can do as follows:

```
quartz://groupName/timerName?cron=0+0/5+12-18+?+*+MON-
FRI&trigger.timeZone=Europe/Stockholm
```

The timeZone value is the values accepted by **java.util.TimeZone**.

50.11. CONFIGURING MISFIRE INSTRUCTIONS

The quartz scheduler can be configured with a misfire instruction to handle misfire situations for the trigger. The concrete trigger type that you are using will have defined a set of additional **MISFIRE_INSTRUCTION_XXX** constants that may be set as this property's value.

For example to configure the simple trigger to use misfire instruction 4:

```
quartz://myGroup/myTimerName?trigger.repeatInterval=2000&trigger.misfireInstruction=4
```

And likewise you can configure the cron trigger with one of its misfire instructions as well:

```
quartz://myGroup/myTimerName?cron=0/2+*+*+*+*+?&trigger.misfireInstruction=2
```

The simple and cron triggers has the following misfire instructions representative:

50.11.1. SimpleTrigger.MISFIRE_INSTRUCTION_FIRE_NOW = 1 (default)

Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be fired now by Scheduler.

This instruction should typically only be used for 'one-shot' (non-repeating) Triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT.

50.11.2. SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING = 2

Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count left as-is. This does obey the Trigger end-time however, so if 'now' is after the end-time the Trigger will not fire again.

Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time).

50.11.3. SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINI = 3

Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to 'now' (even if the associated Calendar excludes 'now') with the repeat count set to what it would be, if it had not missed any firings. This does obey the Trigger end-time however, so if 'now' is after the end-time the Trigger will not fire again.

Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with. Instead, the repeat count on the trigger will be changed to whatever the remaining repeat count is (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time).

This instruction could cause the Trigger to go to the 'COMPLETE' state after firing 'now', if all the repeat-fire-times where missed.

50.11.4. SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINI = 4

Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to the next scheduled time after 'now' - taking into account any associated Calendar and with the repeat count set to what it would be, if it had not missed any firings.

NOTE

This instruction could cause the Trigger to go directly to the 'COMPLETE' state if all fire-times where missed.

50.11.5. SimpleTrigger.MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING = 5

Instructs the Scheduler that upon a mis-fire situation, the SimpleTrigger wants to be re-scheduled to the next scheduled time after 'now' - taking into account any associated Calendar, and with the repeat count left unchanged.

NOTE

This instruction could cause the Trigger to go directly to the 'COMPLETE' state if the end-time of the trigger has arrived.

50.11.6. CronTrigger.MISFIRE_INSTRUCTION_FIRE_ONCE_NOW = 1 (default)

Instructs the Scheduler that upon a mis-fire situation, the CronTrigger wants to be fired now by Scheduler.

50.11.7. CronTrigger.MISFIRE_INSTRUCTION_DO_NOTHING = 2

Instructs the Scheduler that upon a mis-fire situation, the CronTrigger wants to have it's next-fire-time updated to the next time in the schedule after the current time (taking into account any associated Calendar but it does not want to be fired now.

50.12. USING QUARTZSCHEDULEDPOLLCONSUMERSCHEDULER

The Quartz component provides a Polling Consumer scheduler which allows to use cron based scheduling for Polling Consumer such as the File and FTP consumers.

For example to use a cron based expression to poll for files every 2nd second, then a Camel route can be define simply as:

```
from("file:inbox?scheduler=quartz&scheduler.cron=0/2+*+*+*+*+?")
  .to("bean:process");
```

Notice we define the **scheduler=quartz** to instruct Camel to use the Quartz based scheduler. Then we use **scheduler.xxx** options to configure the scheduler. The Quartz scheduler requires the cron option to be set.

The following options is supported:

Parameter	Default	Type	Description
quartzScheduler	**null**	**org.quartz.Scheduler**	To use a custom Quartz scheduler. If none configure then the shared scheduler from the component is used.
cron	**null**	**String**	**Mandatory**: To define the cron expression for triggering the polls.
triggerId	**null**	**String**	To specify the trigger id. If none provided then an UUID is generated and used.

Parameter	Default	Type	Description
triggerGroup	**QuartzScheduledPollConsumerScheduler**	**String**	To specify the trigger group.
timeZone	**Default**	**TimeZone**	The time zone to use for the CRON trigger.

IMPORTANT

Remember configuring these options from the endpoint URIs must be prefixed with **scheduler**.

For example to configure the trigger id and group:

```
    from("file:inbox?scheduler=quartz&scheduler.cron=0/2+*+*+*+*+?
&scheduler.triggerId=myId&scheduler.triggerGroup=myGroup")
        .to("bean:process");
```

There is also a CRON scheduler in Spring, so you can use the following as well:

```
    from("file:inbox?scheduler=spring&scheduler.cron=0/2+*+*+*+*+?")
        .to("bean:process");
```

50.13. CRON COMPONENT SUPPORT

The Quartz component can be used as implementation of the Camel Cron component.

Maven users will need to add the following additional dependency to their **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-cron</artifactId>
    <version>{CamelSBVersion}</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

Users can then use the cron component instead of the quartz component, as in the following route:

```
    from("cron://name?schedule=0+0/5+12-18+?+*+MON-FRI")
    .to("activemq:Totally.Rocks");
```

50.14. SPRING BOOT AUTO-CONFIGURATION

When using quartz with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-quartz-starter</artifactId>
</dependency>
```

The component supports 14 options, which are listed below.

Name	Description	Default	Type
camel.component.quartz.auto-start-scheduler	Whether or not the scheduler should be auto started. This options is default true.	true	Boolean
camel.component.quartz.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.quartz.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.quartz.enable-jmx	Whether to enable Quartz JMX which allows to manage the Quartz scheduler from JMX. This options is default true.	true	Boolean
camel.component.quartz.enabled	Whether to enable auto configuration of the quartz component. This is enabled by default.		Boolean
camel.component.quartz.interrupt-jobs-on-shutdown	Whether to interrupt jobs on shutdown which forces the scheduler to shutdown quicker and attempt to interrupt any running jobs. If this is enabled then any running jobs can fail due to being interrupted. When a job is interrupted then Camel will mark the exchange to stop continue routing and set java.util.concurrent.RejectedExecutionException as caused exception. Therefore use this with care, as its often better to allow Camel jobs to complete and shutdown gracefully.	false	Boolean

Name	Description	Default	Type
camel.component.quartz.prefix-instance-name	Whether to prefix the Quartz Scheduler instance name with the CamelContext name. This is enabled by default, to let each CamelContext use its own Quartz scheduler instance by default. You can set this option to false to reuse Quartz scheduler instances between multiple CamelContext's.	true	Boolean
camel.component.quartz.prefix-job-name-with-endpoint-id	Whether to prefix the quartz job with the endpoint id. This option is default false.	false	Boolean
camel.component.quartz.properties	Properties to configure the Quartz scheduler.		Map
camel.component.quartz.properties-file	File name of the properties to load from the classpath.		String
camel.component.quartz.properties-ref	References to an existing Properties or Map to lookup in the registry to use for configuring quartz.		String
camel.component.quartz.scheduler	To use the custom configured Quartz scheduler, instead of creating a new Scheduler. The option is a org.quartz.Scheduler type.		Scheduler
camel.component.quartz.scheduler-factory	To use the custom SchedulerFactory which is used to create the Scheduler. The option is a org.quartz.SchedulerFactory type.		SchedulerFactory
camel.component.quartz.start-delayed-seconds	Seconds to wait before starting the quartz scheduler.		Integer

CHAPTER 51. REF

Both producer and consumer are supported

The Ref component is used for lookup of existing endpoints bound in the Registry.

51.1. URI FORMAT

```
ref:someName[?options]
```

Where **someName** is the name of an endpoint in the Registry (usually, but not always, the Spring registry). If you are using the Spring registry, **someName** would be the bean ID of an endpoint in the Spring registry.

51.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

51.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

51.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

51.3. COMPONENT OPTIONS

The Ref component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

51.4. ENDPOINT OPTIONS

The Ref endpoint is configured using URI syntax:

```
ref:name
```

with the following path and query parameters:

51.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** Name of endpoint to lookup in the registry.		String

51.4.2. Query Parameters (4 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

51.5. RUNTIME LOOKUP

This component can be used when you need dynamic discovery of endpoints in the Registry where you can compute the URI at runtime. Then you can look up the endpoint using the following code:

```
// lookup the endpoint
String myEndpointRef = "bigspenderOrder";
Endpoint endpoint = context.getEndpoint("ref:" + myEndpointRef);

Producer producer = endpoint.createProducer();
Exchange exchange = producer.createExchange();
```

```
exchange.getIn().setBody(payloadToSend);
// send the exchange
producer.process(exchange);
```

And you could have a list of endpoints defined in the Registry such as:

```
<camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <endpoint id="normalOrder" uri="activemq:order.slow"/>
    <endpoint id="bigspenderOrder" uri="activemq:order.high"/>
</camelContext>
```

51.6. SAMPLE

In the sample below we use the **ref:** in the URI to reference the endpoint with the spring ID, **endpoint2**:

You could, of course, have used the **ref** attribute instead:

```
<to uri="ref:endpoint2"/>
```

Which is the more common way to write it.

51.7. SPRING BOOT AUTO-CONFIGURATION

When using ref with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-ref-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.ref.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.ref.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.ref.enabled	Whether to enable auto configuration of the ref component. This is enabled by default.		Boolean
camel.component.ref.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 52. REST

Both producer and consumer are supported

The REST component allows to define REST endpoints (consumer) using the Rest DSL and plugin to other Camel components as the REST transport.

The rest component can also be used as a client (producer) to call REST services.

52.1. URI FORMAT

```
rest://method:path[:uriTemplate]?[options]
```

52.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

52.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

52.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

52.3. COMPONENT OPTIONS

The REST component supports 8 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
consumerComponentName (consumer)	The Camel Rest component to use for (consumer) the REST transport, such as jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
apiDoc (producer)	The swagger api doc resource to use. The resource is loaded from classpath by default and must be in JSON format.		String
componentName (producer)	**Deprecated** The Camel Rest component to use for (producer) the REST transport, such as http, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestProducerFactory is registered in the registry. If either one is found, then that is being used.		String
host (producer)	Host and port of HTTP service to use (override host in swagger schema).		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
producerComponentName (producer)	The Camel Rest component to use for (producer) the REST transport, such as http, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestProducerFactory is registered in the registry. If either one is found, then that is being used.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

52.4. ENDPOINT OPTIONS

The REST endpoint is configured using URI syntax:

```
rest:method:path:uriTemplate
```

with the following path and query parameters:

52.4.1. Path Parameters (3 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
method (common)	**Required** HTTP method to use. Enum values: • get • post • put • delete • patch • head • trace • connect • options		String
path (common)	**Required** The base path.		String
uriTemplate (common)	The uri template.		String

52.4.2. Query Parameters (16 parameters)

Name	Description	Default	Type
consumes (common)	Media type such as: 'text/xml', or 'application/json' this REST service accepts. By default we accept all kinds of types.		String
inType (common)	To declare the incoming POJO binding type as a FQN class name.		String
outType (common)	To declare the outgoing POJO binding type as a FQN class name.		String
produces (common)	Media type such as: 'text/xml', or 'application/json' this REST service returns.		String
routeId (common)	Name of the route this REST services creates.		String

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
consumerComponentName (consumer)	The Camel Rest component to use for (consumer) the REST transport, such as jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
description (consumer)	Human description to document this REST service.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: ● InOnly ● InOut ● InOptionalOut		ExchangePattern
apiDoc (producer)	The openapi api doc resource to use. The resource is loaded from classpath by default and must be in JSON format.		String

Name	Description	Default	Type
bindingMode (producer)	Configures the binding mode for the producer. If set to anything other than 'off' the producer will try to convert the body of the incoming message from inType to the json or xml, and the response from json or xml to outType. Enum values: • auto • off • json • xml • json_xml		RestBindingMode
host (producer)	Host and port of HTTP service to use (override host in openapi schema).		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
producerComponentName (producer)	The Camel Rest component to use for (producer) the REST transport, such as http, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestProducerFactory is registered in the registry. If either one is found, then that is being used.		String
queryParameters (producer)	Query parameters for the HTTP service to call. The query parameters can contain multiple parameters separated by ampersand such such as foo=123&bar=456.		String

52.5. SUPPORTED REST COMPONENTS

The following components support rest consumer (Rest DSL):

- camel-servlet

The following components support rest producer:

- camel-http

52.6. PATH AND URITEMPLATE SYNTAX

The path and uriTemplate option is defined using a REST syntax where you define the REST context path using support for parameters.

NOTE

If no uriTemplate is configured then path option works the same way. It does not matter if you configure only path or if you configure both options. Though configuring both a path and uriTemplate is a more common practice with REST.

The following is a Camel route using a path only

```
from("rest:get:hello")
  .transform().constant("Bye World");
```

And the following route uses a parameter which is mapped to a Camel header with the key "me".

```
from("rest:get:hello/{me}")
  .transform().simple("Bye ${header.me}");
```

The following examples have configured a base path as "hello" and then have two REST services configured using uriTemplates.

```
from("rest:get:hello:/{me}")
  .transform().simple("Hi ${header.me}");

from("rest:get:hello:/french/{me}")
  .transform().simple("Bonjour ${header.me}");
```

52.7. REST PRODUCER EXAMPLES

You can use the rest component to call REST services like any other Camel component.

For example to call a REST service on using **hello/{me}** you can do

```
from("direct:start")
  .to("rest:get:hello/{me}");
```

And then the dynamic value **{me}** is mapped to Camel message with the same name. So to call this REST service you can send an empty message body and a header as shown:

```
template.sendBodyAndHeader("direct:start", null, "me", "Donald Duck");
```

The Rest producer needs to know the hostname and port of the REST service, which you can configure using the host option as shown:

```
from("direct:start")
  .to("rest:get:hello/{me}?host=myserver:8080/foo");
```

Instead of using the host option, you can configure the host on the **restConfiguration** as shown:

```
restConfiguration().host("myserver:8080/foo");

from("direct:start")
  .to("rest:get:hello/{me}");
```

You can use the **producerComponent** to select which Camel component to use as the HTTP client, for example to use http you can do:

```
restConfiguration().host("myserver:8080/foo").producerComponent("http");

from("direct:start")
  .to("rest:get:hello/{me}");
```

52.8. REST PRODUCER BINDING

The REST producer supports binding using JSon or XML like the rest-dsl does.

For example to use jetty with json binding mode turned on you can configure this in the rest configuration:

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBindingMode.json)
;

from("direct:start")
  .to("rest:post:user");
```

Then when calling the REST service using rest producer it will automatic bind any POJOs to json before calling the REST service:

```
  UserPojo user = new UserPojo();
  user.setId(123);
  user.setName("Donald Duck");

  template.sendBody("direct:start", user);
```

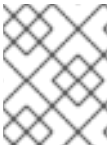
In the example above we send a POJO instance **UserPojo** as the message body. And because we have turned on JSon binding in the rest configuration, then the POJO will be marshalled from POJO to JSon before calling the REST service.

However if you want to also perform binding for the response message (eg what the REST service send back as response) you would need to configure the **outType** option to specify what is the classname of the POJO to unmarshal from JSon to POJO.

For example if the REST service returns a JSon payload that binds to **com.foo.MyResponsePojo** you can configure this as shown:

```
restConfiguration().component("jetty").host("localhost").port(8080).bindingMode(RestBindingMode.json)
```

```
;

  from("direct:start")
    .to("rest:post:user?outType=com.foo.MyResponsePojo");
```



NOTE

You must configure **outType** option if you want POJO binding to happen for the response messages received from calling the REST service.

52.9. MORE EXAMPLES

See Rest DSL which offers more examples and how you can use the Rest DSL to define those in a nicer RESTful way.

There is a **camel-example-servlet-rest-tomcat** example in the Apache Camel distribution, that demonstrates how to use the Rest DSL with SERVLET as transport that can be deployed on Apache Tomcat, or similar web containers.

52.10. SPRING BOOT AUTO-CONFIGURATION

When using rest with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-rest-starter</artifactId>
</dependency>
```

The component supports 12 options, which are listed below.

Name	Description	Default	Type
camel.component.rest-api.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.rest-api.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.rest-api.enabled	Whether to enable auto configuration of the rest-api component. This is enabled by default.		Boolean
camel.component.rest.api-doc	The swagger api doc resource to use. The resource is loaded from classpath by default and must be in JSON format.		String
camel.component.rest.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.rest.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.rest.consumer-component-name	The Camel Rest component to use for (consumer) the REST transport, such as jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.component.rest.enabled	Whether to enable auto configuration of the rest component. This is enabled by default.		Boolean
camel.component.rest.host	Host and port of HTTP service to use (override host in swagger schema).		String

Name	Description	Default	Type
camel.component.rest.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.rest.producer-component-name	The Camel Rest component to use for (producer) the REST transport, such as http, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestProducerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.component.rest.component-name	**Deprecated** The Camel Rest component to use for (producer) the REST transport, such as http, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestProducerFactory is registered in the registry. If either one is found, then that is being used.		String

CHAPTER 53. SAGA

Only producer is supported

The Saga component provides a bridge to execute custom actions within a route using the Saga EIP.

The component should be used for advanced tasks, such as deciding to complete or compensate a Saga with completionMode set to **MANUAL**.

Refer to the Saga EIP documentation for help on using sagas in common scenarios.

53.1. URI FORMAT

```
saga:action
```

53.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

53.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

53.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

53.3. COMPONENT OPTIONS

The Saga component supports 2 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

53.4. ENDPOINT OPTIONS

The Saga endpoint is configured using URI syntax:

```
saga:action
```

with the following path and query parameters:

53.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
action (producer)	**Required** Action to execute (complete or compensate). Enum values: • COMPLETE • COMPENSATE		SagaEndpointAction

53.4.2. Query Parameters (1 parameters)

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

53.5. SPRING BOOT AUTO-CONFIGURATION

When using saga with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-saga-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type
camel.component.saga.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.saga.enabled	Whether to enable auto configuration of the saga component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.saga.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 54. SALESFORCE

Both producer and consumer are supported

This component supports producer and consumer endpoints to communicate with Salesforce using Java DTOs.
There is a companion maven plugin Camel Salesforce Plugin that generates these DTOs (see further below).

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-salesforce</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

NOTE

Developers wishing to contribute to the component are instructed to look at the README.md file on instructions on how to get started and setup your environment for running integration tests.

54.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

54.1.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

54.1.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

54.2. COMPONENT OPTIONS

The Salesforce component supports 90 options, which are listed below.

Name	Description	Default	Type
apexMethod (common)	APEX method name.		String
apexQueryParams (common)	Query params for APEX method.		Map
apiVersion (common)	Salesforce API version.	53.0	String
backoffIncrement (common)	Backoff interval increment for Streaming connection restart attempts for failures beyond CometD auto-reconnect.	1000	long
batchId (common)	Bulk API Batch ID.		String
contentType (common)	Bulk API content type, one of XML, CSV, ZIP_XML, ZIP_CSV. Enum values: • XML • CSV • JSON • ZIP_XML • ZIP_CSV • ZIP_JSON		ContentType
defaultReplayId (common)	Default replayId setting if no value is found in initialReplayIdMap.	-1	Long
fallBackReplayId (common)	ReplayId to fall back to after an Invalid Replay Id response.	-1	Long

Name	Description	Default	Type
format (common)	Payload format to use for Salesforce API calls, either JSON or XML, defaults to JSON. As of Camel 3.12, this option only applies to the Raw operation. Enum values: • JSON • XML		PayloadFormat
httpClient (common)	Custom Jetty Http Client to use to connect to Salesforce.		SalesforceHttpClient
httpClientConnectionTimeout (common)	Connection timeout used by the HttpClient when connecting to the Salesforce server.	60000	long
httpClientIdleTimeout (common)	Timeout used by the HttpClient when waiting for response from the Salesforce server.	10000	long
httpMaxContentLength (common)	Max content length of an HTTP response.		Integer
httpRequestBufferSize (common)	HTTP request buffer size. May need to be increased for large SOQL queries.	8192	Integer
includeDetails (common)	Include details in Salesforce1 Analytics report, defaults to false.		Boolean
initialReplayIdMap (common)	Replay IDs to start from per channel name.		Map
instanceId (common)	Salesforce1 Analytics report execution instance ID.		String
jobId (common)	Bulk API Job ID.		String
limit (common)	Limit on number of returned records. Applicable to some of the API, check the Salesforce documentation.		Integer
locator (common)	Locator provided by salesforce Bulk 2.0 API for use in getting results for a Query job.		String

Name	Description	Default	Type
maxBackoff (common)	Maximum backoff interval for Streaming connection restart attempts for failures beyond CometD auto-reconnect.	30000	long
maxRecords (common)	The maximum number of records to retrieve per set of results for a Bulk 2.0 Query. The request is still subject to the size limits. If you are working with a very large number of query results, you may experience a timeout before receiving all the data from Salesforce. To prevent a timeout, specify the maximum number of records your client is expecting to receive in the maxRecords parameter. This splits the results into smaller sets with this value as the maximum size.		Integer
notFoundBehaviour (common)	Sets the behaviour of 404 not found status received from Salesforce API. Should the body be set to NULL NotFoundBehaviour#NULL or should a exception be signaled on the exchange NotFoundBehaviour#EXCEPTION - the default. Enum values: • EXCEPTION • NULL	EXCEPTION	NotFoundBehaviour
notifyForFields (common)	Notify for fields, options are ALL, REFERENCED, SELECT, WHERE. Enum values: • ALL • REFERENCED • SELECT • WHERE		NotifyForFieldsEnum
notifyForOperationCreate (common)	Notify for create operation, defaults to false (API version = 29.0).		Boolean
notifyForOperationDelete (common)	Notify for delete operation, defaults to false (API version = 29.0).		Boolean

Name	Description	Default	Type
notifyForOperations (common)	Notify for operations, options are ALL, CREATE, EXTENDED, UPDATE (API version 29.0). Enum values: • ALL • CREATE • EXTENDED • UPDATE		NotifyForOperationsEnum
notifyForOperationUndelete (common)	Notify for un-delete operation, defaults to false (API version = 29.0).		Boolean
notifyForOperationUpdate (common)	Notify for update operation, defaults to false (API version = 29.0).		Boolean
objectMapper (common)	Custom Jackson ObjectMapper to use when serializing/deserializing Salesforce objects.		ObjectMapper
packages (common)	In what packages are the generated DTO classes. Typically the classes would be generated using camel-salesforce-maven-plugin. Set it if using the generated DTOs to gain the benefit of using short SObject names in parameters/header values. Multiple packages can be separated by comma.		String
pkChunking (common)	Use PK Chunking. Only for use in original Bulk API. Bulk 2.0 API performs PK chunking automatically, if necessary.		Boolean
pkChunkingChunkSize (common)	Chunk size for use with PK Chunking. If unspecified, salesforce default is 100,000. Maximum size is 250,000.		Integer
pkChunkingParent (common)	Specifies the parent object when you're enabling PK chunking for queries on sharing objects. The chunks are based on the parent object's records rather than the sharing object's records. For example, when querying on AccountShare, specify Account as the parent object. PK chunking is supported for sharing objects as long as the parent object is supported.		String

Name	Description	Default	Type
pkChunkingStartRow (common)	Specifies the 15-character or 18-character record ID to be used as the lower boundary for the first chunk. Use this parameter to specify a starting ID when restarting a job that failed between batches.		String
queryLocator (common)	Query Locator provided by salesforce for use when a query results in more records than can be retrieved in a single call. Use this value in a subsequent call to retrieve additional records.		String
rawPayload (common)	Use raw payload String for request and response (either JSON or XML depending on format), instead of DTOs, false by default.	false	boolean
reportId (common)	Salesforce1 Analytics report Id.		String
reportMetadata (common)	Salesforce1 Analytics report metadata for filtering.		ReportMetadata
resultId (common)	Bulk API Result ID.		String
sObjectBlobFieldName (common)	SObject blob field name.		String
sObjectClass (common)	Fully qualified SObject class name, usually generated using camel-salesforce-maven-plugin.		String
sObjectFields (common)	SObject fields to retrieve.		String
sObjectId (common)	SObject ID if required by API.		String
sObjectIdName (common)	SObject external ID field name.		String
sObjectIdValue (common)	SObject external ID field value.		String
sObjectName (common)	SObject name if required or supported by API.		String
sObjectQuery (common)	Salesforce SOQL query string.		String

Name	Description	Default	Type
sObjectSearch (common)	Salesforce SOSL search string.		String
updateTopic (common)	Whether to update an existing Push Topic when using the Streaming API, defaults to false.	false	boolean
config (common (advanced))	Global endpoint configuration - use to set values that are common to all endpoints.		SalesforceEndpointConfig
httpClientProperties (common (advanced))	Used to set any properties that can be configured on the underlying HTTP client. Have a look at properties of SalesforceHttpClient and the Jetty HttpClient for all available options.		Map
longPollingTransportProperties (common (advanced))	Used to set any properties that can be configured on the LongPollingTransport used by the BayeuxClient (CometD) used by the streaming api.		Map
workerPoolMaxSize (common (advanced))	Maximum size of the thread pool used to handle HTTP responses.	20	int
workerPoolSize (common (advanced))	Size of the thread pool used to handle HTTP responses.	10	int
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
allOrNone (producer)	Composite API option to indicate to rollback all records if any are not successful.	false	boolean
apexUrl (producer)	APEX method URL.		String
compositeMethod (producer)	Composite (raw) method.		String

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
rawHttpHeaders (producer)	Comma separated list of message headers to include as HTTP parameters for Raw operation.		String
rawMethod (producer)	HTTP method to use for the Raw operation.		String
rawPath (producer)	The portion of the endpoint URL after the domain name. E.g., '/services/data/v52.0/sobjects/Account/'.		String
rawQueryParameters (producer)	Comma separated list of message headers to include as query parameters for Raw operation. Do not url-encode values as this will be done automatically.		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
httpProxyExcludedAddresses (proxy)	A list of addresses for which HTTP proxy server should not be used.		Set
httpProxyHost (proxy)	Hostname of the HTTP proxy server to use.		String
httpProxyIncludedAddresses (proxy)	A list of addresses for which HTTP proxy server should be used.		Set
httpProxyPort (proxy)	Port number of the HTTP proxy server to use.		Integer

Name	Description	Default	Type
httpProxySocks4 (proxy)	If set to true the configures the HTTP proxy to use as a SOCKS4 proxy.	false	boolean
authenticationType (security)	Explicit authentication method to be used, one of USERNAME_PASSWORD, REFRESH_TOKEN or JWT. Salesforce component can auto-determine the authentication method to use from the properties set, set this property to eliminate any ambiguity. Enum values: • USERNAME_PASSWORD • REFRESH_TOKEN • JWT		AuthenticationType
clientId (security)	**Required** OAuth Consumer Key of the connected app configured in the Salesforce instance setup. Typically a connected app needs to be configured but one can be provided by installing a package.		String
clientSecret (security)	OAuth Consumer Secret of the connected app configured in the Salesforce instance setup.		String
httpProxyAuthUri (security)	Used in authentication against the HTTP proxy server, needs to match the URI of the proxy server in order for the httpProxyUsername and httpProxyPassword to be used for authentication.		String
httpProxyPassword (security)	Password to use to authenticate against the HTTP proxy server.		String
httpProxyRealm (security)	Realm of the proxy server, used in preemptive Basic/Digest authentication methods against the HTTP proxy server.		String
httpProxySecure (security)	If set to false disables the use of TLS when accessing the HTTP proxy.	true	boolean
httpProxyUseDigestAuth (security)	If set to true Digest authentication will be used when authenticating to the HTTP proxy, otherwise Basic authorization method will be used.	false	boolean
httpProxyUsername (security)	Username to use to authenticate against the HTTP proxy server.		String

Name	Description	Default	Type
instanceUrl (security)	URL of the Salesforce instance used after authentication, by default received from Salesforce on successful authentication.		String
jwtAudience (security)	Value to use for the Audience claim (aud) when using OAuth JWT flow. If not set, the login URL will be used, which is appropriate in most cases.		String
keystore (security)	KeyStore parameters to use in OAuth JWT flow. The KeyStore should contain only one entry with private key and certificate. Salesforce does not verify the certificate chain, so this can easily be a selfsigned certificate. Make sure that you upload the certificate to the corresponding connected app.		KeyStoreParameters
lazyLogin (security)	If set to true prevents the component from authenticating to Salesforce with the start of the component. You would generally set this to the (default) false and authenticate early and be immediately aware of any authentication issues.	false	boolean
loginConfig (security)	All authentication configuration in one nested bean, all properties set there can be set directly on the component as well.		SalesforceLoginConfig
loginUrl (security)	**Required** URL of the Salesforce instance used for authentication, by default set to https://login.salesforce.com.	https://login.salesforce.com	String
password (security)	Password used in OAuth flow to gain access to access token. It's easy to get started with password OAuth flow, but in general one should avoid it as it is deemed less secure than other flows. Make sure that you append security token to the end of the password if using one.		String

Name	Description	Default	Type
refreshToken (security)	Refresh token already obtained in the refresh token OAuth flow. One needs to setup a web application and configure a callback URL to receive the refresh token, or configure using the builtin callback at https://login.salesforce.com/services/oauth2/success or https://test.salesforce.com/services/oauth2/success and then retrive the refresh_token from the URL at the end of the flow. Note that in development organizations Salesforce allows hosting the callback web application at localhost.		String
sslContextParameters (security)	SSL parameters to use, see SSLContextParameters class for all available options.		SSLContextParameters
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean
userName (security)	Username used in OAuth flow to gain access to access token. It's easy to get started with password OAuth flow, but in general one should avoid it as it is deemed less secure than other flows.		String

54.3. ENDPOINT OPTIONS

The Salesforce endpoint is configured using URI syntax:

```
salesforce:operationName:topicName
```

with the following path and query parameters:

54.3.1. Path Parameters (2 parameters)

Name	Description	Default	Type
operationName (producer)	The operation to use. Enum values: • getVersions • getResources • getGlobalObjects		OperationName

Name	Description	Default	Type
	• getBasicInfo • getDescription • getSObject • createSObject • updateSObject • deleteSObject • getSObjectWithId • upsertSObject • deleteSObjectWithId • getBlobField • query • queryMore • queryAll • search • apexCall • recent • createJob • getJob • closeJob • abortJob • createBatch • getBatch • getAllBatches • getRequest • getResults • createBatchQuery • getQueryResultIds • getQueryResult • getRecentReports • getReportDescription • executeSyncReport • executeAsyncReport • getReportInstances		

Name	Description	Default	Type
	• getReportResults • limits • approval • approvals • composite-tree • composite-batch • composite • compositeRetrieveSObjectCollections • compositeCreateSObjectCollections • compositeUpdateSObjectCollections • compositeUpsertSObjectCollections • compositeDeleteSObjectCollections • bulk2GetAllJobs • bulk2CreateJob • bulk2GetJob • bulk2CreateBatch • bulk2CloseJob • bulk2AbortJob • bulk2DeleteJob • bulk2GetSuccessfulResults • bulk2GetFailedResults • bulk2GetUnprocessedRecords • bulk2CreateQueryJob • bulk2GetQueryJob • bulk2GetAllQueryJobs • bulk2GetQueryJobResults • bulk2AbortQueryJob • bulk2DeleteQueryJob • raw		
topicName (consumer)	The name of the topic/channel to use.		String

54.3.2. Query Parameters (57 parameters)

Name	Description	Default	Type
apexMethod (common)	APEX method name.		String
apexQueryParams (common)	Query params for APEX method.		Map
apiVersion (common)	Salesforce API version.	53.0	String
backoffIncrement (common)	Backoff interval increment for Streaming connection restart attempts for failures beyond CometD auto-reconnect.	1000	long
batchId (common)	Bulk API Batch ID.		String
contentType (common)	Bulk API content type, one of XML, CSV, ZIP_XML, ZIP_CSV. Enum values: • XML • CSV • JSON • ZIP_XML • ZIP_CSV • ZIP_JSON		ContentType
defaultReplayId (common)	Default replayId setting if no value is found in initialReplayIdMap.	-1	Long
fallBackReplayId (common)	ReplayId to fall back to after an Invalid Replay Id response.	-1	Long
format (common)	Payload format to use for Salesforce API calls, either JSON or XML, defaults to JSON. As of Camel 3.12, this option only applies to the Raw operation. Enum values: • JSON • XML		PayloadFormat

Name	Description	Default	Type
httpClient (common)	Custom Jetty Http Client to use to connect to Salesforce.		SalesforceHttpClient
includeDetails (common)	Include details in Salesforce1 Analytics report, defaults to false.		Boolean
initialReplayIdMap (common)	Replay IDs to start from per channel name.		Map
instanceId (common)	Salesforce1 Analytics report execution instance ID.		String
jobId (common)	Bulk API Job ID.		String
limit (common)	Limit on number of returned records. Applicable to some of the API, check the Salesforce documentation.		Integer
locator (common)	Locator provided by salesforce Bulk 2.0 API for use in getting results for a Query job.		String
maxBackoff (common)	Maximum backoff interval for Streaming connection restart attempts for failures beyond CometD auto-reconnect.	30000	long
maxRecords (common)	The maximum number of records to retrieve per set of results for a Bulk 2.0 Query. The request is still subject to the size limits. If you are working with a very large number of query results, you may experience a timeout before receiving all the data from Salesforce. To prevent a timeout, specify the maximum number of records your client is expecting to receive in the maxRecords parameter. This splits the results into smaller sets with this value as the maximum size.		Integer
notFoundBehaviour (common)	Sets the behaviour of 404 not found status received from Salesforce API. Should the body be set to NULL NotFoundBehaviour#NULL or should a exception be signaled on the exchange NotFoundBehaviour#EXCEPTION - the default. Enum values: • EXCEPTION • NULL	EXCEPTION	NotFoundBehaviour

Name	Description	Default	Type
notifyForFields (common)	Notify for fields, options are ALL, REFERENCED, SELECT, WHERE. Enum values: • ALL • REFERENCED • SELECT • WHERE		NotifyForFieldsEnum
notifyForOperationCreate (common)	Notify for create operation, defaults to false (API version = 29.0).		Boolean
notifyForOperationDelete (common)	Notify for delete operation, defaults to false (API version = 29.0).		Boolean
notifyForOperations (common)	Notify for operations, options are ALL, CREATE, EXTENDED, UPDATE (API version 29.0). Enum values: • ALL • CREATE • EXTENDED • UPDATE		NotifyForOperationsEnum
notifyForOperationUndelete (common)	Notify for un-delete operation, defaults to false (API version = 29.0).		Boolean
notifyForOperationUpdate (common)	Notify for update operation, defaults to false (API version = 29.0).		Boolean
objectMapper (common)	Custom Jackson ObjectMapper to use when serializing/deserializing Salesforce objects.		ObjectMapper
pkChunking (common)	Use PK Chunking. Only for use in original Bulk API. Bulk 2.0 API performs PK chunking automatically, if necessary.		Boolean

Name	Description	Default	Type
pkChunkingChunkSize (common)	Chunk size for use with PK Chunking. If unspecified, salesforce default is 100,000. Maximum size is 250,000.		Integer
pkChunkingParent (common)	Specifies the parent object when you're enabling PK chunking for queries on sharing objects. The chunks are based on the parent object's records rather than the sharing object's records. For example, when querying on AccountShare, specify Account as the parent object. PK chunking is supported for sharing objects as long as the parent object is supported.		String
pkChunkingStartRow (common)	Specifies the 15-character or 18-character record ID to be used as the lower boundary for the first chunk. Use this parameter to specify a starting ID when restarting a job that failed between batches.		String
queryLocator (common)	Query Locator provided by salesforce for use when a query results in more records than can be retrieved in a single call. Use this value in a subsequent call to retrieve additional records.		String
rawPayload (common)	Use raw payload String for request and response (either JSON or XML depending on format), instead of DTOs, false by default.	false	boolean
reportId (common)	Salesforce1 Analytics report Id.		String
reportMetadata (common)	Salesforce1 Analytics report metadata for filtering.		ReportMetadata
resultId (common)	Bulk API Result ID.		String
sObjectBlobFieldName (common)	SObject blob field name.		String
sObjectClass (common)	Fully qualified SObject class name, usually generated using camel-salesforce-maven-plugin.		String
sObjectFields (common)	SObject fields to retrieve.		String
sObjectId (common)	SObject ID if required by API.		String

Name	Description	Default	Type
sObjectIdName (common)	SObject external ID field name.		String
sObjectIdValue (common)	SObject external ID field value.		String
sObjectName (common)	SObject name if required or supported by API.		String
sObjectQuery (common)	Salesforce SOQL query string.		String
sObjectSearch (common)	Salesforce SOSL search string.		String
updateTopic (common)	Whether to update an existing Push Topic when using the Streaming API, defaults to false.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
replayId (consumer)	The replayId value to use when subscribing.		Long
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
allOrNone (producer)	Composite API option to indicate to rollback all records if any are not successful.	false	boolean
apexUrl (producer)	APEX method URL.		String
compositeMethod (producer)	Composite (raw) method.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
rawHttpHeaders (producer)	Comma separated list of message headers to include as HTTP parameters for Raw operation.		String
rawMethod (producer)	HTTP method to use for the Raw operation.		String
rawPath (producer)	The portion of the endpoint URL after the domain name. E.g., '/services/data/v52.0/sobjects/Account/'.		String
rawQueryParameters (producer)	Comma separated list of message headers to include as query parameters for Raw operation. Do not url-encode values as this will be done automatically.		String

54.4. AUTHENTICATING TO SALESFORCE

The component supports three OAuth authentication flows:

- OAuth 2.0 Username-Password Flow
- OAuth 2.0 Refresh Token Flow
- OAuth 2.0 JWT Bearer Token Flow

For each of the flow different set of properties needs to be set:

Table 54.1. Table 1. Properties to set for each authentication flow

Property	Where to find it on Salesforce	Flow
clientId	Connected App, Consumer Key	All flows
clientSecret	Connected App, Consumer Secret	Username-Password, Refresh Token
userName	Salesforce user username	Username-Password, JWT Bearer Token
password	Salesforce user password	Username-Password
refreshToken	From OAuth flow callback	Refresh Token
keystore	Connected App, Digital Certificate	JWT Bearer Token

The component auto determines what flow you're trying to configure, to be remove ambiguity set the **authenticationType** property.

NOTE

Using Username-Password Flow in production is not encouraged.

NOTE

The certificate used in JWT Bearer Token Flow can be a selfsigned certificate. The KeyStore holding the certificate and the private key must contain only single certificate-private key entry.

54.5. URI FORMAT

When used as a consumer, receiving streaming events, the URI scheme is:

```
salesforce:topic?options
```

When used as a producer, invoking the Salesforce REST APIs, the URI scheme is:

```
salesforce:operationName?options
```

54.6. PASSING IN SALESFORCE HEADERS AND FETCHING SALESFORCE RESPONSE HEADERS

There is support to pass Salesforce headers via inbound message headers, header names that start with **Sforce** or **x-sfdc** on the Camel message will be passed on in the request, and response headers that start with **Sforce** will be present in the outbound message headers.

For example to fetch API limits you can specify:

```
// in your Camel route set the header before Salesforce endpoint
//...
  .setHeader("Sforce-Limit-Info", constant("api-usage"))
  .to("salesforce:getGlobalObjects")
  .to(myProcessor);

// myProcessor will receive `Sforce-Limit-Info` header on the outbound
// message
class MyProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message in = exchange.getIn();
        String apiLimits = in.getHeader("Sforce-Limit-Info", String.class);
    }
}
```

In addition, HTTP response status code and text are available as headers **Exchange.HTTP_RESPONSE_CODE** and **Exchange.HTTP_RESPONSE_TEXT**.

54.7. SUPPORTED SALESFORCE APIS

The component supports the following Salesforce APIs

Producer endpoints can use the following APIs. Most of the APIs process one record at a time, the Query API can retrieve multiple Records.

54.7.1. Rest API

You can use the following for **operationName**:

- getVersions - Gets supported Salesforce REST API versions
- getResources - Gets available Salesforce REST Resource endpoints
- getGlobalObjects - Gets metadata for all available SObject types
- getBasicInfo - Gets basic metadata for a specific SObject type
- getDescription - Gets comprehensive metadata for a specific SObject type
- getSObject - Gets an SObject using its Salesforce Id
- createSObject - Creates an SObject
- updateSObject - Updates an SObject using Id
- deleteSObject - Deletes an SObject using Id
- getSObjectWithId - Gets an SObject using an external (user defined) id field
- upsertSObject - Updates or inserts an SObject using an external id
- deleteSObjectWithId - Deletes an SObject using an external id
- query - Runs a Salesforce SOQL query

- queryMore - Retrieves more results (in case of large number of results) using result link returned from the 'query' API
- search - Runs a Salesforce SOSL query
- limits - fetching organization API usage limits
- recent - fetching recent items
- approval - submit a record or records (batch) for approval process
- approvals - fetch a list of all approval processes
- composite - submit up to 25 possibly related REST requests and receive individual responses. It's also possible to use "raw" composite without limitation.
- composite-tree - create up to 200 records with parent-child relationships (up to 5 levels) in one go
- composite-batch - submit a composition of requests in batch
- compositeRetrieveSObjectCollections - Retrieve one or more records of the same object type.
- compositeCreateSObjectCollections - Add up to 200 records, returning a list of SaveSObjectResult objects.
- compositeUpdateSObjectCollections - Update up to 200 records, returning a list of SaveSObjectResult objects.
- compositeUpsertSObjectCollections - Create or update (upsert) up to 200 records based on an external ID field. Returns a list of UpsertSObjectResult objects.
- compositeDeleteSObjectCollections - Delete up to 200 records, returning a list of SaveSObjectResult objects.
- queryAll - Runs a SOQL query. It returns the results that are deleted because of a merge (merges up to three records into one of the records, deletes the others, and reparents any related records) or delete. Also returns the information about archived Task and Event records.
- getBlobField - Retrieves the specified blob field from an individual record.
- apexCall - Executes a user defined APEX REST API call.
- raw - Send requests to salesforce and have full, raw control over endpoint, parameters, body, etc.

For example, the following producer endpoint uses the upsertSObject API, with the sObjectIdName parameter specifying 'Name' as the external id field. The request message body should be an SObject DTO generated using the maven plugin. The response message will either be **null** if an existing record was updated, or **CreateSObjectResult** with an id of the new record, or a list of errors while creating the new object.

```
...to("salesforce:upsertSObject?sObjectIdName=Name")...
```

54.7.2. Bulk 2.0 API

The Bulk 2.0 API has a simplified model over the original Bulk API. Use it to quickly load a large amount of

data into salesforce, or query a large amount of data out of salesforce. Data must be provided in CSV format. The minimum API version for Bulk 2.0 is v41.0. The minimum API version for Bulk Queries is v47.0. DTO classes mentioned below are from the **org.apache.camel.component.salesforce.api.dto.bulkv2** package. The following operations are supported:

- **bulk2CreateJob** - Create a bulk job. Supply an instance of **Job** in the message body.
- **bulk2GetJob** - Get an existing Job. **jobId** parameter is required.
- **bulk2CreateBatch** - Add a Batch of CSV records to a job. Supply CSV data in the message body. The first row must contain headers. **jobId** parameter is required.
- **bulk2CloseJob** - Close a job. You must close the job in order for it to be processed or aborted/deleted. **jobId** parameter is required.
- **bulk2AbortJob** - Abort a job. **jobId** parameter is required.
- **bulk2DeleteJob** - Delete a job. **jobId** parameter is required.
- **bulk2GetSuccessfulResults** - Get successful results for a job. Returned message body will contain an InputStream of CSV data. **jobId** parameter is required.
- **bulk2GetFailedResults** - Get failed results for a job. Returned message body will contain an InputStream of CSV data. **jobId** parameter is required.
- **bulk2GetUnprocessedRecords** - Get unprocessed records for a job. Returned message body will contain an InputStream of CSV data. **jobId** parameter is required.
- **bulk2GetAllJobs** - Get all jobs. Response body is an instance of **Jobs**. If the **done** property is false, there are additional pages to fetch, and the **nextRecordsUrl** property contains the value to be set in the **queryLocator** parameter on subsequent calls.
- **bulk2CreateQueryJob** - Create a bulk query job. Supply an instance of **QueryJob** in the message body.
- **bulk2GetQueryJob** - Get a bulk query job. **jobId** parameter is required.
- **bulk2GetQueryJobResults** - Get bulk query job results. **jobId** parameter is required. Accepts **maxRecords** and **locator** parameters. Response message headers include **Sforce-NumberOfRecords** and **Sforce-Locator** headers. The value of **Sforce-Locator** can be passed into subsequent calls via the **locator** parameter.
- **bulk2AbortQueryJob** - Abort a bulk query job. **jobId** parameter is required.
- **bulk2DeleteQueryJob** - Delete a bulk query job. **jobId** parameter is required.
- **bulk2GetAllQueryJobs** - Get all jobs. Response body is an instance of **QueryJobs**. If the **done** property is false, there are additional pages to fetch, and the **nextRecordsUrl** property contains the value to be set in the **queryLocator** parameter on subsequent calls.

54.7.3. Rest Bulk (original) API

Producer endpoints can use the following APIs. All Job data formats, i.e. xml, csv, zip/xml, and zip/csv are supported.
The request and response have to be marshalled/unmarshalled by the route. Usually the request will be

some stream source like a CSV file,
and the response may also be saved to a file to be correlated with the request.

You can use the following for **operationName**:

- **createJob** - Creates a Salesforce Bulk Job. Must supply a **JobInfo** instance in body. PK Chunking is supported via the pkChunking* options. See an explanation here.
- **getJob** - Gets a Job using its Salesforce Id
- **closeJob** - Closes a Job
- **abortJob** - Aborts a Job
- **createBatch** - Submits a Batch within a Bulk Job
- **getBatch** - Gets a Batch using Id
- **getAllBatches** - Gets all Batches for a Bulk Job Id
- **getRequest** - Gets Request data (XML/CSV) for a Batch
- **getResults** - Gets the results of the Batch when its complete
- **createBatchQuery** - Creates a Batch from an SOQL query
- **getQueryResultIds** - Gets a list of Result Ids for a Batch Query
- **getQueryResult** - Gets results for a Result Id
- **getRecentReports** - Gets up to 200 of the reports you most recently viewed by sending a GET request to the Report List resource.
- **getReportDescription** - Retrieves the report, report type, and related metadata for a report, either in a tabular or summary or matrix format.
- **executeSyncReport** - Runs a report synchronously with or without changing filters and returns the latest summary data.
- **executeAsyncReport** - Runs an instance of a report asynchronously with or without filters and returns the summary data with or without details.
- **getReportInstances** - Returns a list of instances for a report that you requested to be run asynchronously. Each item in the list is treated as a separate instance of the report.
- **getReportResults**: Contains the results of running a report.

For example, the following producer endpoint uses the createBatch API to create a Job Batch. The in message must contain a body that can be converted into an **InputStream** (usually UTF-8 CSV or XML content from a file, etc.) and header fields 'jobId' for the Job and 'contentType' for the Job content type, which can be XML, CSV, ZIP_XML or ZIP_CSV. The put message body will contain **BatchInfo** on success, or throw a **SalesforceException** on error.

```
...to("salesforce:createBatch")..
```

54.7.4. Rest Streaming API

Consumer endpoints can use the following syntax for streaming endpoints to receive Salesforce notifications on create/update.

To create and subscribe to a topic

```
from("salesforce:CamelTestTopic?
notifyForFields=ALL&notifyForOperations=ALL&sObjectName=Merchandise__c&updateTopic=true&sO
bjectQuery=SELECT Id, Name FROM Merchandise__c")...
```

To subscribe to an existing topic

```
from("salesforce:CamelTestTopic&sObjectName=Merchandise__c")...
```

54.7.5. Platform events

To emit a platform event use **createSObject** operation. And set the message body can be JSON string or InputStream with key-value data – in that case **sObjectName** needs to be set to the API name of the event, or a class that extends from AbstractDTOBase with the appropriate class name for the event.

For example using a DTO:

```
class Order_Event__e extends AbstractDTOBase {
  @JsonProperty("OrderNumber")
  private String orderNumber;
  // ... other properties and getters/setters
}

from("timer:tick")
    .process(exchange -> {
        final Message in = exchange.getIn();
        String orderNumber = "ORD" + exchange.getProperty(Exchange.TIMER_COUNTER);
        Order_Event__e event = new Order_Event__e();
        event.setOrderNumber(orderNumber);
        in.setBody(event);
    })
    .to("salesforce:createSObject");
```

Or using JSON event data:

```
from("timer:tick")
    .process(exchange -> {
        final Message in = exchange.getIn();
        String orderNumber = "ORD" + exchange.getProperty(Exchange.TIMER_COUNTER);
        in.setBody("{\"OrderNumber\":\"" + orderNumber + "\"}");
    })
    .to("salesforce:createSObject?sObjectName=Order_Event__e");
```

To receive platform events use the consumer endpoint with the API name of the platform event prefixed with **event/** (or **/event/**), e.g.: **salesforce:events/Order_Event__e**. Processor consuming from that endpoint will receive either **org.apache.camel.component.salesforce.api.dto.PlatformEvent** object or **org.cometd.bayeux.Message** in the body depending on the **rawPayload** being **false** or **true** respectively.

For example, in the simplest form to consume one event:

```
PlatformEvent event = consumer.receiveBody("salesforce:event/Order_Event__e",
PlatformEvent.class);
```

54.7.6. Change data capture events

On the one hand, Salesforce could be configured to emit notifications for record changes of select objects. On the other hand, the Camel Salesforce component could react to such notifications, allowing for instance to synchronize those changes into an external system .

The notifications of interest could be specified in the **from("salesforce:XXX")** clause of a Camel route via the subscription channel, e.g:

```
from("salesforce:data/ChangeEvents?replayId=-1").log("being notified of all change events")
from("salesforce:data/AccountChangeEvent?replayId=-1").log("being notified of change events for
Account records")
from("salesforce:data/Employee__ChangeEvent?replayId=-1").log("being notified of change events
for Employee__c custom object")
```

The received message contains either **java.util.Map<String,Object>** or **org.cometd.bayeux.Message** in the body depending on the **rawPayload** being **false** or **true** respectively. The **CamelSalesforceChangeType** header could be valued to one of **CREATE**, **UPDATE**, **DELETE** or **UNDELETE**.

More details about how to use the Camel Salesforce component change data capture capabilities could be found in the ChangeEventsConsumerIntegrationTest.

The Salesforce developer guide is a good fit to better know the subtleties of implementing a change data capture integration application. The dynamic nature of change event body fields, high level replication steps as well as security considerations could be of interest.

54.8. EXAMPLES

54.8.1. Uploading a document to a ContentWorkspace

Create the ContentVersion in Java, using a Processor instance:

```
public class ContentProcessor implements Processor {
    public void process(Exchange exchange) throws Exception {
        Message message = exchange.getIn();

        ContentVersion cv = new ContentVersion();
        ContentWorkspace cw = getWorkspace(exchange);
        cv.setFirstPublishLocationId(cw.getId());
        cv.setTitle("test document");
        cv.setPathOnClient("test_doc.html");
        byte[] document = message.getBody(byte[].class);
        ObjectMapper mapper = new ObjectMapper();
        String enc = mapper.convertValue(document, String.class);
        cv.setVersionDataUrl(enc);
        message.setBody(cv);
    }

    protected ContentWorkspace getWorkSpace(Exchange exchange) {
```

```
        // Look up the content workspace somehow, maybe use enrich() to add it to a
        // header that can be extracted here
        ----
    }
}
```

Give the output from the processor to the Salesforce component:

```
from("file:///home/camel/library")
    .to(new ContentProcessor())     // convert bytes from the file into a ContentVersion SObject
                                    // for the salesforce component
    .to("salesforce:createSObject");
```

54.9. USING SALESFORCE LIMITS API

With **salesforce:limits** operation you can fetch of API limits from Salesforce and then act upon that data received. The result of **salesforce:limits** operation is mapped to **org.apache.camel.component.salesforce.api.dto.Limits** class and can be used in a custom processors or expressions.

For instance, consider that you need to limit the API usage of Salesforce so that 10% of daily API requests is left for other routes. The body of output message contains an instance of **org.apache.camel.component.salesforce.api.dto.Limits** object that can be used in conjunction with Content Based Router and Content Based Router and Spring Expression Language (SpEL) to choose when to perform queries.

Notice how multiplying **1.0** with the integer value held in **body.dailyApiRequests.remaining** makes the expression evaluate as with floating point arithmetic, without it - it would end up making integral division which would result with either **0** (some API limits consumed) or **1** (no API limits consumed).

```
from("direct:querySalesforce")
    .to("salesforce:limits")
    .choice()
    .when(spel("#{1.0 * body.dailyApiRequests.remaining / body.dailyApiRequests.max < 0.1}"))
        .to("salesforce:query?...")
    .otherwise()
        .setBody(constant("Used up Salesforce API limits, leaving 10% for critical routes"))
    .endChoice()
```

54.10. WORKING WITH APPROVALS

All the properties are named exactly the same as in the Salesforce REST API prefixed with **approval.**. You can set approval properties by setting **approval.PropertyName** of the Endpoint these will be used as template – meaning that any property not present in either body or header will be taken from the Endpoint configuration. Or you can set the approval template on the Endpoint by assigning **approval** property to a reference onto a bean in the Registry.

You can also provide header values using the same **approval.PropertyName** in the incoming message headers.

And finally body can contain one **AprovalRequest** or an **Iterable** of **ApprovalRequest** objects to process as a batch.

The important thing to remember is the priority of the values specified in these three mechanisms:

1. value in body takes precedence before any other
2. value in message header takes precedence before template value
3. value in template is set if no other value in header or body was given

For example to send one record for approval using values in headers use:

Given a route:

```
from("direct:example1")//
        .setHeader("approval.ContextId", simple("${body['contextId']}"))
        .setHeader("approval.NextApproverIds", simple("${body['nextApproverIds']}"))
        .to("salesforce:approval?"//
           + "approval.actionType=Submit"//
           + "&approval.comments=this is a test"//
           + "&approval.processDefinitionNameOrId=Test_Account_Process"//
           + "&approval.skipEntryCriteria=true");
```

You could send a record for approval using:

```
final Map<String, String> body = new HashMap<>();
body.put("contextId", accountIds.iterator().next());
body.put("nextApproverIds", userId);

final ApprovalResult result = template.requestBody("direct:example1", body, ApprovalResult.class);
```

54.11. USING SALESFORCE RECENT ITEMS API

To fetch the recent items use **salesforce:recent** operation. This operation returns an **java.util.List** of **org.apache.camel.component.salesforce.api.dto.RecentItem** objects (**List<RecentItem>**) that in turn contain the **Id**, **Name** and **Attributes** (with **type** and **url** properties). You can limit the number of returned items by specifying **limit** parameter set to maximum number of records to return. For example:

```
from("direct:fetchRecentItems")
    to("salesforce:recent")
        .split().body()
            .log("${body.name} at ${body.attributes.url}");
```

54.12. USING SALESFORCE COMPOSITE API TO SUBMIT SOBJECT TREE

To create up to 200 records including parent-child relationships use **salesforce:composite-tree** operation. This requires an instance of **org.apache.camel.component.salesforce.api.dto.composite.SObjectTree** in the input message and returns the same tree of objects in the output message. The **org.apache.camel.component.salesforce.api.dto.AbstractSObjectBase** instances within the tree get updated with the identifier values (**Id** property) or their corresponding **org.apache.camel.component.salesforce.api.dto.composite.SObjectNode** is populated with **errors** on failure.

Note that for some records operation can succeed and for some it can fail – so you need to manually check for errors.

Easiest way to use this functionality is to use the DTOs generated by the **camel-salesforce-maven-plugin**, but you also have the option of customizing the references that identify the each object in the tree, for instance primary keys from your database.

Lets look at an example:

```
Account account = ...
Contact president = ...
Contact marketing = ...

Account anotherAccount = ...
Contact sales = ...
Asset someAsset = ...

// build the tree
SObjectTree request = new SObjectTree();
request.addObject(account).addChildren(president, marketing);
request.addObject(anotherAccount).addChild(sales).addChild(someAsset);

final SObjectTree response = template.requestBody("salesforce:composite-tree", tree,
SObjectTree.class);
final Map<Boolean, List<SObjectNode>> result = response.allNodes()
                                        .collect(Collectors.groupingBy(SObjectNode::hasErrors));

final List<SObjectNode> withErrors = result.get(true);
final List<SObjectNode> succeeded = result.get(false);

final String firstId = succeeded.get(0).getId();
```

54.13. USING SALESFORCE COMPOSITE API TO SUBMIT MULTIPLE REQUESTS IN A BATCH

The Composite API batch operation (**composite-batch**) allows you to accumulate multiple requests in a batch and then submit them in one go, saving the round trip cost of multiple individual requests. Each response is then received in a list of responses with the order preserved, so that the n-th requests response is in the n-th place of the response.

NOTE

The results can vary from API to API so the result of the request is given as a **java.lang.Object**. In most cases the result will be a **java.util.Map** with string keys and values or other **java.util.Map** as value. Requests are made in JSON format and hold some type information (i.e. it is known what values are strings and what values are numbers).

Lets look at an example:

```
final String acountId = ...
final SObjectBatch batch = new SObjectBatch("38.0");

final Account updates = new Account();
updates.setName("NewName");
batch.addUpdate("Account", accountId, updates);

final Account newAccount = new Account();
```

```
newAccount.setName("Account created from Composite batch API");
batch.addCreate(newAccount);

batch.addGet("Account", accountId, "Name", "BillingPostalCode");

batch.addDelete("Account", accountId);

final SObjectBatchResponse response = template.requestBody("salesforce:composite-batch", batch,
SObjectBatchResponse.class);

boolean hasErrors = response.hasErrors(); // if any of the requests has resulted in either 4xx or 5xx
HTTP status
final List<SObjectBatchResult> results = response.getResults(); // results of three operations sent in
batch

final SObjectBatchResult updateResult = results.get(0); // update result
final int updateStatus = updateResult.getStatusCode(); // probably 204
final Object updateResultData = updateResult.getResult(); // probably null

final SObjectBatchResult createResult = results.get(1); // create result
@SuppressWarnings("unchecked")
final Map<String, Object> createData = (Map<String, Object>) createResult.getResult();
final String newAccountId = createData.get("id"); // id of the new account, this is for JSON, for XML it
would be createData.get("Result").get("id")

final SObjectBatchResult retrieveResult = results.get(2); // retrieve result
@SuppressWarnings("unchecked")
final Map<String, Object> retrieveData = (Map<String, Object>) retrieveResult.getResult();
final String accountName = retrieveData.get("Name"); // Name of the retrieved account, this is for
JSON, for XML it would be createData.get("Account").get("Name")
final String accountBillingPostalCode = retrieveData.get("BillingPostalCode"); // Name of the retrieved
account, this is for JSON, for XML it would be createData.get("Account").get("BillingPostalCode")

final SObjectBatchResult deleteResult = results.get(3); // delete result
final int updateStatus = deleteResult.getStatusCode(); // probably 204
final Object updateResultData = deleteResult.getResult(); // probably null
```

54.14. USING SALESFORCE COMPOSITE API TO SUBMIT MULTIPLE CHAINED REQUESTS

The **composite** operation allows submitting up to 25 requests that can be chained together, for instance identifier generated in previous request can be used in subsequent request. Individual requests and responses are linked with the provided *reference*.

NOTE

Composite API supports only JSON payloads.

NOTE

As with the batch API the results can vary from API to API so the result of the request is given as a **java.lang.Object**. In most cases the result will be a **java.util.Map** with string keys and values or other **java.util.Map** as value. Requests are made in JSON format hold some type information (i.e. it is known what values are strings and what values are numbers).

Lets look at an example:

```
SObjectComposite composite = new SObjectComposite("38.0", true);

// first insert operation via an external id
final Account updateAccount = new TestAccount();
updateAccount.setName("Salesforce");
updateAccount.setBillingStreet("Landmark @ 1 Market Street");
updateAccount.setBillingCity("San Francisco");
updateAccount.setBillingState("California");
updateAccount.setIndustry(Account_IndustryEnum.TECHNOLOGY);
composite.addUpdate("Account", "001xx000003DIpcAAG", updateAccount, "UpdatedAccount");

final Contact newContact = new TestContact();
newContact.setLastName("John Doe");
newContact.setPhone("1234567890");
composite.addCreate(newContact, "NewContact");

final AccountContactJunction__c junction = new AccountContactJunction__c();
junction.setAccount__c("001xx000003DIpcAAG");
junction.setContactId__c("@{NewContact.id}");
composite.addCreate(junction, "JunctionRecord");

final SObjectCompositeResponse response = template.requestBody("salesforce:composite",
composite, SObjectCompositeResponse.class);
final List<SObjectCompositeResult> results = response.getCompositeResponse();

final SObjectCompositeResult accountUpdateResult = results.stream().filter(r ->
"UpdatedAccount".equals(r.getReferenceId())).findFirst().get()
final int statusCode = accountUpdateResult.getHttpStatusCode(); // should be 200
final Map<String, ?> accountUpdateBody = accountUpdateResult.getBody();

final SObjectCompositeResult contactCreationResult = results.stream().filter(r ->
"JunctionRecord".equals(r.getReferenceId())).findFirst().get()
```

54.15. USING "RAW" SALESFORCE COMPOSITE

It's possible to directly call Salesforce composite by preparing the Salesforce JSON request in the route thanks to the **rawPayload** option.

For instance, you can have the following route:

```
from("timer:fire?period=2000").setBody(constant("{\n" +
    " \"allOrNone\" : true,\n" +
    " \"records\" : [ { \n" +
    "   \"attributes\" : {\"type\" : \"FOO\"},\n" +
```

```
    "   \"Name\" : \"123456789\",\n" +
    "   \"FOO\" : \"XXXX\",\n" +
    "   \"ACCOUNT\" : 2100.0\n" +
    "   \"ExternalID\" : \"EXTERNAL\"\n"
    " }]\n" +
    "}")
.to("salesforce:composite?rawPayload=true")
.log("${body}");
```

The route directly creates the body as JSON and directly submit to salesforce endpoint using **rawPayload=true** option.

With this approach, you have the complete control on the Salesforce request.

POST is the default HTTP method used to send raw Composite requests to salesforce. Use the **compositeMethod** option to override to the other supported value, **GET**, which returns a list of other available composite resources.

54.16. USING RAW OPERATION

Send HTTP requests to salesforce with full, raw control of all aspects of the call. Any serialization or deserialization of request and response bodies must be performed in the route. The **Content-Type** HTTP header will be automatically set based on the **format** option, but this can be overridden with the **rawHttpHeaders** option.

Parameter	Type	Description	Default	Required
request body	**String** or **InputStream**	Body of the HTTP request		
rawPath	**String**	The portion of the endpoint URL after the domain name, e.g., '/services/data/v51.0/sobjects/Account/'		x
rawMethod	**String**	The HTTP method		x
rawQueryParameters	**String**	Comma separated list of message headers to include as query parameters. Do not url-encode values as this will be done automatically.		
rawHttpHeaders	**String**	Comma separated list of message headers to include as HTTP headers		

54.16.1. Query example

In this example we'll send a query to the REST API. The query must be passed in a URL parameter called "q", so we'll create a message header called q and tell the raw operation to include that message header as a URL parameter:

```
from("direct:queryExample")
  .setHeader("q", "SELECT Id, LastName FROM Contact")
  .to("salesforce:raw?
format=JSON&rawMethod=GET&rawQueryParameters=q&rawPath=/services/data/v51.0/query")
  // deserialize JSON results or handle in some other way
```

54.16.2. SObject example

In this example, we'll pass a Contact the REST API in a **create** operation. Since the **raw** operation does not perform any serialization, we make sure to pass XML in the message body

```
from("direct:createAContact")
  .setBody(constant("<Contact><LastName>TestLast</LastName></Contact>"))
  .to("salesforce:raw?
format=XML&rawMethod=POST&rawPath=/services/data/v51.0/sobjects/Contact")
```

The response is:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Result>
    <id>0034x00000RnV6zAAF</id>
    <success>true</success>
</Result>
```

54.17. USING COMPOSITE SOBJECT COLLECTIONS

The SObject Collections API executes actions on multiple records in one request. Use sObject Collections to reduce the number of round-trips between the client and server. The entire request counts as a single call toward your API limits. This resource is available in API version 42.0 and later. **SObject** records (aka DTOs) supplied to these operations must be instances of subclasses of **AbstractDescribedSObjectBase**. See the Maven Plugin section for information on generating these DTO classes. These operations serialize supplied DTOs to JSON.

54.17.1. compositeRetrieveSObjectCollections

Retrieve one or more records of the same object type.

Parameter	Type	Description	Default	Required
ids	List of String or comma-separated string	A list of one or more IDs of the objects to return. All IDs must belong to the same object type.		x

Parameter	Type	Description	Default	Required
fields	List of String or comma-separated string	A list of fields to include in the response. The field names you specify must be valid, and you must have read-level permissions to each field.		x
sObjectName	String	Type of SObject, e.g. **Account**		x
sObjectClass	String	Fully-qualified class name of DTO class to use for deserializing the response.		Required if **sObjectName** parameter does not resolve to a class that exists in the package specified by the **package** option.

54.17.2. compositeCreateSObjectCollections

Add up to 200 records, returning a list of SaveSObjectResult objects. Mixed SObject types is supported.

Parameter	Type	Description	Default	Required
request body	List of **SObject**	A list of SObjects to create		x
allOrNone	boolean	Indicates whether to roll back the entire request when the creation of any object fails (true) or to continue with the independent creation of other objects in the request.	false	

54.17.3. compositeUpdateSObjectCollections

Update up to 200 records, returning a list of SaveSObjectResult objects. Mixed SObject types is supported.

Parameter	Type	Description	Default	Required
request body	List of **SObject**	A list of SObjects to update		x
allOrNone	boolean	Indicates whether to roll back the entire request when the update of any object fails (true) or to continue with the independent update of other objects in the request.	false	

54.17.4. compositeUpsertSObjectCollections

Create or update (upsert) up to 200 records based on an external ID field, returning a list of UpsertSObjectResult objects. Mixed SObject types is not supported.

Parameter	Type	Description	Default	Required
request body	List of **SObject**	A list of SObjects to upsert		x
allOrNone	boolean	Indicates whether to roll back the entire request when the upsert of any object fails (true) or to continue with the independent upsert of other objects in the request.	false	
sObjectName	String	Type of SObject, e.g. **Account**		x
sObjectIdName	String	Name of External ID field		x

54.17.5. compositeDeleteSObjectCollections

Delete up to 200 records, returning a list of DeleteSObjectResult objects. Mixed SObject types is supported.

Parameter	Type	Description	Default	Required
sObjectIds or request body	List of String or comma-separated string	A list of up to 200 IDs of objects to be deleted.		x

Parameter	Type	Description	Default	Required
allOrNone	boolean	Indicates whether to roll back the entire request when the deletion of any object fails (true) or to continue with the independent deletion of other objects in the request.	false	

54.18. SENDING NULL VALUES TO SALESFORCE

By default, SObject fields with null values are not sent to salesforce. In order to send null values to salesforce, use the **fieldsToNull** property, as follows:

```
accountSObject.getFieldsToNull().add("Site");
```

54.19. GENERATING SOQL QUERY STRINGS

org.apache.camel.component.salesforce.api.utils.QueryHelper contains helper methods to generate SOQL queries. For instance to fetch all custom fields from *Account* SObject you can simply generate the SOQL SELECT by invoking:

```
String allCustomFieldsQuery = QueryHelper.queryToFetchFilteredFieldsOf(new Account(),
SObjectField::isCustom);
```

54.20. CAMEL SALESFORCE MAVEN PLUGIN

This Maven plugin generates DTOs for the Camel.

For obvious security reasons it is recommended that the clientId, clientSecret, userName and password fields be not set in the pom.xml. The plugin should be configured for the rest of the properties, and can be executed using the following command:

```
mvn camel-salesforce:generate -DcamelSalesforce.clientId=<clientid> -
DcamelSalesforce.clientSecret=<clientsecret> \
    -DcamelSalesforce.userName=<username> -DcamelSalesforce.password=<password>
```

The generated DTOs use Jackson annotations. All Salesforce field types are supported. Date and time fields are mapped to **java.time.ZonedDateTime** by default, and picklist fields are mapped to generated Java Enumerations.

Please refer to README.md for details on how to generate the DTO.

54.21. SPRING BOOT AUTO-CONFIGURATION

When using salesforce with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
```

```
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-salesforce-starter</artifactId>
</dependency>
```

The component supports 91 options, which are listed below.

Name	Description	Default	Type
camel.component.salesforce.all-or-none	Composite API option to indicate to rollback all records if any are not successful.	false	Boolean
camel.component.salesforce.apex-method	APEX method name.		String
camel.component.salesforce.apex-query-params	Query params for APEX method.		Map
camel.component.salesforce.apex-url	APEX method URL.		String
camel.component.salesforce.api-version	Salesforce API version.	53.0	String
camel.component.salesforce.authentication-type	Explicit authentication method to be used, one of USERNAME_PASSWORD, REFRESH_TOKEN or JWT. Salesforce component can auto-determine the authentication method to use from the properties set, set this property to eliminate any ambiguity.		AuthenticationType
camel.component.salesforce.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.salesforce.backoff-increment	Backoff interval increment for Streaming connection restart attempts for failures beyond CometD auto-reconnect. The option is a long type.	1000	Long
camel.component.salesforce.batch-id	Bulk API Batch ID.		String

Name	Description	Default	Type
camel.component.salesforce.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.salesforce.client-id	OAuth Consumer Key of the connected app configured in the Salesforce instance setup. Typically a connected app needs to be configured but one can be provided by installing a package.		String
camel.component.salesforce.client-secret	OAuth Consumer Secret of the connected app configured in the Salesforce instance setup.		String
camel.component.salesforce.composite-method	Composite (raw) method.		String
camel.component.salesforce.config	Global endpoint configuration - use to set values that are common to all endpoints. The option is a org.apache.camel.component.salesforce.SalesforceEndpointConfig type.		SalesforceEndpointConfig
camel.component.salesforce.content-type	Bulk API content type, one of XML, CSV, ZIP_XML, ZIP_CSV.		ContentType
camel.component.salesforce.default-replay-id	Default replayId setting if no value is found in initialReplayIdMap.	-1	Long
camel.component.salesforce.enabled	Whether to enable auto configuration of the salesforce component. This is enabled by default.		Boolean
camel.component.salesforce.fall-back-replay-id	ReplayId to fall back to after an Invalid Replay Id response.	-1	Long
camel.component.salesforce.format	Payload format to use for Salesforce API calls, either JSON or XML, defaults to JSON. As of Camel 3.12, this option only applies to the Raw operation.		PayloadFormat

Name	Description	Default	Type
camel.component.salesforce.http-client	Custom Jetty Http Client to use to connect to Salesforce. The option is a org.apache.camel.component.salesforce.SalesforceHttpClient type.		SalesforceHttpClient
camel.component.salesforce.http-client-connection-timeout	Connection timeout used by the HttpClient when connecting to the Salesforce server.	60000	Long
camel.component.salesforce.http-client-idle-timeout	Timeout used by the HttpClient when waiting for response from the Salesforce server.	10000	Long
camel.component.salesforce.http-client-properties	Used to set any properties that can be configured on the underlying HTTP client. Have a look at properties of SalesforceHttpClient and the Jetty HttpClient for all available options.		Map
camel.component.salesforce.http-max-content-length	Max content length of an HTTP response.		Integer
camel.component.salesforce.http-proxy-auth-uri	Used in authentication against the HTTP proxy server, needs to match the URI of the proxy server in order for the httpProxyUsername and httpProxyPassword to be used for authentication.		String
camel.component.salesforce.http-proxy-excluded-addresses	A list of addresses for which HTTP proxy server should not be used.		Set
camel.component.salesforce.http-proxy-host	Hostname of the HTTP proxy server to use.		String
camel.component.salesforce.http-proxy-included-addresses	A list of addresses for which HTTP proxy server should be used.		Set
camel.component.salesforce.http-proxy-password	Password to use to authenticate against the HTTP proxy server.		String

Name	Description	Default	Type
camel.component.salesforce.http-proxy-port	Port number of the HTTP proxy server to use.		Integer
camel.component.salesforce.http-proxy-realm	Realm of the proxy server, used in preemptive Basic/Digest authentication methods against the HTTP proxy server.		String
camel.component.salesforce.http-proxy-secure	If set to false disables the use of TLS when accessing the HTTP proxy.	true	Boolean
camel.component.salesforce.http-proxy-socks4	If set to true the configures the HTTP proxy to use as a SOCKS4 proxy.	false	Boolean
camel.component.salesforce.http-proxy-use-digest-auth	If set to true Digest authentication will be used when authenticating to the HTTP proxy, otherwise Basic authorization method will be used.	false	Boolean
camel.component.salesforce.http-proxy-username	Username to use to authenticate against the HTTP proxy server.		String
camel.component.salesforce.http-request-buffer-size	HTTP request buffer size. May need to be increased for large SOQL queries.	8192	Integer
camel.component.salesforce.include-details	Include details in Salesforce1 Analytics report, defaults to false.		Boolean
camel.component.salesforce.initial-replay-id-map	Replay IDs to start from per channel name.		Map
camel.component.salesforce.instance-id	Salesforce1 Analytics report execution instance ID.		String
camel.component.salesforce.instance-url	URL of the Salesforce instance used after authentication, by default received from Salesforce on successful authentication.		String

Name	Description	Default	Type
camel.component.salesforce.job-id	Bulk API Job ID.		String
camel.component.salesforce.jwt-audience	Value to use for the Audience claim (aud) when using OAuth JWT flow. If not set, the login URL will be used, which is appropriate in most cases.		String
camel.component.salesforce.keystore	KeyStore parameters to use in OAuth JWT flow. The KeyStore should contain only one entry with private key and certificate. Salesforce does not verify the certificate chain, so this can easily be a selfsigned certificate. Make sure that you upload the certificate to the corresponding connected app. The option is a org.apache.camel.support.jsse.KeyStoreParameters type.		KeyStoreParameters
camel.component.salesforce.lazy-login	If set to true prevents the component from authenticating to Salesforce with the start of the component. You would generally set this to the (default) false and authenticate early and be immediately aware of any authentication issues.	false	Boolean
camel.component.salesforce.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.salesforce.limit	Limit on number of returned records. Applicable to some of the API, check the Salesforce documentation.		Integer
camel.component.salesforce.locator	Locator provided by salesforce Bulk 2.0 API for use in getting results for a Query job.		String
camel.component.salesforce.login-config	All authentication configuration in one nested bean, all properties set there can be set directly on the component as well. The option is a org.apache.camel.component.salesforce.SalesforceLoginConfig type.		SalesforceLoginConfig

Name	Description	Default	Type
camel.component.salesforce.login-url	URL of the Salesforce instance used for authentication, by default set to .		String
camel.component.salesforce.long-polling-transport-properties	Used to set any properties that can be configured on the LongPollingTransport used by the BayeuxClient (CometD) used by the streaming api.		Map
camel.component.salesforce.max-backoff	Maximum backoff interval for Streaming connection restart attempts for failures beyond CometD auto-reconnect. The option is a long type.	30000	Long
camel.component.salesforce.max-records	The maximum number of records to retrieve per set of results for a Bulk 2.0 Query. The request is still subject to the size limits. If you are working with a very large number of query results, you may experience a timeout before receiving all the data from Salesforce. To prevent a timeout, specify the maximum number of records your client is expecting to receive in the maxRecords parameter. This splits the results into smaller sets with this value as the maximum size.		Integer
camel.component.salesforce.not-found-behaviour	Sets the behaviour of 404 not found status received from Salesforce API. Should the body be set to NULL NotFoundBehaviour#NULL or should a exception be signaled on the exchange NotFoundBehaviour#EXCEPTION - the default.		NotFoundBehaviour
camel.component.salesforce.notify-for-fields	Notify for fields, options are ALL, REFERENCED, SELECT, WHERE.		NotifyForFieldsEnum
camel.component.salesforce.notify-for-operation-create	Notify for create operation, defaults to false (API version = 29.0).		Boolean
camel.component.salesforce.notify-for-operation-delete	Notify for delete operation, defaults to false (API version = 29.0).		Boolean
camel.component.salesforce.notify-for-operation-undelete	Notify for un-delete operation, defaults to false (API version = 29.0).		Boolean

Name	Description	Default	Type
camel.component.salesforce.notify-for-operation-update	Notify for update operation, defaults to false (API version = 29.0).		Boolean
camel.component.salesforce.notify-for-operations	Notify for operations, options are ALL, CREATE, EXTENDED, UPDATE (API version 29.0).		NotifyForOperationsEnum
camel.component.salesforce.object-mapper	Custom Jackson ObjectMapper to use when serializing/deserializing Salesforce objects. The option is a com.fasterxml.jackson.databind.ObjectMapper type.		ObjectMapper
camel.component.salesforce.packages	In what packages are the generated DTO classes. Typically the classes would be generated using camel-salesforce-maven-plugin. Set it if using the generated DTOs to gain the benefit of using short SObject names in parameters/header values. Multiple packages can be separated by comma.		String
camel.component.salesforce.password	Password used in OAuth flow to gain access to access token. It's easy to get started with password OAuth flow, but in general one should avoid it as it is deemed less secure than other flows. Make sure that you append security token to the end of the password if using one.		String
camel.component.salesforce.pk-chunking	Use PK Chunking. Only for use in original Bulk API. Bulk 2.0 API performs PK chunking automatically, if necessary.		Boolean
camel.component.salesforce.pk-chunking-chunk-size	Chunk size for use with PK Chunking. If unspecified, salesforce default is 100,000. Maximum size is 250,000.		Integer
camel.component.salesforce.pk-chunking-parent	Specifies the parent object when you're enabling PK chunking for queries on sharing objects. The chunks are based on the parent object's records rather than the sharing object's records. For example, when querying on AccountShare, specify Account as the parent object. PK chunking is supported for sharing objects as long as the parent object is supported.		String

Name	Description	Default	Type
camel.component.salesforce.pk-chunking-start-row	Specifies the 15-character or 18-character record ID to be used as the lower boundary for the first chunk. Use this parameter to specify a starting ID when restarting a job that failed between batches.		String
camel.component.salesforce.query-locator	Query Locator provided by salesforce for use when a query results in more records than can be retrieved in a single call. Use this value in a subsequent call to retrieve additional records.		String
camel.component.salesforce.raw-http-headers	Comma separated list of message headers to include as HTTP parameters for Raw operation.		String
camel.component.salesforce.raw-method	HTTP method to use for the Raw operation.		String
camel.component.salesforce.raw-path	The portion of the endpoint URL after the domain name. E.g., '/services/data/v52.0/sobjects/Account/'.		String
camel.component.salesforce.raw-payload	Use raw payload String for request and response (either JSON or XML depending on format), instead of DTOs, false by default.	false	Boolean
camel.component.salesforce.raw-query-parameters	Comma separated list of message headers to include as query parameters for Raw operation. Do not url-encode values as this will be done automatically.		String
camel.component.salesforce.refresh-token	Refresh token already obtained in the refresh token OAuth flow. One needs to setup a web application and configure a callback URL to receive the refresh token, or configure using the builtin callback at and then retrive the refresh_token from the URL at the end of the flow. Note that in development organizations Salesforce allows hosting the callback web application at localhost.		String
camel.component.salesforce.report-id	Salesforce1 Analytics report Id.		String

Name	Description	Default	Type
camel.component.salesforce.report-metadata	Salesforce1 Analytics report metadata for filtering. The option is a org.apache.camel.component.salesforce.api.dto.analytics.reports.ReportMetadata type.		ReportMetadata
camel.component.salesforce.result-id	Bulk API Result ID.		String
camel.component.salesforce.s-object-blob-field-name	SObject blob field name.		String
camel.component.salesforce.s-object-class	Fully qualified SObject class name, usually generated using camel-salesforce-maven-plugin.		String
camel.component.salesforce.s-object-fields	SObject fields to retrieve.		String
camel.component.salesforce.s-object-id	SObject ID if required by API.		String
camel.component.salesforce.s-object-id-name	SObject external ID field name.		String
camel.component.salesforce.s-object-id-value	SObject external ID field value.		String
camel.component.salesforce.s-object-name	SObject name if required or supported by API.		String
camel.component.salesforce.s-object-query	Salesforce SOQL query string.		String
camel.component.salesforce.s-object-search	Salesforce SOSL search string.		String

Name	Description	Default	Type
camel.component.salesforce.ssl-context-parameters	SSL parameters to use, see SSLContextParameters class for all available options. The option is a org.apache.camel.support.jsse.SSLContextParameters type.		SSLContextParameters
camel.component.salesforce.update-topic	Whether to update an existing Push Topic when using the Streaming API, defaults to false.	false	Boolean
camel.component.salesforce.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean
camel.component.salesforce.user-name	Username used in OAuth flow to gain access to access token. It's easy to get started with password OAuth flow, but in general one should avoid it as it is deemed less secure than other flows.		String
camel.component.salesforce.worker-pool-max-size	Maximum size of the thread pool used to handle HTTP responses.	20	Integer
camel.component.salesforce.worker-pool-size	Size of the thread pool used to handle HTTP responses.	10	Integer

CHAPTER 55. SAP COMPONENT

The SAP component is a package consisting of ten different SAP components. There are remote function call (RFC) components that support the sRFC, tRFC, and qRFC protocols and there are IDoc components that facilitate communication using messages in IDoc format. The component uses the SAP Java Connector (SAP JCo) library to facilitate bidirectional communication with SAP and the SAP IDoc library to transmit the documents in the Intermediate Document (IDoc) format.

55.1. DEPENDENCIES

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.fusesource</groupId>
  <artifactId>camel-sap-starter</artifactId>
  <version>3.20.1.redhat-00047</version>
</dependency>
```

55.1.1. Additional platform restrictions for the SAP component

Because the SAP component depends on the third-party JCo 3 and IDoc 3 libraries, it can only be installed on the platforms that these libraries support.

55.1.2. SAP JCo and SAP IDoc libraries

A prerequisite for using the SAP component is that the SAP Java Connector (SAP JCo) libraries and the SAP IDoc library are installed into the **lib**/ directory of the Java runtime. You must make sure that you download the appropriate set of SAP libraries for your target operating system from the SAP Service Marketplace.

The names of the library files vary depending on the target operating system, as shown below.

Table 55.1. Required SAP Libraries

SAP Component	Linux and UNIX	Windows
SAP JCo 3	**sapjco3.jar** **libsapjco3.so**	**sapjco3.jar** **sapjco3.dll**
SAP IDoc	**sapidoc3.jar**	**sapidoc3.jar**

55.2. URI FORMAT

There are two different kinds of endpoint provided by the SAP component: the Remote Function Call (RFC) endpoints, and the Intermediate Document (IDoc) endpoints.

The URI formats for the RFC endpoints are as follows:

```
sap-srfc-destination:destinationName:rfcName
sap-trfc-destination:destinationName:rfcName
sap-qrfc-destination:destinationName:queueName:rfcName
```

```
sap-srfc-server:serverName:rfcName[?options]
sap-trfc-server:serverName:rfcName[?options]
```

The URI formats for the IDoc endpoints are as follows:

```
sap-idoc-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
sap-idoclist-
destination:destinationName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
sap-qidoc-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applicationR
elease]]]
sap-qidoclist-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applicationR
elease]]]
sap-idoclist-server:serverName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
[?options]
```

The URI formats prefixed by sap-*endpointKind*-destination are used to define destination endpoints (in other words, Camel producer endpoints) and *destinationName* is the name of a specific outbound connection to an SAP instance. Outbound connections are named and configured at the component level.

The URI formats prefixed by sap-*endpointKind*-server are used to define server endpoints (in other words, Camel consumer endpoints) and *serverName* is the name of a specific inbound connection from an SAP instance. Inbound connections are named and configured at the component level.

The other components of an RFC endpoint URI are as follows:

rfcName

(Required) In a destination endpoint URI, is the name of the RFC invoked by the endpoint in the connected SAP instance. In a server endpoint URI, is the name of the RFC handled by the endpoint when invoked from the connected SAP instance.

queueName

Specifies the queue this endpoint sends an SAP request to.

The other components of an IDoc endpoint URI are as follows:

idocType

(Required) Specifies the Basic IDoc Type of an IDoc produced by this endpoint.

idocTypeExtension

Specifies the IDoc Type Extension, if any, of an IDoc produced by this endpoint.

systemRelease

Specifies the associated SAP Basis Release, if any, of an IDoc produced by this endpoint.

applicationRelease

Specifies the associated Application Release, if any, of an IDoc produced by this endpoint.

queueName

Specifies the queue this endpoint sends an SAP request to.

55.2.1. Options for RFC destination endpoints

The RFC destination endpoints (**sap-srfc-destination**, **sap-trfc-destination**, and **sap-qrfc-destination**) support the following URI options:

Name	Default	Description
stateful	**false**	If **true**, specifies that this endpoint initiates an SAP stateful session
transacted	**false**	If **true**, specifies that this endpoint initiates an SAP transaction

55.2.2. Options for RFC server endpoints

The SAP RFC server endpoints (**sap-srfc-server** and **sap-trfc-server**) support the following URI options:

Name	Default	Description
stateful	**false**	If **true**, specifies that this endpoint initiates an SAP stateful session.
propagateExceptions	**false**	**(sap-trfc-server endpoint only)** If **true**, specifies that this endpoint propagates exceptions back to the caller in SAP, instead of the exchange's exception handler.

55.2.3. Options for the IDoc List Server endpoint

The SAP IDoc List Server endpoint (**sap-idoclist-server**) supports the following URI options:

Name	Default	Description
stateful	**false**	If **true**, specifies that this endpoint initiates an SAP stateful session.
propagateExceptions	**false**	If **true**, specifies that this endpoint propagates exceptions back to the caller in SAP, instead of the exchange's exception handler.

55.2.4. Summary of the RFC and IDoc endpoints

The SAP component package provides the following RFC and IDoc endpoints:

sap-srfc-destination

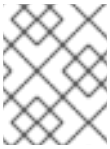
Camel SAP Synchronous Remote Function Call Destination Camel component. This endpoint should be used in cases where Camel routes require synchronous delivery of requests to and responses from an SAP system.

NOTE

The sRFC protocol used by this component delivers requests and responses to and from an SAP system with **best effort**. In case of a communication error while sending a request, the completion status of a remote function call in the receiving SAP system remains **in doubt**.

sap-trfc-destination

Camel SAP Transactional Remote Function Call Destination Camel component. This endpoint should be used in cases where requests must be delivered to the receiving SAP system **at most once**. To accomplish this, the component generates a transaction ID, **tid**, which accompanies every request sent through the component in a route's exchange. The receiving SAP system records the **tid** accompanying a request before delivering the request; if the SAP system receives the request again with the same **tid** it will not deliver the request. Thus if a route encounters a communication error when sending a request through an endpoint of this component, it can retry sending the request within the same exchange knowing it will be delivered and executed only once.



NOTE

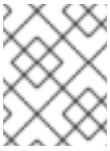
The tRFC protocol used by this component is asynchronous and does not return a response. Thus the endpoints of this component do not return a response message.

NOTE

This component does not guarantee the order of a series of requests through its endpoints, and the delivery and execution order of these requests may differ on the receiving SAP system due to communication errors and resends of a request. For guaranteed delivery order, please see the Camel SAP Queued Remote Function Call Destination Camel component.

sap-qrfc-destination

Camel SAP Queued Remote Function Call Destination Camel component. This component extends the capabilities of the Transactional Remote Function Call Destination camel component by adding **in order** delivery guarantees to the delivery of requests through its endpoints. This endpoint should be used in cases where a series of requests depend on each other and must be delivered to the receiving SAP system **at most once** and **in order**. The component accomplishes the **at most once** delivery guarantees using the same mechanisms as the Camel SAP Transactional Remote Function Call Destination Camel component. The ordering guarantee is accomplished by serializing the requests in the order they are received by the SAP system to an *inbound queue*. Inbound queues are processed by the *QIN scheduler* within SAP. When the inbound queue is **activated**, the QIN Scheduler will execute the queue requests in order.



NOTE

The qRFC protocol used by this component is asynchronous and does not return a response. Thus the endpoints of this component do not return a response message.

sap-srfc-server

Camel SAP Synchronous Remote Function Call Server Camel component. This component and its endpoints should be used in cases where a Camel route is required to synchronously handle requests from and responses to an SAP system.

sap-trfc-server

Camel SAP Transactional Remote Function Call Server Camel component. This endpoint should be used in cases where the sending SAP system requires **at most once** delivery of its requests to a Camel route. To accomplish this, the sending SAP system generates a transaction ID, **tid**, which accompanies every request it sends to the component's endpoints. The sending SAP system will first check with the component whether a given **tid** has been received by it before sending a series of requests associated with the **tid**. The component will check the list of received **tid**s it maintains, record the sent **tid** if it is not in that list, and then respond to the sending SAP system, indicating whether or not the **tid** has already been recorded. The sending SAP system will only then send the series of requests, if the **tid** has not been previously recorded. This enables a sending SAP system to reliably send a series of requests once to a camel route.

sap-idoc-destination

Camel SAP IDoc Destination Camel component. This endpoint should be used in cases where a Camel route sends a list of Intermediate Documents (IDocs) to an SAP system.

sap-idoclist-destination

Camel SAP IDoc List Destination Camel component. This endpoint should be used in cases where a Camel route sends a list of Intermediate documents (IDocs) list to an SAP system.

sap-qidoc-destination

Camel SAP Queued IDoc Destination Camel component. This component and its endpoints should be used in cases where a Camel route is required to send a list of Intermediate documents (IDocs) to an SAP system in order.

sap-qidoclist-destination

Camel SAP Queued IDoc List Destination Camel component. This component and its endpoints are used in cases where a camel route sends the Intermediate documents (IDocs) list to an SAP system in order.

sap-idoclist-server

Camel SAP IDoc List Server Camel component. This endpoint should be used in cases where a sending SAP system requires delivery of Intermediate Document lists to a Camel route. This component uses the tRFC protocol to communicate with SAP as described in the **sap-trfc-server-standalone** quick start.

55.2.5. SAP RFC destination endpoint

An RFC destination endpoint supports outbound communication to SAP, which enable these endpoints to make RFC calls out to ABAP function modules in SAP. An RFC destination endpoint is configured to make an RFC call to a specific ABAP function over a specific connection to an SAP instance. An RFC destination is a logical designation for an outbound connection and has a unique name. An RFC destination is specified by a set of connection parameters called *destination data*.

An RFC destination endpoint will extract an RFC request from the input message of the IN-OUT exchanges it receives and dispatch that request in a function call to SAP. The response from the function call will be returned in the output message of the exchange. Since SAP RFC destination endpoints only

support outbound communication, an RFC destination endpoint only supports the creation of producers.

55.2.6. SAP RFC server endpoint

An RFC server endpoint supports inbound communication from SAP, which enables ABAP applications in SAP to make RFC calls into server endpoints. An ABAP application interacts with an RFC server endpoint as if it were a remote function module. An RFC server endpoint is configured to receive an RFC call to a specific RFC function over a specific connection from an SAP instance. An RFC server is a logical designation for an inbound connection and has a unique name. An RFC server is specified by a set of connection parameters called *server data*.

An RFC server endpoint will handle an incoming RFC request and dispatch it as the input message of an IN-OUT exchange. The output message of the exchange will be returned as the response of the RFC call. Since SAP RFC server endpoints only support inbound communication, an RFC server endpoint only supports the creation of consumers.

55.2.7. SAP IDoc and IDoc list destination endpoints

An IDoc destination endpoint supports outbound communication to SAP, which can then perform further processing on the IDoc message. An IDoc document represents a business transaction, which can easily be exchanged with non-SAP systems. An IDoc destination is specified by a set of connection parameters called *destination data*.

An IDoc list destination endpoint is similar to an IDoc destination endpoint, except that the messages it handles consist of a **list** of IDoc documents.

55.2.8. SAP IDoc list server endpoint

An IDoc list server endpoint supports inbound communication from SAP, enabling a Camel route to receive a list of IDoc documents from an SAP system. An IDoc list server is specified by a set of connection parameters called *server data*.

55.2.9. Metadata repositories

A metadata repository is used to store the following kinds of metadata:

Interface descriptions of function modules

This metadata is used by the JCo and ABAP runtimes to check RFC calls to ensure the type-safe transfer of data between communication partners before dispatching those calls. A repository is populated with repository data. Repository data is a map of named function templates. A function template contains the metadata describing all the parameters and their typing information passed to and from a function module and has the unique name of the function module it describes.

IDoc type descriptions

This metadata is used by the IDoc runtime to ensure that the IDoc documents are correctly formatted before being sent to a communication partner. A basic IDoc type consists of a name, a list of permitted segments, and a description of the hierarchical relationship between the segments. Some additional constraints can be imposed on the segments: a segment can be mandatory or optional; and it is possible to specify a minimum/maximum range for each segment (defining the number of allowed repetitions of that segment).

SAP destination and server endpoints thus require access to a repository, in order to send and receive RFC calls and in order to send and receive IDoc documents. For RFC calls, the metadata for all function modules invoked and handled by the endpoints must reside within the repository; and for IDoc

endpoints, the metadata for all IDoc types and IDoc type extensions handled by the endpoints must reside within the repository. The location of the repository used by a destination and server endpoint is specified in the destination data and the server data of their respective connections.

In the case of an SAP destination endpoint, the repository it uses typically resides in an SAP system and it defaults to the SAP system it is connected to. This default requires no explicit configuration in the destination data. Furthermore, the metadata for the remote function call that a destination endpoint makes will already exist in a repository for any existing function module that it calls. The metadata for calls made by destination endpoints thus require no configuration in the SAP component.

On the other hand, the metadata for function calls handled by server endpoints do not typically reside in the repository of an SAP system and must instead be provided by a repository residing in the SAP component. The SAP component maintains a map of named metadata repositories. The name of a repository corresponds to the name of the server to which it provides metadata.

55.3. CONFIGURATION

The SAP component maintains three maps to store destination data, server data, and repository data. The *destination data store* and the *server data store* are configured on a special configuration object, **SapConnectionConfiguration**, which automatically gets injected into the SAP component (in the context of Blueprint XML configuration or Spring XML configuration files). The *repository data store* must be configured directly on the relevant SAP component.

55.3.1. Configuration Overview

The SAP component maintains three maps to store destination data, server data, and repository data. The component's property, **destinationDataStore**, stores destination data keyed by destination name, the property, **serverDataStore**, stores server data keyed by server name and the property, **repositoryDataStore**, stores repository data keyed by repository name. These configurations must be passed to the component during its initialization.

Example

The following example shows how to configure a sample destination data store and a sample server data store in a Blueprint XML file. The **sap-configuration** bean (of type **SapConnectionConfiguration**) will automatically be injected into any SAP component that is used in this XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>
```

```
    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

55.3.2. Destination Configuration

The configurations for destinations are maintained in the **destinationDataStore** property of the SAP component. Each entry in this map configures a distinct outbound connection to an SAP instance. The key for each entry is the name of the outbound connection and is used in the *destinationName* component of a destination endpoint URI as described in the URI format section.

The value for each entry is a destination data configuration object - **org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl** - that specifies the configuration of an outbound SAP connection.

Sample destination configuration

The following Blueprint XML code shows how to configure a sample destination with the name, **quickstartDest**.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Create interceptor to support tRFC processing -->
    <bean id="currentProcessorDefinitionInterceptor"
          class="org.fusesource.camel.component.sap.CurrentProcessorDefinitionInterceptStrategy" />

    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
```

```
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="password" />
        <property name="lang" value="en" />
    </bean>

</blueprint>
```

For example, after configuring the destination as shown in the preceding Blueprint XML file, you could invoke the **BAPI_FLCUST_GETLIST** remote function call on the **quickstartDest** destination using the following URI:

```
sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST
```

55.3.2.1. Interceptor for tRFC and qRFC destinations

The preceding sample destination configuration shows the instantiation of a **CurrentProcessorDefinitionInterceptStrategy** object. This object installs an interceptor in the Camel runtime, which enables the Camel SAP component to keep track of its position within a Camel route while it is handling RFC transactions.

IMPORTANT

This interceptor is critically important for transactional RFC destination endpoints (such as **sap-trfc-destination** and **sap-qrfc-destination**) and must be installed in the Camel runtime for outbound transactional RFC communication to be properly managed. The Destination RFC Transaction Handlers issues warnings into the Camel log if the strategy is not found at runtime. In this situation the Camel runtime will need to be re-provisioned and restarted to properly manage outbound transactional RFC communication.

55.3.2.2. Log on and authentication options

The following table lists the **log on and authentication** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description
client		SAP client, mandatory log on parameter.
user		log on user, log on parameter for password based authentication.

aliasUser		log on user alias, can be used instead of log on user.
userId		User identity used for log on to the ABAP AS. Used by the JCo runtime, if the destination configuration uses SSO/assertion ticket, certificate, current user ,or SNC environment for authentication. The user ID is mandatory, if neither user nor user alias is set. This ID will never be sent to the SAP backend, it will be used by the JCo runtime locally.
passwd		log on password, log on parameter for password based authentication.
lang		log on language, if not defined, the default user language is used.
mysapsso2		Use the specified SAP Cookie Version 2 as a log on ticket for SSO based authentication.
x509cert		Use the specified X509 certificate for certificate based authentication.
lcheck		Postpone the authentication until the first call - 1 (enable). Used in special cases only.
useSapGui		Use a visible, hidden, or do not use SAP GUI
codePage		Additional log on parameter to define the codepage used to convert the log on parameters. Used in special cases only.
getsso2		Order an SSO ticket after log on, the obtained ticket is available in the destination attributes.
denyInitialPassword		If set to **1**, using initial passwords will lead to an exception (default is **0**).

55.3.2.3. Connection options

The following table lists the **connection** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description
saprouter		SAP Router string for connection to systems behind a SAP Router. SAP Router string contains the chain of SAP Routers and theirs port numbers and has the form: **(/H/<host>[/S/<port>])+**.
sysnr		System number of the SAP ABAP application server, mandatory for a direct connection.
ashost		SAP ABAP application server, mandatory for a direct connection.
mshost		SAP message server, mandatory property for a load balancing connection.
msserv		SAP message server port, optional property for a load balancing connection. In order to resolve the service names sapmsXXX a lookup in **etc/services** is performed by the network layer of the operating system. If using port numbers instead of symbolic service names, no lookups are performed and no additional entries are needed.
gwhost		Allows specifying a concrete gateway, which should be used for establishing the connection to an application server. If not specified, the gateway on the application server is used.

gwserv		Should be set, when using gwhost. Allows specifying the port used on that gateway. If not specified, the port of the gateway on the application server is used. In order to resolve the service names sapgwXXX a lookup in etc/services is performed by the network layer of the operating system. If using port numbers instead of symbolic service names, no lookups are performed and no additional entries are needed.
r3name		System ID of the SAP system, mandatory property for a load balancing connection.
group		Group of SAP application servers, mandatory property for a load balancing connection.
network	**LAN**	Set this value depending on the network quality between JCo and your target system to optimize performance. The valid values are **LAN** or **WAN** (which is relevant for fast serialization only). If you set the **network** configuration option to **WAN**, a slower but more efficient compression algorithm is used and the data is analyzed for further compression options. If you set the **network** configuration to **LAN** a very fast compression algorithm is used and data analysis is performed only at a very basic level. When you set the **LAN** option, the compression ratio is not as efficient but the network transfer time is considered to be less significant. The default setting is **LAN**.
serializationFormat	**rowBased**	The valid values are **rowBased** or **columnBased**. For fast serialization **columnBased** must be set. The default serialization setting is **rowBased**.

55.3.2.4. Connection pool options

The following table lists the **connection pool** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description
peakLimit	**0**	Maximum number of active outbound connections that can be created for a destination simultaneously. A value of **0** allows an unlimited number of active connections. Otherwise, if the value is less than the value of **jpoolCapacity**, it will be automatically increased to this value. Default setting is the value of **poolCapacity**, or in case of **poolCapacity** not being specified as well, the default is **0** (unlimited).
poolCapacity	**1**	Maximum number of idle outbound connections kept open by the destination. A value of **0** has the effect that there is no connection pooling (default is **1**).
expirationTime		Time in milliseconds after which a free connection held internally by the destination can be closed.
expirationPeriod		Period in milliseconds after which the destination checks the released connections for expiration.
maxGetTime		Maximum time in milliseconds to wait for a connection, if the maximum allowed number of connections has already been allocated by the application.

55.3.2.5. Secure network connection options

The following table lists the **secure network** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description

sncMode		Secure network connection (SNC) mode, **0** (off) or **1** (on).
sncPartnername		SNC partner, for example: **p:CN=R3, O=XYZ-INC, C=EN**.
sncQop		SNC level of security: **1** to **9**.
sncMyname		Own SNC name. Overrides the environment settings.
sncLibrary		Path to the library that provides the SNC service.

55.3.2.6. Repository options

The following table lists the **repository** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description
repositoryDest		Specifies the destination which is used as a repository.
repositoryUser		If a repository destination is not set, and this property is set, it is used as user for repository calls. This enables you to use a different user for repository lookups.
repositoryPasswd		The password for a repository user. Mandatory, if a repository user is used.
repositorySnc		**(Optional)** If SNC is used for this destination, it is possible to turn it off for repository connections, if this property is set to **0**. Default setting is the value of **jco.client.snc**_mode. For special cases only.

repositoryRoundtripOptimization		Enable the **RFC_METADATA_GET** API, which provides the repository data in one single round trip. **1** Activates use of **RFC_METADATA_GET** in ABAP System. **0** Deactivates **RFC_METADATA_GET** in ABAP System. If the property is not set, the destination initially does a remote call to check whether **RFC_METADATA_GET** is available. If it is available, the destination will use it. **Note:** If the repository is already initialized (for example, because it is used by some other destination), this property does not have any effect. Generally, this property is related to the ABAP System, and should have the same value on all destinations pointing to the same ABAP System. See note 1456826 for backend prerequisites.

55.3.2.7. Trace configuration options

The following table lists the **trace configuration** options for configuring a destination in the SAP destination data store:

Name	Default Value	Description
trace		Enable/disable RFC trace (**0** or **1**).
cpicTrace		Enable/disable CPIC trace **[0..3]**.

55.3.3. Server Configuration

The configurations for servers are maintained in the **serverDataStore** property of the SAP component. Each entry in this map configures a distinct inbound connection from an SAP instance. The key for each entry is the name of the outbound connection and is used in the **serverName** component of a server

endpoint URI as described in the URI format section.

The value for each entry is a *server data configuration object*, **org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl**, which defines the configuration of an inbound SAP connection.

Sample server configuration

The following Blueprint XML code shows how to create a sample server configuration with the name, **quickstartServer**.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the Inbound and Outbound SAP Connections -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="destinationDataStore">
            <map>
                <entry key="quickstartDest" value-ref="quickstartDestinationData" />
            </map>
        </property>
        <property name="serverDataStore">
            <map>
                <entry key="quickstartServer" value-ref="quickstartServerData" />
            </map>
        </property>
    </bean>

    <!-- Configures an Outbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment *** -->
    <bean id="quickstartDestinationData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.DestinationDataImpl">
        <property name="ashost" value="example.com" />
        <property name="sysnr" value="00" />
        <property name="client" value="000" />
        <property name="user" value="username" />
        <property name="passwd" value="passowrd" />
        <property name="lang" value="en" />
    </bean>

    <!-- Configures an Inbound SAP Connection -->
    <!-- *** Please enter the connection property values for your environment ** -->
    <bean id="quickstartServerData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.ServerDataImpl">
        <property name="gwhost" value="example.com" />
        <property name="gwserv" value="3300" />
        <!-- The following property values should not be changed -->
        <property name="progid" value="QUICKSTART" />
        <property name="repositoryDestination" value="quickstartDest" />
        <property name="connectionCount" value="2" />
    </bean>
</blueprint>
```

Notice how this example also configures a destination connection, **quickstartDest**, which the server uses to retrieve metadata from a remote SAP instance. This destination is configured in the server data

through the **repositoryDestination** option. If you do not configure this option, you must create a local metadata repository instead.

For example, after configuring the destination as shown in the preceding Blueprint XML file, you could handle the **BAPI_FLCUST_GETLIST** remote function call from an invoking client, using the following URI:

```
sap-srfc-server:quickstartServer:BAPI_FLCUST_GETLIST
```

55.3.3.1. Required options

The required options for the server data configuration object are, as follows:

Name	Default Value	Description
gwhost		Gateway host on which the server connection should be registered.
gwserv		Gateway service, which is the port on which a registration can be done. In order to resolve the service names **sapgwXXX**, a lookup in **etc/services** is performed by the network layer of the operating system. If using port numbers instead of symbolic service names, no lookups are performed and no additional entries are needed.
progid		The program ID with which the registration is done. Serves as an identifier on the gateway and in the destination in the ABAP system.
repositoryDestination		Specifies a destination name that the server can use in order to retrieve metadata from a metadata repository hosted in a remote SAP server.
connectionCount		The number of connections that should be registered at the gateway.

55.3.3.2. Secure network connection options

The secure network connection options for the server data configuration object are as follows:

Name	Default Value	Description

sncMode		Secure network connection (SNC) mode, **0** (off) or **1** (on).
sncQop		SNC level of security, **1** to **9**.
sncMyname		SNC name of your server. Overrides the default SNC name. Typically something like **p:CN=JCoServer, O=ACompany, C=EN**.
sncLib		Path to library which provides SNC service. If this property is not provided, the value of the **jco.middleware.snc_lib** property is used instead.

55.3.3.3. Other options

The other options for the server data configuration object are, as follows:

Name	Default Value	Description
saprouter		SAP router string to use for a system protected by a firewall, which can therefore only be reached through a SAProuter, when registering the server at the gateway of that ABAP System. A typical router string is **/H/firewall.hostname/H/**.
maxStartupDelay		The maximum time (in seconds) between two start-up attempts in case of failures. The waiting time is doubled from initially 1 second after each start-up failure until either the maximum value is reached or the server could be started successfully.
trace		Enable/disable RFC trace (**0** or **1**)
workerThreadCount		The maximum number of threads used by the server connection. If not set, the value for the **connectionCount** is used as the **workerThreadCount**. The maximum number of threads can not exceed 99.

workerThreadMinCount		The minimum number of threads used by server connection. If not set, the value for **connectionCount** is used as the **workerThreadMinCount**.

55.3.4. Repository Configuration

The configurations for repositories are maintained in the **repositoryDataStore** property of the SAP Component. Each entry in this map configures a distinct repository. The key for each entry is the name of the repository and this key also corresponds to the name of the server to which this repository is attached.

The value of each entry is a repository data configuration object, **org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl**, that defines the contents of a metadata repository. A repository data object is a map of function template configuration objects, **org.fuesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl**. Each entry in this map specifies the interface of a function module and the key for each entry is the name of the function module specified.

Repository data example

The following code shows a simple example of configuring a metadata repository:

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint ... >
    ...
    <!-- Configures the sap-srfc-server component -->
    <bean id="sap-configuration"
        class="org.fusesource.camel.component.sap.SapConnectionConfiguration">
        <property name="repositoryDataStore">
            <map>
                <entry key="nplServer" value-ref="nplRepositoryData" />
            </map>
        </property>
    </bean>

    <!-- Configures a Metadata Repository -->
    <bean id="nplRepositoryData"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RepositoryDataImpl">
        <property name="functionTemplates">
            <map>
                <entry key="BOOK_FLIGHT" value-ref="bookFlightFunctionTemplate" />
            </map>
        </property>
    </bean>
    ...
</blueprint>
```

55.3.4.1. Function template properties

The interface of a function module consists of four parameter lists by which data is transferred back and forth to the function module in an RFC call. Each parameter list consists of one or more fields, each of

which is a named parameter transferred in an RFC call. The following parameter lists and exception list are supported:

- The *import parameter list* contains parameter values sent to a function module in an RFC call;
- The *export parameter list* contains parameter values that are returned by a function module in an RFC call;
- The *changing parameter list* contains parameter values sent to and returned by a function module in an RFC call;
- The *table parameter list* contains internal table values sent to and returned by a function module in an RFC call.
- The interface of a function module also consists of an *exception list* of ABAP exceptions that may be raised when the module is invoked in an RFC call.

A function template describes the name and type of parameters in each parameter list of a function interface and the ABAP exceptions thrown by the function. A function template object maintains five property lists of metadata objects, as described in the following table.

Property	Description
importParameterList	A list of list field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**. Specifies the parameters sent in an RFC call to a function module.
changingParameterList	A list of list field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**. Specifies the parameters sent and returned in an RFC call to and from a function module.
exportParameterList	A list of list field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**. Specifies the parameters returned in an RFC call from a function module.
tableParameterList	A list of list field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**. Specifies the table parameters that are sent and returned in an RFC call to and from a function module.
exceptionList	A list of ABAP exception metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.AbapExceptionImpl**. Specifies the ABAP exceptions potentially raised in an RFC call of the function module.

Function template example

The following example shows an outline of how to configure a function template:

```
<bean id="bookFlightFunctionTemplate"
    class="org.fusesource.camel.component.sap.model.rfc.impl.FunctionTemplateImpl">
    <property name="importParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="changingParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exportParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="tableParameterList">
        <list>
            ...
        </list>
    </property>
    <property name="exceptionList">
        <list>
            ...
        </list>
    </property>
</bean>
```

55.3.4.2. List field metadata properties

A list field metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMeataDataImpl**, specifies the name and type of a field in a parameter list. For an elementary parameter field (**CHAR**, **DATE**, **BCD**, **TIME**, **BYTE**, **NUM**, **FLOAT**, **INT**, **INT1**, **INT2**, **DECF16**, **DECF34**, **STRING**, **XSTRING**), the following table lists the configuration properties that may be set on a list field metadata object:

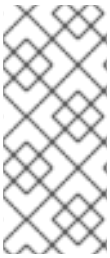
Name	Default Value	Description
name	-	The name of the parameter field.
type	-	The parameter type of the field.
byteLength	-	The field length in bytes for a non-Unicode layout. This value depends on the parameter type.
unicodeByteLength	-	The field length in bytes for a Unicode layout. This value depends on the parameter type.

decimals	**0**	The number of decimals in field value. Required for parameter types BCD and FLOAT.
optional	**false**	If **true**, the field is optional and need not be set in an RFC call.

Note that all elementary parameter fields require that the **name**, **type**, **byteLength**, and **unicodeByteLength** properties be specified in the field metadata object. In addition, the **BCD**, **FLOAT**, **DECF16**, and **DECF34** fields require the decimal property to be specified in the field metadata object.

For a complex parameter field of type **TABLE** or **STRUCTURE**, the following table lists the configuration properties that may be set on a list field metadata object:

Name	Default Value	Description
name	-	The name of the parameter field.
type	-	The parameter type of the field.
recordMetaData	-	The metadata for the structure or table. A record metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl**, is passed to specify the fields in the structure or table rows.
optional	**false**	If **true**, the field is optional and need not be set in a RFC call.



NOTE

All complex parameter fields require that the **name**, **type**, and **recordMetaData** properties be specified in the field metadata object. The value of the **recordMetaData** property is a record field metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl**, which specifies the structure of a nested structure or the structure of a table row.

Elementary list field metadata example

The following metadata configuration specifies an optional, 24-digit packed BCD number parameter with two decimal places named **TICKET_PRICE**:

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
   <property name="name" value="TICKET_PRICE" />
   <property name="type" value="BCD" />
   <property name="byteLength" value="12" />
   <property name="unicodeByteLength" value="24" />
```

```
    <property name="decimals" value="2" />
    <property name="optional" value="true" />
</bean>
```

Complex list field metadata example

The following metadata configuration specifies a required **TABLE** parameter named **CONNINFO** with a row structure specified by the **connectionInfo** record metadata object:

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.ListFieldMetaDataImpl">
    <property name="name" value="CONNINFO" />
    <property name="type" value="TABLE" />
    <property name="recordMetaData" ref="connectionInfo" />
</bean>
```

55.3.4.3. Record metadata properties

A record metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl**, specifies the name and contents of a nested **STRUCTURE** or the row of a **TABLE** parameter. A record metadata object maintains a list of record field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**, which specifies the parameters that reside in the nested structure or table row.

The following table lists configuration properties that may be set on a record metadata object:

Name	Default Value	Description
name	-	The name of the record.
recordFieldMetaData	-	The list of record field metadata objects, **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**. Specifies the fields contained within the structure.

NOTE

All properties of the record metadata object are required.

Record metadata example

The following example shows how to configure a record metadata object:

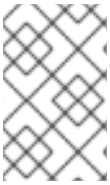
```
<bean    id="connectionInfo"
        class="org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl">
    <property name="name" value="CONNECTION_INFO" />
    <property name="recordFieldMetaData">
        <list>
            ...
```

```
        </list>
      </property>
    </bean>
```

55.3.4.4. Record field metadata properties

A record field metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl**, specifies the name and type of a parameter field within a structure.

A record field metadata object is similar to a parameter field metadata object, except that the offsets of the individual field locations within the nested structure or table row must be additionally specified. The non-Unicode and Unicode offsets of an individual field must be calculated and specified from the sum of non-Unicode and Unicode byte lengths of the preceding fields in the structure or row.



NOTE

The failure to properly specify the offsets of fields in nested structures and table rows will cause the field storage of parameters in the underlying JCo and ABAP runtimes to overlap and prevent the proper transfer of values in RFC calls.

For an elementary parameter field (**CHAR**, **DATE**, **BCD**, **TIME**, **BYTE**, **NUM**, **FLOAT**, **INT**, **INT1**, **INT2**, **DECF16**, **DECF34**, **STRING**, **XSTRING**), the following table lists the configuration properties that may be set on a record field metadata object:

Name	Default Value	Description
name	-	The name of the parameter field.
type	-	The parameter type of the field.
byteLength	-	The field length in bytes for a non-Unicode layout. This value depends on the parameter type.
unicodeByteLength	-	The field length in bytes for a Unicode layout. This value depends on the parameter type.
byteOffset	-	The field offset in bytes for non-Unicode layout. This offset is the byte location of the field within the enclosing structure.
unicodeByteOffset	-	The field offset in bytes for Unicode layout. This offset is the byte location of the field within the enclosing structure.

decimals	**0**	The number of decimals in field value; only required for parameter types **BCD** and **FLOAT**.

For a complex parameter field of type **TABLE** or **STRUCTURE**, the following table lists the configuration properties that may be set on a record field metadata object:

Name	Default Value	Description
name	-	The name of the parameter field.
type	-	The parameter type of the field.
byteOffset	-	The field offset in bytes for non-Unicode layout. This offset is the byte location of the field within the enclosing structure.
unicodeByteOffset	-	The field offset in bytes for Unicode layout. This offset is the byte location of the field within the enclosing structure.
recordMetaData	-	The metadata for the structure or table. A record metadata object, **org.fusesource.camel.component.sap.model.rfc.impl.RecordMetaDataImpl**, is passed to specify the fields in the structure or table rows.

Elementary record field metadata example

The following metadata configuration specifies a **DATE** field parameter named **ARRDATE** located 85 bytes into the enclosing structure in the case of a non-Unicode layout and located 170 bytes into the enclosing structure in the case of a Unicode layout.

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
   <property name="name" value="ARRDATE" />
   <property name="type" value="DATE" />
   <property name="byteLength" value="8" />
   <property name="unicodeByteLength" value="16" />
   <property name="byteOffset" value="85" />
   <property name="unicodeByteOffset" value="170" />
</bean>
```

Complex record field metadata example

The following metadata configuration specifies a **STRUCTURE** field parameter named **FLTINFO** with a structure specified by the **flightInfo** record metadata object. The parameter is located at the beginning of the enclosing structure in both the case of a non-Unicode and Unicode layout.

```
<bean    class="org.fusesource.camel.component.sap.model.rfc.impl.FieldMetaDataImpl">
   <property name="name" value="FLTINFO" />
   <property name="type" value="STRUCTURE" />
   <property name="byteOffset" value="0" />
   <property name="unicodeByteOffset" value="0" />
   <property name="recordMetaData" ref="flightInfo" />
</bean>
```

55.4. MESSAGE HEADERS

The SAP component supports the following message headers:

Header	Description
CamelSap.scheme	The URI scheme of the last endpoint to process the message. Use one of the following values: **sap-srfc-destination** **sap-trfc-destination** **sap-qrfc-destination** **sap-srfc-server** **sap-trfc-server** **sap-idoc-destination** **sap-idoclist-destination** **sap-qidoc-destination** **sap-qidoclist-destination** **sap-idoclist-server**
CamelSap.destinationName	The destination name of the last destination endpoint to process the message.
CamelSap.serverName	The server name of the last server endpoint to process the message.
CamelSap.queueName	The queue name of the last queuing endpoint to process the message.
CamelSap.rfcName	The RFC name of the last RFC endpoint to process the message.
CamelSap.idocType	The IDoc type of the last IDoc endpoint to process the message.

CamelSap.idocTypeExtension	The IDoc type extension, if any, of the last IDoc endpoint to process the message.
CamelSap.systemRelease	The system release, if any, of the last IDoc endpoint to process the message.
CamelSap.applicationRelease	The application release, if any, of the last IDoc endpoint to process the message.

55.5. EXCHANGE PROPERTIES

The SAP component adds the following exchange properties:

Property	Description
CamelSap.destinationPropertiesMap	A map containing the properties of each SAP destination encountered by the exchange. The map is keyed by destination name and each entry is a **java.util.Properties** object containing the configuration properties of that destination.
CamelSap.serverPropertiesMap	A map containing the properties of each SAP server encountered by the exchange. The map is keyed by server name and each entry is a **java.util.Properties** object containing the configuration properties of that server.

55.6. MESSAGE BODY FOR RFC

55.6.1. Request and response objects

An SAP endpoint expects to receive a message with a message body containing an SAP request object and will return a message with a message body containing an SAP response object. SAP requests and responses are fixed map data structures containing named fields with each field having a predefined data type.

Note that the named fields in an SAP request and response are specific to an SAP endpoint, with each endpoint defining the parameters in the SAP request and response it will accept. An SAP endpoint provides factory methods to create the request and response objects that are specific to it.

```
public class SAPEndpoint ... {
    ...
    public Structure getRequest() throws Exception;

    public Structure getResponse() throws Exception;
    ...
}
```

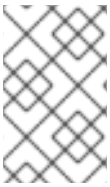
55.6.2. Structure objects

Both SAP request and response objects are represented in Java as a structure object which supports the **org.fusesource.camel.component.sap.model.rfc.Structure** interface. This interface extends both the **java.util.Map** and **org.eclipse.emf.ecore.EObject** interfaces.

```
public interface Structure extends org.eclipse.emf.ecore.EObject,
                                   java.util.Map<String, Object> {

    <T> T get(Object key, Class<T> type);

}
```

The field values in a structure object are accessed through the field's getter methods in the map interface. In addition, the structure interface provides a type-restricted method to retrieve field values.

Structure objects are implemented in the component runtime using the Eclipse Modeling Framework (EMF) and support that framework's **EObject** interface. Instances of a structure object have attached metadata which define and restrict the structure and contents of the map of fields it provides. This metadata can be accessed and introspected using the standard methods provided by EMF. Please refer to the EMF documentation for further details.



NOTE

Attempts to get a parameter not defined on a structure object will return null. Attempts to set a parameter not defined on a structure will throw an exception as well as attempts to set the value of a parameter with an incorrect type.

As discussed in the following sections, structure objects can contain fields that contain values of the complex field types, **STRUCTURE**, and **TABLE**.

NOTE

It is unnecessary to create instances of these types and add them to the structure. Instances of these field values are created on demand if necessary when accessed in the enclosing structure.

55.6.3. Field types

The fields that reside within the structure object of an SAP request or response may be either *elementary* or *complex*. An elementary field contains a single scalar value, whereas a complex field will contain one or more fields of either an elementary or complex type.

55.6.3.1. Elementary field types

An elementary field may be a character, numeric, hexadecimal or string field type. The following table summarizes the types of elementary fields that may reside in a structure object:

Field Type	Corresponding Java Type	Byte Length	Unicode Byte Length	Number Decimals Digits	Description

CHAR	**java.lang.String**	1 to 65535	1 to 65535	-	ABAP Type 'C': Fixed sized character string
DATE	**java.util.Date**	8	16	-	ABAP Type 'D': Date (format: YYYYMMDD)
BCD	**java.math.BigDecimal**	1 to 16	1 to 16	0 to 14	ABAP Type 'P': Packed BCD number. A BCD number contains two digits per byte.
TIME	**java.util.Date**	6	12	-	ABAP Type 'T': Time (format: HHMMSS)
BYTE	**byte[]**	1 to 65535	1 to 65535	-	ABAP Type 'X':Fixed sized byte array
NUM	**java.lang.String**	1 to 65535	1 to 65535	-	ABAP Type 'N': Fixed sized numeric character string
FLOAT	**java.lang.Double**	8	8	0 to 15	ABAP Type 'F': Floating point number
INT	**java.lang.Integer**	4	4	-	ABAP Type 'I': 4-byte Integer
INT2	**java.lang.Integer**	2	2	-	ABAP Type 'S': 2-byte Integer
INT1	**java.lang.Integer**	1	1	-	ABAP Type 'B': 1-byte Integer
DECF16	**java.match.BigDecimal**	8	8	16	ABAP Type 'decfloat16': 8-byte Decimal Floating Point Number

DECF34	**java.math.BigDecimal**	16	16	34	ABAP Type 'decfloat34': 16-byte Decimal Floating Point Number
STRING	**java.lang.String**	8	8	-	ABAP Type 'G': Variable length character string
XSTRING	**byte[]**	8	8	-	ABAP Type 'Y': Variable length byte array

55.6.3.2. Character field types

A character field contains a fixed sized character string that may use either a non-Unicode or Unicode character encoding in the underlying JCo and ABAP runtimes. Non-Unicode character strings encode one character per byte. Unicode character strings are encoded in two bytes using UTF-16 encoding. Character field values are represented in Java as **java.lang.String** objects and the underlying JCo runtime is responsible for the conversion to their ABAP representation.

A character field declares its field length in its associated **byteLength** and **unicodeByteLength** properties, which determine the length of the field's character string in each encoding system.

CHAR

A **CHAR** character field is a text field containing alphanumeric characters and corresponds to the ABAP type C.

NUM

A **NUM** character field is a numeric text field containing numeric characters only and corresponds to the ABAP type N.

DATE

A **DATE** character field is an 8 character date field with the year, month and day formatted as **YYYYMMDD** and corresponds to the ABAP type D.

TIME

A **TIME** character field is a 6 character time field with the hours, minutes and seconds formatted as **HHMMSS** and corresponds to the ABAP type T.

55.6.3.3. Numeric field types

A numeric field contains a number. The following numeric field types are supported:

INT

An **INT** numeric field is an integer field stored as a 4-byte integer value in the underlying JCo and ABAP runtimes and corresponds to the ABAP type I. An **INT** field value is represented in Java as a **java.lang.Integer** object.

INT2

An **INT2** numeric field is an integer field stored as a 2-byte integer value in the underlying JCo and ABAP runtimes and corresponds to the ABAP type S. An **INT2** field value is represented in Java as a **java.lang.Integer** object.

INT1

An **INT1** field is an integer field stored as a 1-byte integer value in the underlying JCo and ABAP runtimes value and corresponds to the ABAP type B. An **INT1** field value is represented in Java as a **java.lang.Integer** object.

FLOAT

A **FLOAT** field is a binary floating point number field stored as an 8-byte double value in the underlying JCo and ABAP runtimes and corresponds to the ABAP type F. A **FLOAT** field declares the number of decimal digits that the field's value contains in its associated decimal property. In the case of a **FLOAT** field, this decimal property can have a value between 1 and 15 digits. A **FLOAT** field value is represented in Java as a **java.lang.Double** object.

BCD

A **BCD** field is a binary coded decimal field stored as a 1 to 16 byte packed number in the underlying JCo and ABAP runtimes and corresponds to the ABAP type P. A packed number stores two decimal digits per byte. A **BCD** field declares its field length in its associated **byteLength** and **unicodeByteLength** properties. In the case of a **BCD** field, these properties can have a value between 1 and 16 bytes, and both properties will have the same value. A **BCD** field declares the number of decimal digits that the field's value contains in its associated decimal property. In the case of a **BCD** field, this decimal property can have a value between 1 and 14 digits. A **BCD** field value is represented in Java as a **java.math.BigDecimal**.

DECF16

A **DECF16** field is a decimal floating point stored as an 8-byte IEEE 754 decimal64 floating point value in the underlying JCo and ABAP runtimes and corresponds to the ABAP type **decfloat16**. The value of a **DECF16** field has 16 decimal digits. The value of a **DECF16** field is represented in Java as **java.math.BigDecimal**.

DECF34

A **DECF34** field is a decimal floating point stored as a 16-byte IEEE 754 decimal128 floating point value in the underlying JCo and ABAP runtimes and corresponds to the ABAP type **decfloat34**. The value of a **DECF34** field has 34 decimal digits. The value of a **DECF34** field is represented in Java as **java.math.BigDecimal**.

55.6.3.4. Hexadecimal field types

A hexadecimal field contains raw binary data. The following hexadecimal field types are supported:

BYTE

A **BYTE** field is a fixed sized byte string stored as a byte array in the underlying JCo and ABAP runtimes and corresponds to the ABAP type X. A **BYTE** field declares its field length in its associated **byteLength** and **unicodeByteLength** properties. In the case of a **BYTE** field, these properties can have a value between 1 and 65535 bytes and both properties will have the same value. The value of a **BYTE** field is represented in Java as a **byte[]** object.

55.6.3.5. String field types

A string field references a variable length string value. The length of that string value is not fixed until runtime. The storage for the string value is dynamically created in the underlying JCo and ABAP runtimes. The storage for the string field itself is fixed and contains only a string header.

STRING

A **STRING** field refers to a character string stored in the underlying JCo and ABAP runtimes as an 8-byte value. It corresponds to the ABAP type G. The value of the **STRING** field is represented in Java as a **java.lang.String** object.

XSTRING

An **XSTRING** field refers to a byte string stored in the underlying JCo and ABAP runtimes as an 8-byte value. It corresponds to the ABAP type Y. The value of the **STRING** field is represented in Java as a **byte[]** object.

55.6.3.6. Complex field types

A complex field may be either a structure or table field type. The following table summarizes these complex field types.

Field Type	Corresponding Java Type	Byte Length	Unicode Byte Length	Number Decimals Digits	Description
STRUCTURE	**org.fusesource.camel.component.sap.model.rfc.Structure**	Total of individual field byte lengths	Total of individual field Unicode byte lengths	-	ABAP Type 'u' & 'v': Heterogeneous Structure
TABLE	**org.fusesource.camel.component.sap.model.rfc.Table**	Byte length of row structure	Unicode byte length of row structure	-	ABAP Type 'h': Table

55.6.3.7. Structure field types

A **STRUCTURE** field contains a structure object and is stored in the underlying JCo and ABAP runtimes as an ABAP structure record. It corresponds to either an ABAP type **u** or **v**. The value of a **STRUCTURE** field is represented in Java as a structure object with the interface **org.fusesource.camel.component.sap.model.rfc.Structure**.

55.6.3.8. Table field types

A **TABLE** field contains a table object and is stored in the underlying JCo and ABAP runtimes as an ABAP internal table. It corresponds to the ABAP type **h**. The value of the field is represented in Java by a table object with the interface **org.fusesource.camel.component.sap.model.rfc.Table**.

55.6.3.9. Table objects

A table object is a homogeneous list data structure containing rows of structure objects with the same structure. This interface extends both the **java.util.List** and **org.eclipse.emf.ecore.EObject** interfaces.

```
public interface Table<S extends Structure>
    extends org.eclipse.emf.ecore.EObject,
    java.util.List<S> {

    /**
```

```
     * Creates and adds a table row at the end of the row list
     */
    S add();

    /**
     * Creates and adds a table row at the index in the row list
     */
    S add(int index);

}
```

The list of rows in a table object is accessed and managed using the standard methods defined in the list interface. In addition, the table interface provides two factory methods for creating and adding structure objects to the row list.

Table objects are implemented in the component runtime using the Eclipse Modeling Framework (EMF) and support that framework's EObject interface. Instances of a table object have attached metadata which define and restrict the structure and contents of the rows it provides. This metadata can be accessed and introspected using the standard methods provided by EMF. Please refer to the EMF documentation for further details.

NOTE

Attempts to add or set a row structure value of the wrong type will throw an exception.

55.7. MESSAGE BODY FOR IDOC

55.7.1. IDoc message type

When using one of the IDoc Camel SAP endpoints, the type of the message body depends on which particular endpoint you are using.

For a **sap-idoc-destination** endpoint or a **sap-qidoc-destination** endpoint, the message body is of **Document** type:

```
org.fusesource.camel.component.sap.model.idoc.Document
```

For a **sap-idoclist-destination** endpoint, a **sap-qidoclist-destination** endpoint, or a **sap-idoclist-server** endpoint, the message body is of **DocumentList** type:

```
org.fusesource.camel.component.sap.model.idoc.DocumentList
```

55.7.2. The IDoc document model

For the Camel SAP component, an IDoc document is modeled using the Eclipse Modeling Framework (EMF), which provides a wrapper API around the underlying SAP IDoc API. The most important types in this model are:

```
org.fusesource.camel.component.sap.model.idoc.Document
org.fusesource.camel.component.sap.model.idoc.Segment
```

The **Document** type represents an IDoc document instance. In outline, the **Document** interface exposes the following methods:

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Document extends EObject {
    // Access the field values from the IDoc control record
    String getArchiveKey();
    void setArchiveKey(String value);
    String getClient();
    void setClient(String value);
    ...

    // Access the IDoc document contents
    Segment getRootSegment();
}
```

The following kinds of method are exposed by the **Document** interface:

Methods for accessing the control record

Most of the methods are for accessing or modifying field values of the IDoc control record. These methods are of the form *AttributeName*, where *AttributeName* is the name of a field value.

Method for accessing the document contents

The **getRootSegment** method provides access to the document contents (IDoc data records), returning the contents as a **Segment** object. Each **Segment** object can contain an arbitrary number of child segments, and the segments can be nested to an arbitrary degree.

Note, however, that the precise layout of the segment hierarchy is defined by the particular *IDoc type* of the document. When creating (or reading) a segment hierarchy, therefore, you must be sure to follow the exact structure as defined by the IDoc type.

The **Segment** type is used to access the data records of the IDoc document, where the segments are laid out in accordance with the structure defined by the document's IDoc type. In outline, the **Segment** interface exposes the following methods:

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface Segment extends EObject, java.util.Map<String, Object> {
    // Returns the value of the '<em><b>Parent</b></em>' reference.
    Segment getParent();

    // Return an immutable list of all child segments
    <S extends Segment> EList<S> getChildren();

    // Returns a list of child segments of the specified segment type.
    <S extends Segment> SegmentList<S> getChildren(String segmentType);

    EList<String> getTypes();

    Document getDocument();

    String getDescription();

    String getType();
```

```
    String getDefinition();

    int getHierarchyLevel();

    String getIdocType();

    String getIdocTypeExtension();

    String getSystemRelease();

    String getApplicationRelease();

    int getNumFields();

    long getMaxOccurrence();

    long getMinOccurrence();

    boolean isMandatory();

    boolean isQualified();

    int getRecordLength();

    <T> T get(Object key, Class<T> type);
}
```

The **getChildren(String segmentType)** method is particularly useful for adding new (nested) children to a segment. It returns an object of type, **SegmentList**, which is defined as follows:

```
// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface SegmentList<S extends Segment> extends EObject, EList<S> {
    S add();

    S add(int index);
}
```

Hence, to create a data record of **E1SCU_CRE** type, you could use Java code like the following:

```
Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();
```

55.7.3. How an IDoc is related to a Document object

According to the SAP documentation, an IDoc document consists of the following main parts:

Control record

The control record (which contains the metadata for the IDoc document) is represented by the attributes on the **Document** object.

Data records

The data records are represented by the **Segment** objects, which are constructed as a nested hierarchy of segments. You can access the root segment through the **Document.getRootSegment** method.

Status records

In the Camel SAP component, the status records are **not** represented by the document model. But you do have access to the latest status value through the **status** attribute on the control record.

Example of creating a Document instance

The following example shows how to create an IDoc document with the IDoc type, **FLCUSTOMER_CREATEFROMDATA01**, using the IDoc model API in Java.

Example 55.1. Creating an IDoc Document in Java

```
// Java
import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.util.IDocUtil;

import org.fusesource.camel.component.sap.model.idoc.Document;
import org.fusesource.camel.component.sap.model.idoc.DocumentList;
import org.fusesource.camel.component.sap.model.idoc.IdocFactory;
import org.fusesource.camel.component.sap.model.idoc.IdocPackage;
import org.fusesource.camel.component.sap.model.idoc.Segment;
import org.fusesource.camel.component.sap.model.idoc.SegmentChildren;
...
//
// Create a new IDoc instance using the modeling classes
//

// Get the SAP Endpoint bean from the Camel context.
// In this example, it's a 'sap-idoc-destination' endpoint.
SapTransactionalIDocDestinationEndpoint endpoint =
    exchange.getContext().getEndpoint(
        "bean:SapEndpointBeanID",
        SapTransactionalIDocDestinationEndpoint.class
    );

// The endpoint automatically populates some required control record attributes
Document document = endpoint.createDocument()

// Initialize additional control record attributes
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
document.setRecipientPartnerNumber("QUICKCLNT");
document.setRecipientPartnerType("LS");
document.setSenderPartnerNumber("QUICKSTART");
document.setSenderPartnerType("LS");

Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();

Segment E1BPSCUNEW_Segment =
E1SCU_CRE_Segment.getChildren("E1BPSCUNEW").add();
E1BPSCUNEW_Segment.put("CUSTNAME", "Fred Flintstone");
E1BPSCUNEW_Segment.put("FORM", "Mr.");
```

```
E1BPSCUNEW_Segment.put("STREET", "123 Rubble Lane");
E1BPSCUNEW_Segment.put("POSTCODE", "01234");
E1BPSCUNEW_Segment.put("CITY", "Bedrock");
E1BPSCUNEW_Segment.put("COUNTR", "US");
E1BPSCUNEW_Segment.put("PHONE", "800-555-1212");
E1BPSCUNEW_Segment.put("EMAIL", "fred@bedrock.com");
E1BPSCUNEW_Segment.put("CUSTTYPE", "P");
E1BPSCUNEW_Segment.put("DISCOUNT", "005");
E1BPSCUNEW_Segment.put("LANGU", "E");
```

55.8. DOCUMENT ATTRIBUTES

IDoc Document Attributes table shows the control record attributes that you can set on the **Document** object.

Table 55.2. IDoc Document Attributes

Attribute	Length	SAP Field	Description
archiveKey	70	**ARCKEY**	EDI archive key
client	3	**MANDT**	Client
creationDate	8	**CREDAT**	Date IDoc was created
creationTime	6	**CRETIM**	Time IDoc was created
direction	1	**DIRECT**	Direction
eDIMessage	14	**REFMES**	Reference to message
eDIMessageGroup	14	**REFGRP**	Reference to message group
eDIMessageType	6	**STDMES**	EDI message type
eDIStandardFlag	1	**STD**	EDI standard
eDIStandardVersion	6	**STDVRS**	Version of EDI standard
eDITransmissionFile	14	**REFINT**	Reference to interchange file
iDocCompoundType	8	**DOCTYP**	IDoc type
iDocNumber	16	**DOCNUM**	IDoc number
iDocSAPRelease	4	**DOCREL**	SAP Release of IDoc

Attribute	Length	SAP Field	Description
iDocType	30	**IDOCTP**	Name of basic IDoc type
iDocTypeExtension	30	**CIMTYP**	Name of extension type
messageCode	3	**MESCOD**	Logical message code
messageFunction	3	**MESFCT**	Logical message function
messageType	30	**MESTYP**	Logical message type
outputMode	1	**OUTMOD**	Output mode
recipientAddress	10	**RCVSAD**	Receiver address (SADR)
recipientLogicalAddress	70	**RCVLAD**	Logical address of receiver
recipientPartnerFunction	2	**RCVPFC**	Partner function of receiver
recipientPartnerNumber	10	**RCVPRN**	Partner number of receiver
recipientPartnerType	2	**RCVPRT**	Partner type of receiver
recipientPort	10	**RCVPOR**	Receiver port (SAP System, EDI subsystem)
senderAddress		**SNDSAD**	Sender address (SADR)
senderLogicalAddress	70	**SNDLAD**	Logical address of sender
senderPartnerFunction	2	**SNDPFC**	Partner function of sender
senderPartnerNumber	10	**SNDPRN**	Partner number of sender
senderPartnerType	2	**SNDPRT**	Partner type of sender
senderPort	10	**SNDPOR**	Sender port (SAP System, EDI subsystem)

Attribute	Length	SAP Field	Description
serialization	20	**SERIAL**	EDI/ALE: Serialization field
status	2	**STATUS**	Status of IDoc
testFlag	1	**TEST**	Test flag

55.8.1. Setting document attributes in Java

When setting the control record attributes in Java, the usual convention for Java bean properties is followed. That is, a **name** attribute can be accessed through the **getName** and **setName** methods, for getting and setting the attribute value. For example, the **iDocType**, **iDocTypeExtension**, and **messageType** attributes can be set as follows on a **Document** object:

```
// Java
document.setIDocType("FLCUSTOMER_CREATEFROMDATA01");
document.setIDocTypeExtension("");
document.setMessageType("FLCUSTOMER_CREATEFROMDATA");
```

55.8.2. Setting document attributes in XML

When setting the control record attributes in XML, the attributes must be set on the **idoc:Document** element. For example, the **iDocType**, **iDocTypeExtension**, and **messageType** attributes can be set as follows:

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document ...
            iDocType="FLCUSTOMER_CREATEFROMDATA01"
            iDocTypeExtension=""
            messageType="FLCUSTOMER_CREATEFROMDATA" ... >
   ...
</idoc:Document>
```

55.9. TRANSACTION SUPPORT

55.9.1. BAPI transaction model

The SAP Component supports the BAPI transaction model for outbound communication with SAP. A destination endpoint with a URL containing the transacted option set to **true** will, if necessary, initiate a stateful session on the outbound connection of the endpoint and register a Camel Synchronization object with the exchange. This synchronization object will call the BAPI service method **BAPI_TRANSACTION_COMMIT** and end the stateful session when the processing of the message exchange is complete. If the processing of the message exchange fails, the synchronization object will call the BAPI server method **BAPI_TRANSACTION_ROLLBACK** and end the stateful session.

55.9.2. RFC transaction model

The tRFC protocol accomplishes an AT-MOST-ONCE delivery and processing guarantee by identifying

each transactional request with a unique transaction identifier (TID). A TID accompanies each request sent in the protocol. A sending application using the tRFC protocol must identify each instance of a request with a unique TID when sending the request. An application may send a request with a given TID multiple times, but the protocol ensures that the request is delivered and processed in the receiving system at most once. An application may choose to resend a request with a given TID when encountering a communication or system error when sending the request, and is thus in doubt as to whether that request was delivered and processed in the receiving system. By resending a request when encountering a communication error, a client application using the tRFC protocol can thus ensure EXACTLY-ONCE delivery and processing guarantees for its request.

55.9.3. Which transaction model to use?

A BAPI transaction is an application level transaction, in the sense that it imposes ACID guarantees on the persistent data changes performed by a BAPI method or RFC function in the SAP database. An RFC transaction is a communication transaction, in the sense that it imposes delivery guarantees (AT-MOST-ONCE, EXACTLY-ONCE, EXACTLY-ONCE-IN-ORDER) on requests to a BAPI method and/or RFC function.

55.9.4. Transactional RFC destination endpoints

The following destination endpoints support RFC transactions:

- **sap-trfc-destination**
- **sap-qrfc-destination**

A single Camel route can include multiple transactional RFC destination endpoints, sending messages to multiple RFC destinations and even sending messages to the same RFC destination multiple times. This implies that the Camel SAP component potentially needs to keep track of **many** transaction IDs (TIDs) for each **Exchange** object passing along a route. Now if the route processing fails and must be retried, the situation gets quite complicated. The RFC transaction semantics demand that each RFC destination along the route must be invoked using the **same** TID that was used the first time around (and where the TIDs for each destinations are distinct from each other). In other words, the Camel SAP component must keep track of which TID was used at which point along the route, and remember this information, so that the TIDs can be replayed in the correct order.

By default, Camel does not provide a mechanism that enables an **Exchange** to know where it is in a route. To provide such a mechanism, it is necessary to install the **CurrentProcessorDefinitionInterceptStrategy** interceptor into the Camel runtime. This interceptor must be installed into the Camel runtime, in order for the Camel SAP component to keep track of the TIDs in a route.

55.9.5. Transactional RFC server endpoints

The following server endpoints support RFC transactions:

- **sap-trfc-server**

When a Camel exchange processing a transactional request encounters a processing error, Camel handles the processing error through its standard error handling mechanisms. If the Camel route processing the exchange is configured to propagate the error back to the caller, the SAP server endpoint that initiated the exchange takes note of the failure and the sending SAP system is notified of the error. The sending SAP system can then respond by sending another transaction request with the same TID to process the request again.

55.10. XML SERIALIZATION FOR RFC

SAP request and response objects support an XML serialization format which enable these objects to be serialized to and from an XML document.

55.10.1. XML namespace

Each RFC in a repository defines a specific XML namespace for the elements which compose the serialized forms of its Request and Response objects. The form of this namespace URL is as follows:

```
http://sap.fusesource.org/rfc/<Repository Name>/<RFC Name>
```

RFC namespace URLs have a common **http://sap.fusesource.org/rfc** prefix followed by the name of the repository in which the RFC's metadata is defined. The final component in the URL is the name of the RFC itself.

55.10.2. Request and response XML documents

An SAP request object will be serialized into an XML document with the root element of that document named Request and scoped by the namespace of the request's RFC.

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Request>
```

An SAP response object will be serialized into an XML document with the root element of that document named Response and scoped by the namespace of the response's RFC.

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Response
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:Response>
```

55.10.3. Structure fields

Structure fields in parameter lists or nested structures are serialized as elements. The element name of the serialized structure corresponds to the field name of the structure within the enclosing parameter list, structure or table row entry it resides.

```
<BOOK_FLIGHT:FLTINFO
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
   ...
</BOOK_FLIGHT:FLTINFO>
```

Note that the type name of the structure element in the RFC namespace will correspond to the name of the record metadata object which defines the structure, as in the following example:

```
<xs:schema
    targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
    ...
    <xs:complexType name="FLTINFO_STRUCTURE”>
    ...
    </xs:complexType>
    ...
</xs:schema>
```

This distinction will be important when specifying a JAXB bean to marshal and unmarshal the structure.

55.10.4. Table fields

Table fields in parameter lists or nested structures are serialized as elements. The element name of the serialized structure will correspond to the field name of the table within the enclosing parameter list, structure, or table row entry it resides. The table element will contain a series of row elements to hold the serialized values of the table’s row entries.

```
<BOOK_FLIGHT:CONNINFO
     xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT">
    <row ... > ... </row>
    ...
    <row ... > ... </row>
</BOOK_FLIGHT:CONNINFO>
```

Note that the type name of the table element in the RFC namespace corresponds to the name of the record metadata object which defines the row structure of the table suffixed by **_TABLE**. The type name of the table row element in the RFC name corresponds to the name of the record metadata object which defines the row structure of the table, as in the following example:

```
<xs:schema
     targetNamespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
    <xs:complextType name="CONNECTION_INFO_STRUCTURE_TABLE”>
        <xs:sequence>
            <xs:element
                name="row”
                minOccures="0”
                maxOccurs="unbounded”
                type="CONNECTION_INFO_STRUCTURE”/>
            ...
            <xs:sequence>
        </xs:sequence>
    </xs:complexType>

    <xs:complextType name="CONNECTION_INFO_STRUCTURE”>
            ...
    </xs:complexType>
    ...
</xs:schema>
```

This distinction will be important when specifying a JAXB bean to marshal and unmarshal the structure.

55.10.5. Elementary fields

Elementary fields in parameter lists or nested structures are serialized as attributes on the element of the enclosing parameter list or structure. The attribute name of the serialized field corresponds to the field name of the field within the enclosing parameter list, structure, or table row entry it resides, as in the following example:

```
<?xml version="1.0" encoding="ASCII"?>
<BOOK_FLIGHT:Request
    xmlns:BOOK_FLIGHT="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT"
    CUSTNAME="James Legrand"
   PASSFORM="Mr"
   PASSNAME="Travelin Joe"
   PASSBIRTH="1990-03-17T00:00:00.000-0500"
   FLIGHTDATE="2014-03-19T00:00:00.000-0400"
   TRAVELAGENCYNUMBER="00000110"
   DESTINATION_FROM="SFO"
   DESTINATION_TO="FRA"/>
```

55.10.6. Date and time formats

Date and Time fields are serialized into attribute values using the following format:

```
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

Date fields will be serialized with only the year, month, day and timezone components set:

```
DEPDATE="2014-03-19T00:00:00.000-0400"
```

Time fields will be serialized with only the hour, minute, second, millisecond and timezone components set:

```
DEPTIME="1970-01-01T16:00:00.000-0500"
```

55.11. XML SERIALIZATION FOR IDOC

An IDoc message body can be serialized into an XML string format, with the help of a built-in type converter.

55.11.1. XML namespace

Each serialized IDoc is associated with an XML namespace, which has the following general format:

```
http://sap.fusesource.org/idoc/repositoryName/idocType/idocTypeExtension/systemRelease/applicationRelease
```

Both the *repositoryName* (name of the remote SAP metadata repository) and the *idocType* (IDoc document type) are mandatory, but the other components of the namespace can be left blank. For example, you could have an XML namespace like the following:

```
http://sap.fusesource.org/idoc/MY_REPO/FLCUSTOMER_CREATEFROMDATA01///
```

55.11.2. Built-in type converter

The Camel SAP component has a built-in type converter, which is capable of converting a **Document** object or a **DocumentList** object to and from a **String** type.

For example, to serialize a **Document** object to an XML string, you can simply add the following line to a route in XML DSL:

```
<convertBodyTo type="java.lang.String"/>
```

You can also use this approach to a serialized XML message into a **Document** object. For example, given that the current message body is a serialized XML string, you can convert it back into a **Document** object by adding the following line to a route in XML DSL:

```
<convertBodyTo type="org.fusesource.camel.component.sap.model.idoc.Document"/>
```

55.11.3. Sample IDoc message body in XML format

When you convert an IDoc message to a **String**, it is serialized into an XML document, where the root element is either **idoc:Document** (for a single document) or **idoc:DocumentList** (for a list of documents). It shows that a single IDoc document that has been serialized to an **idoc:Document** element.

Example 55.2. IDoc Message Body in XML

```
<?xml version="1.0" encoding="ASCII"?>
<idoc:Document
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:FLCUSTOMER_CREATEFROMDATA01---
="http://sap.fusesource.org/idoc/XXX/FLCUSTOMER_CREATEFROMDATA01///"
    xmlns:idoc="http://sap.fusesource.org/idoc"
    creationDate="2015-01-28T12:39:13.980-0500"
    creationTime="2015-01-28T12:39:13.980-0500"
    iDocType="FLCUSTOMER_CREATEFROMDATA01"
    iDocTypeExtension=""
    messageType="FLCUSTOMER_CREATEFROMDATA"
    recipientPartnerNumber="QUICKCLNT"
    recipientPartnerType="LS"
    senderPartnerNumber="QUICKSTART"
    senderPartnerType="LS">
  <rootSegment xsi:type="FLCUSTOMER_CREATEFROMDATA01---:ROOT" document="/">
    <segmentChildren parent="//@rootSegment">
      <E1SCU_CRE parent="//@rootSegment" document="/">
        <segmentChildren parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0">
          <E1BPSCUNEW parent="//@rootSegment/@segmentChildren/@E1SCU_CRE.0"
              document="/"
              CUSTNAME="Fred Flintstone" FORM="Mr."
              STREET="123 Rubble Lane"
              POSTCODE="01234"
              CITY="Bedrock"
              COUNTR="US"
              PHONE="800-555-1212"
              EMAIL="fred@bedrock.com"
              CUSTTYPE="P"
              DISCOUNT="005"
              LANGU="E"/>
```

```
        </segmentChildren>
      </E1SCU_CRE>
    </segmentChildren>
  </rootSegment>
</idoc:Document>
```

55.12. EXAMPLE 1: READING DATA FROM SAP

This example demonstrates a route that reads **FlightCustomer** business object data from SAP. The route invokes the **FlightCustomer** BAPI method, **BAPI_FLCUST_GETLIST**, using an SAP synchronous RFC destination endpoint to retrieve the data.

55.12.1. Java DSL for route

The Java DSL for the example route is as follows:

```
from("direct:getFlightCustomerInfo")
    .to("bean:createFlightCustomerGetListRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST")
    .to("bean:returnFlightCustomerInfo");
```

55.12.2. XML DSL for route

And the Spring DSL for the same route is as follows:

```
<route>
    <from uri="direct:getFlightCustomerInfo"/>
    <to uri="bean:createFlightCustomerGetListRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST"/>
    <to uri="bean:returnFlightCustomerInfo"/>
</route>
```

55.12.3. **createFlightCustomerGetListRequest** bean

The **createFlightCustomerGetListRequest** bean is responsible for building an SAP request object in its exchange method that is used in the RFC call of the subsequent SAP endpoint. The following code snippet demonstrates the sequence of operations to build the request object:

```
public void create(Exchange exchange) throws Exception {

    // Get SAP Endpoint to be called from context.
    SapSynchronousRfcDestinationEndpoint endpoint =
        exchange.getContext().getEndpoint("sap-srfc-destination:nplDest:BAPI_FLCUST_GETLIST",
                                          SapSynchronousRfcDestinationEndpoint.class);

    // Retrieve bean from message containing Flight Customer name to
    // look up.
    BookFlightRequest bookFlightRequest =
        exchange.getIn().getBody(BookFlightRequest.class);

    // Create SAP Request object from target endpoint.
```

```
    Structure request = endpoint.getRequest();

    // Add Customer Name to request if set
    if (bookFlightRequest.getCustomerName() != null &&
        bookFlightRequest.getCustomerName().length() > 0) {
            request.put("CUSTOMER_NAME",
                    bookFlightRequest.getCustomerName());
        }
    } else {
        throw new Exception("No Customer Name");
    }

    // Put request object into body of exchange message.
    exchange.getIn().setBody(request);
}
```

55.12.4. returnFlightCustomerInfo bean

The **returnFlightCustomerInfo** bean is responsible for extracting data from the SAP response object in its exchange method that it receives from the previous SAP endpoint. The following code snippet demonstrates the sequence of operations to extract the data from the response object:

```
public void createFlightCustomerInfo(Exchange exchange) throws Exception {

    // Retrieve SAP response object from body of exchange message.
    Structure flightCustomerGetListResponse =
        exchange.getIn().getBody(Structure.class);

    if (flightCustomerGetListResponse == null) {
        throw new Exception("No Flight Customer Get List Response");
    }

    // Check BAPI return parameter for errors
    @SuppressWarnings("unchecked")
    Table<Structure> bapiReturn =
        flightCustomerGetListResponse.get("RETURN", Table.class);
    Structure bapiReturnEntry = bapiReturn.get(0);
    if (bapiReturnEntry.get("TYPE", String.class) != "S") {
        String message = bapiReturnEntry.get("MESSAGE", String.class);
        throw new Exception("BAPI call failed: " + message);
    }

    // Get customer list table from response object.
    @SuppressWarnings("unchecked")
    Table<? extends Structure> customerList =
        flightCustomerGetListResponse.get("CUSTOMER_LIST", Table.class);

    if (customerList == null || customerList.size() == 0) {
        throw new Exception("No Customer Info.");
    }

    // Get Flight Customer data from first row of table.
    Structure customer = customerList.get(0);

    // Create bean to hold Flight Customer data.
```

```
    FlightCustomerInfo flightCustomerInfo = new FlightCustomerInfo();

    // Get customer id from Flight Customer data and add to bean.
    String customerId = customer.get("CUSTOMERID", String.class);
    if (customerId != null) {
        flightCustomerInfo.setCustomerNumber(customerId);
    }

    ...

    // Put bean into body of exchange message.
    exchange.getIn().setHeader("flightCustomerInfo", flightCustomerInfo);

}
```

55.13. EXAMPLE 2: WRITING DATA TO SAP

This example demonstrates a route that creates a **FlightTrip** business object instance in SAP. The route invokes the **FlightTrip** BAPI method, **BAPI_FLTRIP_CREATE**, using a destination endpoint to create the object.

55.13.1. Java DSL for route

The Java DSL for the example route is as follows:

```
from("direct:createFlightTrip")
    .to("bean:createFlightTripRequest")
    .to("sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true")
    .to("bean:returnFlightTripResponse");
```

55.13.2. XML DSL for route

And the Spring DSL for the same route is as follows:

```
<route>
    <from uri="direct:createFlightTrip"/>
    <to uri="bean:createFlightTripRequest"/>
    <to uri="sap-srfc-destination:nplDest:BAPI_FLTRIP_CREATE?transacted=true"/>
    <to uri="bean:returnFlightTripResponse"/>
</route>
```

55.13.3. Transaction support

Note that the URL for the SAP endpoint has the **transacted** option set to **true**. When this option is enabled, the endpoint ensures that an SAP transaction session has been initiated before invoking the RFC call. Because this endpoint's RFC creates new data in SAP, this option is necessary to make the route's changes permanent in SAP.

55.13.4. Populating request parameters

The **createFlightTripRequest** and **returnFlightTripResponse** beans are responsible for populating request parameters into the SAP request and extracting response parameters from the SAP response respectively, following the same sequence of operations as demonstrated in the previous example.

55.14. EXAMPLE 3: HANDLING REQUESTS FROM SAP

This example demonstrates a route which handles a request from SAP to the **BOOK_FLIGHT** RFC, which is implemented by the route. In addition, it demonstrates the component's XML serialization support, using JAXB to unmarshal and marshal SAP request objects and response objects to custom beans.

This route creates a **FlightTrip** business object on behalf of a travel agent, **FlightCustomer**. The route first unmarshals the SAP request object received by the SAP server endpoint into a custom JAXB bean. This custom bean is then multicasted in the exchange to three sub-routes, which gather the travel agent, flight connection, and passenger information required to create the flight trip. The final sub-route creates the flight trip object in SAP, as demonstrated in the previous example. The final sub-route also creates and returns a custom JAXB bean which is marshaled into an SAP response object and returned by the server endpoint.

55.14.1. Java DSL for route

The Java DSL for the example route is as follows:

```
DataFormat jaxb = new JaxbDataFormat("org.fusesource.sap.example.jaxb");

from("sap-srfc-server:nplserver:BOOK_FLIGHT")
    .unmarshal(jaxb)
    .multicast()
    .to("direct:getFlightConnectionInfo",
        "direct:getFlightCustomerInfo",
        "direct:getPassengerInfo")
    .end()
    .to("direct:createFlightTrip")
    .marshal(jaxb);
```

55.14.2. XML DSL for route

And the XML DSL for the same route is as follows:

```
<route>
    <from uri="sap-srfc-server:nplserver:BOOK_FLIGHT"/>
    <unmarshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </unmarshal>
    <multicast>
        <to uri="direct:getFlightConnectionInfo"/>
        <to uri="direct:getFlightCustomerInfo"/>
        <to uri="direct:getPassengerInfo"/>
    </multicast>
    <to uri="direct:createFlightTrip"/>
    <marshal>
        <jaxb contextPath="org.fusesource.sap.example.jaxb"/>
    </marshal>
</route>
```

55.14.3. BookFlightRequest bean

The following listing illustrates a JAXB bean which unmarshals from the serialized form of an SAP **BOOK_FLIGHT** request object:

```
@XmlRootElement(name="Request",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightRequest {

    @XmlAttribute(name="CUSTNAME")
    private String customerName;

    @XmlAttribute(name="FLIGHTDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date flightDate;

    @XmlAttribute(name="TRAVELAGENCYNUMBER")
    private String travelAgencyNumber;

    @XmlAttribute(name="DESTINATION_FROM")
    private String startAirportCode;

    @XmlAttribute(name="DESTINATION_TO")
    private String endAirportCode;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlAttribute(name="CLASS")
    private String flightClass;

    ...
}
```

55.14.4. BookFlightResponse bean

The following listing illustrates a JAXB bean which marshals to the serialized form of an SAP **BOOK_FLIGHT** response object:

```
@XmlRootElement(name="Response",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightResponse {

    @XmlAttribute(name="TRIPNUMBER")
    private String tripNumber;

    @XmlAttribute(name="TICKET_PRICE")
    private BigDecimal ticketPrice;
```

```
    @XmlAttribute(name="TICKET_TAX")
    private BigDecimal ticketTax;

    @XmlAttribute(name="CURRENCY")
    private String currency;

    @XmlAttribute(name="PASSFORM")
    private String passengerFormOfAddress;

    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @XmlAttribute(name="PASSBIRTH")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date passengerDateOfBirth;

    @XmlElement(name="FLTINFO")
    private FlightInfo flightInfo;

    @XmlElement(name="CONNINFO")
    private ConnectionInfoTable connectionInfo;

    ...
}
```

NOTE

The complex parameter fields of the response object are serialized as child elements of the response.

55.14.5. FlightInfo bean

The following listing illustrates a JAXB bean which marshals to the serialized form of the complex structure parameter **FLTINFO**:

```
@XmlRootElement(name="FLTINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class FlightInfo {

    @XmlAttribute(name="FLIGHTTIME")
    private String flightTime;

    @XmlAttribute(name="CITYFROM")
    private String cityFrom;

    @XmlAttribute(name="DEPDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date departureTime;
```

```
    @XmlAttribute(name="CITYTO")
    private String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date arrivalTime;

    ...
}
```

55.14.6. ConnectionInfoTable bean

The following listing illustrates a JAXB bean which marshals to the serialized form of the complex table parameter, **CONNINFO**. Note that the name of the root element type of the JAXB bean corresponds to the name of the row structure type suffixed with **_TABLE** and the bean contains a list of row elements.

```
@XmlRootElement(name="CONNINFO_TABLE",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfoTable {

    @XmlElement(name="row")
    List<ConnectionInfo> rows;

    ...
}
```

55.14.7. ConnectionInfo bean

The following listing illustrates a JAXB bean, which marshals to the serialized form of the above tables row elements:

```
@XmlRootElement(name="CONNINFO",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionInfo {

    @XmlAttribute(name="CONNID")
    String connectionId;

    @XmlAttribute(name="AIRLINE")
    String airline;

    @XmlAttribute(name="PLANETYPE")
    String planeType;

    @XmlAttribute(name="CITYFROM")
    String cityFrom;

    @XmlAttribute(name="DEPDATE")
```

```
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureDate;

    @XmlAttribute(name="DEPTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date departureTime;

    @XmlAttribute(name="CITYTO")
    String cityTo;

    @XmlAttribute(name="ARRDATE")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalDate;

    @XmlAttribute(name="ARRTIME")
    @XmlJavaTypeAdapter(DateAdapter.class)
    Date arrivalTime;

    ...
}
```

CHAPTER 56. SCHEDULER

Only consumer is supported

The Scheduler component is used to generate message exchanges when a scheduler fires. This component is similar to the Timer component, but it offers more functionality in terms of scheduling. Also this component uses JDK **ScheduledExecutorService**. Where as the timer uses a JDK **Timer**.

You can only consume events from this endpoint.

56.1. URI FORMAT

```
scheduler:name[?options]
```

Where **name** is the name of the scheduler, which is created and shared across endpoints. So if you use the same name for all your scheduler endpoints, only one scheduler thread pool and thread will be used - but you can configure the thread pool to allow more concurrent threads.

NOTE

The IN body of the generated exchange is **null**. So **exchange.getIn().getBody()** returns **null**.

56.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

56.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

56.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

56.3. COMPONENT OPTIONS

The Scheduler component supports 3 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
poolSize (scheduler)	Number of core threads in the thread pool used by the scheduling thread pool. Is by default using a single thread.	1	int

56.4. ENDPOINT OPTIONS

The Scheduler endpoint is configured using URI syntax:

```
scheduler:name
```

with the following path and query parameters:

56.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (consumer)	**Required** The name of the scheduler.		String

56.4.2. Query Parameters (21 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int

Name	Description	Default	Type
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
poolSize (scheduler)	Number of core threads in the thread pool used by the scheduling thread pool. Is by default using a single thread.	1	int
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService

Name	Description	Default	Type
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean

56.5. MORE INFORMATION

This component is a scheduler Polling Consumer where you can find more information about the options above, and examples at the Polling Consumer page.

56.6. EXCHANGE PROPERTIES

When the timer is fired, it adds the following information as properties to the **Exchange**:

Name	Type	Description
Exchange.TIMER_NAME	**String**	The value of the **name** option.
Exchange.TIMER_FIRED_TIME	**Date**	The time when the consumer fired.

56.7. SAMPLE

To set up a route that generates an event every 60 seconds:

```
from("scheduler://foo?delay=60000").to("bean:myBean?method=someMethodName");
```

The above route will generate an event and then invoke the **someMethodName** method on the bean called **myBean** in the Registry such as JNDI or Spring.

And the route in Spring DSL:

```
<route>
  <from uri="scheduler://foo?delay=60000"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

56.8. FORCING THE SCHEDULER TO TRIGGER IMMEDIATELY WHEN COMPLETED

To let the scheduler trigger as soon as the previous task is complete, you can set the option **greedy=true**. But beware then the scheduler will keep firing all the time. So use this with caution.

56.9. FORCING THE SCHEDULER TO BE IDLE

There can be use cases where you want the scheduler to trigger and be greedy. But sometimes you want "tell the scheduler" that there was no task to poll, so the scheduler can change into idle mode using the backoff options. To do this you would need to set a property on the exchange with the key **Exchange.SCHEDULER_POLLED_MESSAGES** to a boolean value of false. This will cause the consumer to indicate that there was no messages polled.

The consumer will otherwise as by default return 1 message polled to the scheduler, every time the consumer has completed processing the exchange.

56.10. SPRING BOOT AUTO-CONFIGURATION

When using scheduler with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

-

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-scheduler-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.scheduler.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.scheduler.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.scheduler.enabled	Whether to enable auto configuration of the scheduler component. This is enabled by default.		Boolean
camel.component.scheduler.pool-size	Number of core threads in the thread pool used by the scheduling thread pool. Is by default using a single thread.	1	Integer

CHAPTER 57. SEDA

Both producer and consumer are supported

The SEDA component provides asynchronous SEDA behavior, so that messages are exchanged on a BlockingQueue and consumers are invoked in a separate thread from the producer.

Note that queues are only visible within a *single* CamelContext. If you want to communicate across **CamelContext** instances (for example, communicating between Web applications), see the component.

This component does not implement any kind of persistence or recovery, if the VM terminates while messages are yet to be processed. If you need persistence, reliability or distributed SEDA, try using either JMS or ActiveMQ.

NOTE

Synchronous
The Direct component provides synchronous invocation of any consumers when a producer sends a message exchange.

57.1. URI FORMAT

```
seda:someName[?options]
```

Where **someName** can be any string that uniquely identifies the endpoint within the current CamelContext.

57.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

57.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

57.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

57.3. COMPONENT OPTIONS

The SEDA component supports 10 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Sets the default number of concurrent threads processing exchanges.	1	int
defaultPollTimeout (consumer (advanced))	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	int
defaultBlockWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	boolean
defaultDiscardWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	boolean

Name	Description	Default	Type
defaultOfferTimeout (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, where a configured timeout can be added to the block case. Utilizing the .offer(timeout) method of the underlining java queue.		long
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
defaultQueueFactory (advanced)	Sets the default queue factory.		BlockingQueueFactory
queueSize (advanced)	Sets the default maximum capacity of the SEDA queue (i.e., the number of messages it can hold).	1000	int

57.4. ENDPOINT OPTIONS

The SEDA endpoint is configured using URI syntax:

```
seda:name
```

with the following path and query parameters:

57.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** Name of queue.		String

57.4.2. Query Parameters (18 parameters)

Name	Description	Default	Type
size (common)	The maximum capacity of the SEDA queue (i.e., the number of messages it can hold). Will by default use the defaultSize set on the SEDA component.	1000	int
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Number of concurrent threads processing exchanges.	1	int
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
limitConcurrentConsumers (consumer (advanced))	Whether to limit the number of concurrentConsumers to the maximum of 500. By default, an exception will be thrown if an endpoint is configured with a greater number. You can disable that check by turning this option off.	true	boolean

Name	Description	Default	Type
multipleConsumers (consumer (advanced))	Specifies whether multiple consumers are allowed. If enabled, you can use SEDA for Publish-Subscribe messaging. That is, you can send a message to the SEDA queue and have each consumer receive a copy of the message. When enabled, this option should be specified on every consumer endpoint.	false	boolean
pollTimeout (consumer (advanced))	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	int
purgeWhenStopping (consumer (advanced))	Whether to purge the task queue when stopping the consumer/route. This allows to stop faster, as any pending messages on the queue is discarded.	false	boolean
blockWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	boolean
discardIfNoConsumers (producer)	Whether the producer should discard the message (do not add the message to the queue), when sending to a queue with no active consumers. Only one of the options discardIfNoConsumers and failIfNoConsumers can be enabled at the same time.	false	boolean
discardWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	boolean
failIfNoConsumers (producer)	Whether the producer should fail by throwing an exception, when sending to a queue with no active consumers. Only one of the options discardIfNoConsumers and failIfNoConsumers can be enabled at the same time.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
offerTimeout (producer)	Offer timeout (in milliseconds) can be added to the block case when queue is full. You can disable timeout by using 0 or a negative value.		long
timeout (producer)	Timeout (in milliseconds) before a SEDA producer will stop waiting for an asynchronous task to complete. You can disable timeout by using 0 or a negative value.	30000	long
waitForTaskToComplete (producer)	Option to specify whether the caller should wait for the async task to complete or not before continuing. The following three options are supported: Always, Never or IfReplyExpected. The first two values are self-explanatory. The last value, IfReplyExpected, will only wait if the message is Request Reply based. The default option is IfReplyExpected. Enum values: • Never • IfReplyExpected • Always	IfReplyExpected	WaitForTaskToComplete
queue (advanced)	Define the queue instance which will be used by the endpoint.		BlockingQueue

57.5. CHOOSING BLOCKINGQUEUE IMPLEMENTATION

By default, the SEDA component always intantiates LinkedBlockingQueue, but you can use different implementation, you can reference your own BlockingQueue implementation, in this case the size option is not used

```
<bean id="arrayQueue" class="java.util.ArrayBlockingQueue">
  <constructor-arg index="0" value="10" ><!-- size -->
  <constructor-arg index="1" value="true" ><!-- fairness -->
```

```
</bean>

<!-- ... and later -->
<from>seda:array?queue=#arrayQueue</from>
```

Or you can reference a BlockingQueueFactory implementation, 3 implementations are provided LinkedBlockingQueueFactory, ArrayBlockingQueueFactory and PriorityBlockingQueueFactory:

```
<bean id="priorityQueueFactory"
class="org.apache.camel.component.seda.PriorityBlockingQueueFactory">
  <property name="comparator">
    <bean class="org.apache.camel.demo.MyExchangeComparator" />
  </property>
</bean>

<!-- ... and later -->
<from>seda:priority?queueFactory=#priorityQueueFactory&size=100</from>
```

57.6. USE OF REQUEST REPLY

The SEDA component supports using Request Reply, where the caller will wait for the Async route to complete. For instance:

```
from("mina:tcp://0.0.0.0:9876?textline=true&sync=true").to("seda:input");

from("seda:input").to("bean:processInput").to("bean:createResponse");
```

In the route above, we have a TCP listener on port 9876 that accepts incoming requests. The request is routed to the **seda:input** queue. As it is a Request Reply message, we wait for the response. When the consumer on the **seda:input** queue is complete, it copies the response to the original message response.

57.7. CONCURRENT CONSUMERS

By default, the SEDA endpoint uses a single consumer thread, but you can configure it to use concurrent consumer threads. So instead of thread pools you can use:

```
from("seda:stageName?concurrentConsumers=5").process(...)
```

As for the difference between the two, note a *thread pool* can increase/shrink dynamically at runtime depending on load, whereas the number of concurrent consumers is always fixed.

57.8. THREAD POOLS

Be aware that adding a thread pool to a SEDA endpoint by doing something like:

```
from("seda:stageName").thread(5).process(...)
```

Can wind up with two **BlockQueues**: one from the SEDA endpoint, and one from the workqueue of the thread pool, which may not be what you want. Instead, you might wish to configure a Direct endpoint with a thread pool, which can process messages both synchronously and asynchronously. For example:

```
from("direct:stageName").thread(5).process(...)
```

You can also directly configure number of threads that process messages on a SEDA endpoint using the **concurrentConsumers** option.

57.9. SAMPLE

In the route below we use the SEDA queue to send the request to this async queue to be able to send a fire-and-forget message for further processing in another thread, and return a constant reply in this thread to the original caller.

We send a Hello World message and expects the reply to be OK.

```
@Test
public void testSendAsync() throws Exception {
    MockEndpoint mock = getMockEndpoint("mock:result");
    mock.expectedBodiesReceived("Hello World");

    // START SNIPPET: e2
    Object out = template.requestBody("direct:start", "Hello World");
    assertEquals("OK", out);
    // END SNIPPET: e2

    assertMockEndpointsSatisfied();
}

@Override
protected RouteBuilder createRouteBuilder() throws Exception {
    return new RouteBuilder() {
        // START SNIPPET: e1
        public void configure() throws Exception {
            from("direct:start")
                // send it to the seda queue that is async
                .to("seda:next")
                // return a constant response
                .transform(constant("OK"));

            from("seda:next").to("mock:result");
        }
        // END SNIPPET: e1
    };
}
```

The "Hello World" message will be consumed from the SEDA queue from another thread for further processing. Since this is from a unit test, it will be sent to a **mock** endpoint where we can do assertions in the unit test.

57.10. USING MULTIPLECONSUMERS

In this example we have defined two consumers.

```
@Test
public void testSameOptionsProducerStillOkay() throws Exception {
    getMockEndpoint("mock:foo").expectedBodiesReceived("Hello World");
```

```
        getMockEndpoint("mock:bar").expectedBodiesReceived("Hello World");

        template.sendBody("seda:foo", "Hello World");

        assertMockEndpointsSatisfied();
    }

    @Override
    protected RouteBuilder createRouteBuilder() throws Exception {
        return new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                from("seda:foo?multipleConsumers=true").routeId("foo").to("mock:foo");
                from("seda:foo?multipleConsumers=true").routeId("bar").to("mock:bar");
            }
        };
    }
```

Since we have specified **multipleConsumers=true** on the seda foo endpoint we can have those two consumers receive their own copy of the message as a kind of pub-sub style messaging.

As the beans are part of an unit test they simply send the message to a mock endpoint.

57.11. EXTRACTING QUEUE INFORMATION

If needed, information such as queue size, etc. can be obtained without using JMX in this fashion:

```
SedaEndpoint seda = context.getEndpoint("seda:xxxx");
int size = seda.getExchanges().size();
```

57.12. SPRING BOOT AUTO-CONFIGURATION

When using seda with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-seda-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.component.seda.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.seda.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.seda.concurrent-consumers	Sets the default number of concurrent threads processing exchanges.	1	Integer
camel.component.seda.default-block-when-full	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	Boolean
camel.component.seda.default-discard-when-full	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	Boolean
camel.component.seda.default-offer-timeout	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, where a configured timeout can be added to the block case. Utilizing the .offer(timeout) method of the underlining java queue.		Long
camel.component.seda.default-poll-timeout	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	Integer
camel.component.seda.default-queue-factory	Sets the default queue factory. The option is a org.apache.camel.component.seda.BlockingQueueFactory<org.apache.camel.Exchange> type.		BlockingQueueFactory
camel.component.seda.enabled	Whether to enable auto configuration of the seda component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.seda.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.seda.queue-size	Sets the default maximum capacity of the SEDA queue (i.e., the number of messages it can hold).	1000	Integer

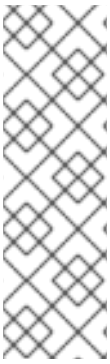
CHAPTER 58. SERVLET

Only consumer is supported

The Servlet component provides HTTP based endpoints for consuming HTTP requests that arrive at a HTTP endpoint that is bound to a published Servlet.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-servlet</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```



NOTE

Stream

Servlet is stream based, which means the input it receives is submitted to Camel as a stream. That means you will only be able to read the content of the stream **once**. If you find a situation where the message body appears to be empty or you need to access the data multiple times (eg: doing multicasting, or redelivery error handling) you should use Stream caching or convert the message body to a **String** which is safe to be read multiple times.

58.1. URI FORMAT

```
servlet://relative_path[?options]
```

58.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

58.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre-configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

58.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a *type safe* way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

58.3. COMPONENT OPTIONS

The Servlet component supports 11 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which means any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
muteException (consumer)	If enabled and an Exchange failed processing on the consumer side the response's body won't contain the exception's stack trace.	false	boolean
servletName (consumer)	Default name of servlet to use. The default name is CamelServlet.	Camel Servlet	String
attachmentMultipartBinding (consumer (advanced))	Whether to automatic bind multipart/form-data as attachments on the Camel Exchange. The options attachmentMultipartBinding=true and disableStreamCache=false cannot work together. Remove disableStreamCache to use AttachmentMultipartBinding. This is turned off by default as this may require servlet specific configuration to enable this when using Servlets.	false	boolean
fileNameExtWhitelist (consumer (advanced))	Whitelist of accepted filename extensions for accepting uploaded files. Multiple extensions can be separated by comma, such as txt,xml.		String

Name	Description	Default	Type
httpRegistry (consumer (advanced))	To use a custom org.apache.camel.component.servlet.HttpRegistry.		HttpRegistry
allowJavaSerializedObject (advanced)	Whether to allow java serialization when a request uses context-type=application/x-java-serialized-object. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
httpBinding (advanced)	To use a custom HttpBinding to control the mapping between Camel message and HttpClient.		HttpBinding
httpConfiguration (advanced)	To use the shared HttpConfiguration as base configuration.		HttpConfiguration
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy

58.4. ENDPOINT OPTIONS

The Servlet endpoint is configured using URI syntax:

```
servlet:contextPath
```

with the following path and query parameters:

58.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
contextPath (consumer)	**Required** The context-path to use.		String

58.4.2. Query Parameters (22 parameters)

Name	Description	Default	Type
chunked (consumer)	If this option is false the Servlet will disable the HTTP streaming and set the content-length header on the response.	true	boolean
disableStreamCache (common)	Determines whether or not the raw input stream from Servlet is cached or not (Camel will read the stream into a in memory/overflow to file, Stream caching) cache. By default Camel will cache the Servlet input stream to support reading it multiple times to ensure Camel can retrieve all data from the stream. However you can set this option to true when you for example need to access the raw stream, such as streaming it directly to a file or other persistent store. DefaultHttpBinding will copy the request input stream into a stream cache and put it into message body if this option is false to support reading the stream multiple times. If you use Servlet to bridge/proxy an endpoint then consider enabling this option to improve performance, in case you do not need to read the message payload multiple times. The http producer will by default cache the response body stream. If this option is set to true, then the producers will not cache the response body stream but use the response stream as-is as the message body.	false	boolean
headerFilterStrategy (common)	To use a custom HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy
httpBinding (common (advanced))	To use a custom HttpBinding to control the mapping between Camel message and HttpClient.		HttpBinding
async (consumer)	Configure the consumer to work in async mode.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which means any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
httpMethodRestrict (consumer)	Used to only allow consuming if the HttpMethod matches, such as GET/POST/PUT etc. Multiple methods can be specified separated by comma.		String
matchOnUriPrefix (consumer)	Whether or not the consumer should try to find a target consumer by matching the URI prefix if no exact match is found.	false	boolean
muteException (consumer)	If enabled and an Exchange failed processing on the consumer side the response's body won't contain the exception's stack trace.	false	boolean
responseBufferSize (consumer)	To use a custom buffer size on the javax.servlet.ServletResponse.		Integer
servletName (consumer)	Name of the servlet to use.	Camel Servlet	String
transferException (consumer)	If enabled and an Exchange failed processing on the consumer side, and if the caused Exception was sent back serialized in the response as an application/x-java-serialized-object content type. On the producer side the exception will be deserialized and thrown as is, instead of the HttpOperationFailedException. The caused exception is required to be serialized. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	boolean
attachmentMultipartBinding (consumer (advanced))	Whether to automatic bind multipart/form-data as attachments on the Camel Exchange. The options attachmentMultipartBinding=true and disableStreamCache=false cannot work together. Remove disableStreamCache to use AttachmentMultipartBinding. This is turned off by default as this may require servlet specific configuration to enable this when using Servlets.	false	boolean
eagerCheckContentAvailable (consumer (advanced))	Whether to eager check whether the HTTP requests has content if the content-length header is 0 or not present. This can be turned on in case HTTP clients do not send streamed data.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
fileNameExtWhitelist (consumer (advanced))	Whitelist of accepted filename extensions for accepting uploaded files. Multiple extensions can be separated by comma, such as txt,xml.		String
mapHttpMessageBody (consumer (advanced))	If this option is true then IN exchange Body of the exchange will be mapped to HTTP body. Setting this to false will avoid the HTTP mapping.	true	boolean
mapHttpMessageFormUrlEncodedBody (consumer (advanced))	If this option is true then IN exchange Form Encoded body of the exchange will be mapped to HTTP. Setting this to false will avoid the HTTP Form Encoded body mapping.	true	boolean
mapHttpMessageHeaders (consumer (advanced))	If this option is true then IN exchange Headers of the exchange will be mapped to HTTP headers. Setting this to false will avoid the HTTP Headers mapping.	true	boolean
optionsEnabled (consumer (advanced))	Specifies whether to enable HTTP OPTIONS for this Servlet consumer. By default OPTIONS is turned off.	false	boolean
traceEnabled (consumer (advanced))	Specifies whether to enable HTTP TRACE for this Servlet consumer. By default TRACE is turned off.	false	boolean

58.5. MESSAGE HEADERS

Camel will apply the same Message Headers as the HTTP component.

Camel will also populate **all request.parameter** and **request.headers**. For example, if a client request has the URL, http://myserver/myserver?orderid=123, the exchange will contain a header named **orderid** with the value 123.

58.6. USAGE

You can consume only **from** endpoints generated by the Servlet component. Therefore, it should be used only as input into your Camel routes. To issue HTTP requests against other HTTP endpoints, use the HTTP component.

58.7. SPRING BOOT AUTO-CONFIGURATION

When using servlet with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-servlet-starter</artifactId>
</dependency>
```

The component supports 15 options, which are listed below.

Name	Description	Default	Type
camel.component.servlet.allow-java-serialized-object	Whether to allow java serialization when a request uses context-type=application/x-java-serialized-object. This is by default turned off. If you enable this then be aware that Java will deserialize the incoming data from the request to Java and that can be a potential security risk.	false	Boolean
camel.component.servlet.attachment-multipart-binding	Whether to automatic bind multipart/form-data as attachments on the Camel Exchange. The options attachmentMultipartBinding=true and disableStreamCache=false cannot work together. Remove disableStreamCache to use AttachmentMultipartBinding. This is turned off by default as this may require servlet specific configuration to enable this when using Servlet's.	false	Boolean
camel.component.servlet.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.servlet.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which means any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean

Name	Description	Default	Type
camel.component.servlet.enabled	Whether to enable auto configuration of the servlet component. This is enabled by default.		Boolean
camel.component.servlet.file-name-ext-whitelist	Whitelist of accepted filename extensions for accepting uploaded files. Multiple extensions can be separated by comma, such as txt,xml.		String
camel.component.servlet.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.servlet.http-binding	To use a custom HttpBinding to control the mapping between Camel message and HttpClient. The option is a org.apache.camel.http.common.HttpBinding type.		HttpBinding
camel.component.servlet.http-configuration	To use the shared HttpConfiguration as base configuration. The option is a org.apache.camel.http.common.HttpConfiguration type.		HttpConfiguration
camel.component.servlet.http-registry	To use a custom org.apache.camel.component.servlet.HttpRegistry. The option is a org.apache.camel.http.common.HttpRegistry type.		HttpRegistry
camel.component.servlet.mute-exception	If enabled and an Exchange failed processing on the consumer side the response's body won't contain the exception's stack trace.	false	Boolean
camel.component.servlet.servlet-name	Default name of servlet to use. The default name is CamelServlet.	Camel Servlet	String
camel.servlet.mapping.context-path	Context path used by the servlet component for automatic mapping.	/camel/*	String
camel.servlet.mapping.enabled	Enables the automatic mapping of the servlet component into the Spring web context.	true	Boolean
camel.servlet.mapping.servlet-name	The name of the Camel servlet.	Camel Servlet	String

CHAPTER 59. SLACK

Both producer and consumer are supported

The Slack component allows you to connect to an instance of Slack and delivers a message contained in the message body via a pre established Slack incoming webhook.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-slack</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

59.1. URI FORMAT

To send a message to a channel.

```
slack:#channel[?options]
```

To send a direct message to a slackuser.

```
slack:@userID[?options]
```

59.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

59.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

59.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

59.3. COMPONENT OPTIONS

The Slack component supports 5 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
token (token)	The token to use.		String
webhookUrl (webhook)	The incoming webhook URL.		String

59.4. ENDPOINT OPTIONS

The Slack endpoint is configured using URI syntax:

```
slack:channel
```

with the following path and query parameters:

59.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
channel (common)	**Required** The channel name (syntax #name) or slackuser (syntax userName) to send a message directly to an user.		String

59.4.2. Query Parameters (29 parameters)

Name	Description	Default	Type
token (common)	The token to use.		String
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
conversationType (consumer)	Type of conversation. Enum values: • PUBLIC_CHANNEL • PRIVATE_CHANNEL • MPIM • IM	PUBLIC_CHANNEL	ConversationType
maxResults (consumer)	The Max Result for the poll.	10	String
naturalOrder (consumer)	Create exchanges in natural order (oldest to newest) or not.	false	boolean

Name	Description	Default	Type
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
serverUrl (consumer)	The Server URL of the Slack instance.		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
iconEmoji (producer)	**Deprecated** Use a Slack emoji as an avatar.		String
iconUrl (producer)	**Deprecated** The avatar that the component will use when sending message to a channel or user.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
username (producer)	**Deprecated** This is the username that the bot will have when sending messages to a channel or user.		String
webhookUrl (producer)	The incoming webhook URL.		String
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long

Name	Description	Default	Type
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit

Name	Description	Default	Type
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean

59.5. CONFIGURING IN SPRINT XML

The Slack component with XML must be configured as a Spring or Blueprint bean that contains the incoming webhook url or the app token for the integration as a parameter.

```
<bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
    <property name="webhookUrl"
value="https://hooks.slack.com/services/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"/>
    <property name="token" value="xoxb-12345678901-1234567890123-
xxxxxxxxxxxxxxxxxxxxxxxx"/>
</bean>
```

For Java you can configure this using Java code.

59.6. EXAMPLE

A CamelContext with Blueprint could be as:

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" default-activation="lazy">

    <bean id="slack" class="org.apache.camel.component.slack.SlackComponent">
        <property name="webhookUrl"
value="https://hooks.slack.com/services/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"/>
    </bean>

    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="direct:test"/>
            <to uri="slack:#channel?iconEmoji=:camel:&amp;username=CamelTest"/>
        </route>
    </camelContext>

</blueprint>
```

59.7. PRODUCER

You can now use a token to send a message instead of WebhookUrl.

```
from("direct:test")
    .to("slack:#random?token=RAW(<YOUR_TOKEN>)");
```

You can now use the Slack API model to create blocks. You can read more about it here https://api.slack.com/block-kit.

```
    public void testSlackAPIModelMessage() {
        Message message = new Message();
        message.setBlocks(Collections.singletonList(SectionBlock
                .builder()
                .text(MarkdownTextObject
                        .builder()
                        .text("*Hello from Camel!*")
                        .build())
                .build()));

        template.sendBody(test, message);
    }
```

59.8. CONSUMER

You can use also a consumer for messages in channel.

```
from("slack://general?token=RAW(<YOUR_TOKEN>)&maxResults=1")
    .to("mock:result");
```

In this way you'll get the last message from general channel. The consumer will take track of the timestamp of the last message consumed and in the next poll it will check from that timestamp.

You'll need to create a Slack app and use it on your workspace.

Use the 'Bot User OAuth Access Token' as token for the consumer endpoint.

NOTE

Add the corresponding history (**channels:history** or **groups:history** or **mpim:history** or **im:history**) and read (**channels:read** or **groups:read** or **mpim:read** or **im:read**) user token scope to your app to grant it permission to view messages in the corresponding channel. You will need to use the conversationType option to set it up too (**PUBLIC_CHANNEL**, **PRIVATE_CHANNEL**, **MPIM**, **IM**)

The naturalOrder option allows consuming messages from the oldest to the newest. Originally you would get the newest first and consume backward (message 3 ⇒ message 2 ⇒ message 1)

NOTE

You can use the conversationType option to read history and messages from a channel that is not only public (**PUBLIC_CHANNEL**,**PRIVATE_CHANNEL**, **MPIM**, **IM**)

59.9. SPRING BOOT AUTO-CONFIGURATION

When using slack with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-slack-starter</artifactId>
</dependency>
```

The component supports 6 options, which are listed below.

Name	Description	Default	Type
camel.component.slack.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.slack.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.slack.enabled	Whether to enable auto configuration of the slack component. This is enabled by default.		Boolean
camel.component.slack.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.slack.token	The token to use.		String
camel.component.slack.webhook-url	The incoming webhook URL.		String

CHAPTER 60. SPRING BATCH

Since Camel 2.10

Only producer is supported

The Spring Batch component and support classes provide integration bridge between Camel and Spring Batch infrastructure.

Maven users must add the following dependency to their **pom.xml** for this component:

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

60.1. URI FORMAT

```
spring-batch:jobName[?options]
```

jobName represents the name of the Spring Batch job located in the Camel registry. If a JobRegistry is provided is used to locate the job.

This component is only used to define producer endpoints, that means you cannot use the Spring Batch component in a **from()** statement.

60.2. CONFIGURING OPTIONS

Camel components are configured on two levels:

- Component level
- Endpoint level

60.2.1. Component Level Options

The **component level** is the highest level. The configurations you define at this level are inherited by all the endpoints. For example, a component can have security settings, credentials for authentication, urls for network connection, and so on.

Since components typically have pre-configured defaults for the most common cases, you may need to only configure a few component options, or maybe none at all.

You can configure components with Component DSL in a configuration file (application.properties|yaml), or directly with Java code.

60.2.2. Endpoint Level Options

At the **Endpoint level** you have many options, which you can use to configure what you want the endpoint to do. The options are categorized according to whether the endpoint is used as a consumer (from) or as a producer (to) or used for both.

You can configure endpoints directly in the endpoint URI as **path** and **query** parameters. You can also use Endpoint DSL and DataFormat DSL as *type safe* ways of configuring endpoints and data formats in Java.

When configuring options, use Property Placeholders for urls, port numbers, sensitive information, and other settings.

Placeholders allows you to externalize the configuration from your code, giving you more flexible and reusable code.

60.3. COMPONENT OPTIONS

The Spring Batch component supports 4 options, which are listed below.

Name	Description	Default	Type
jobLauncher (producer)	Explicitly specifies a JobLauncher to be used.		JobLauncher
jobRegistry (producer)	Explicitly specifies a JobRegistry to be used.		JobRegistry
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

60.4. ENDPOINT OPTIONS

The Spring Batch endpoint is configured using URI syntax:

```
spring-batch:jobName
```

Following are the path and query parameters:

60.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
jobName (producer)	**Required** The name of the Spring Batch job located in the registry.		String

60.4.2. Query Parameters (4 parameters)

Name	Description	Default	Type
jobFromHeader (producer)	Explicitly defines if the jobName should be taken from the headers instead of the URI.	false	boolean
jobLauncher (producer)	Explicitly specifies a JobLauncher to be used.		JobLauncher

Name	Description	Default	Type
jobRegistry (producer)	Explicitly specifies a JobRegistry to be used.		JobRegistry
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

60.5. USAGE

When Spring Batch component receives the message, it triggers the job execution. The job is executed using the **org.springframework.batch.core.launch.JobLaucher** instance resolved according to the following algorithm.

- if **JobLauncher** is manually set on the component, then use it.
- if **jobLauncherRef** option is set on the component, then search Camel Registry for the **JobLauncher** with the given name.
- if there is **JobLauncher** registered in the Camel Registry under **jobLauncher** name, then use it.
- if none of the steps above allow to resolve the **JobLauncher** and there is exactly one **JobLauncher** instance in the Camel Registry, then use it.

All headers found in the message are passed to the **JobLauncher** as job parameters. **String**, **Long**, **Double** and **java.util.Date** values are copied to the **org.springframework.batch.core.JobParametersBuilder** and other data types are converted to Strings.

60.6. EXAMPLES

Triggering the Spring Batch job execution:

```
from("direct:startBatch").to("spring-batch:myJob");
```

Triggering the Spring Batch job execution with the **JobLauncher** set explicitly.

```
from("direct:startBatch").to("spring-batch:myJob?jobLauncherRef=myJobLauncher");
```

A **JobExecution** instance returned by the **JobLauncher** is forwarded by the **SpringBatchProducer** as the output message. You can use the **JobExecution** instance to perform some operations using the Spring Batch API directly.

```
from("direct:startBatch").to("spring-batch:myJob").to("mock:JobExecutions");
...
MockEndpoint mockEndpoint = ...;
JobExecution jobExecution =
mockEndpoint.getExchanges().get(0).getIn().getBody(JobExecution.class);
BatchStatus currentJobStatus = jobExecution.getStatus();
```

60.7. SUPPORT CLASSES

Apart from the component, Camel Spring Batch also provides support classes that you can use to hook into Spring Batch infrastructure.

60.7.1. CamelItemReader

CamelItemReader can be used to read batch data directly from the Camel infrastructure.

For example, the snippet below configures Spring Batch to read data from JMS queue:

```
<bean id="camelReader"
class="org.apache.camel.component.spring.batch.support.CamelItemReader">
  <constructor-arg ref="consumerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="camelReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

60.7.2. CamelItemWriter

CamelItemWriter has similar purpose as **CamelItemReader**, but it is dedicated to write chunk of the processed data.

For example the snippet below configures Spring Batch to read data from JMS queue.

```
<bean id="camelwriter"
class="org.apache.camel.component.spring.batch.support.CamelItemWriter">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:dataQueue"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="camelwriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

60.7.3. CamelItemProcessor

CamelItemProcessor is the implementation of Spring Batch **org.springframework.batch.item.ItemProcessor** interface. The latter implementation relays on Request Reply pattern to delegate the processing of the batch item to the Camel infrastructure. The item to process is sent to the Camel endpoint as the body of the message.

For example the snippet below performs simple processing of the batch item using the Direct endpoint and the Simple expression language.

```
<camel:camelContext>
  <camel:route>
    <camel:from uri="direct:processor"/>
    <camel:setExchangePattern pattern="InOut"/>
    <camel:setBody>
      <camel:simple>Processed ${body}</camel:simple>
    </camel:setBody>
  </camel:route>
</camel:camelContext>

<bean id="camelProcessor"
class="org.apache.camel.component.spring.batch.support.CamelItemProcessor">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="direct:processor"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" processor="camelProcessor" commit-
interval="100"/>
    </batch:tasklet>
  </batch:step>
</batch:job>
```

60.7.4. CamelJobExecutionListener

CamelJobExecutionListener is the implementation of the **org.springframework.batch.core.JobExecutionListener** interface sending job execution events to the Camel endpoint.

The **org.springframework.batch.core.JobExecution** instance produced by the Spring Batch is sent as a body of the message. To distinguish between before- and after-callbacks **SPRING_BATCH_JOB_EVENT_TYPE** header is set to the **BEFORE** or **AFTER** value.

The example snippet below sends Spring Batch job execution events to the JMS queue.

```
<bean id="camelJobExecutionListener"
class="org.apache.camel.component.spring.batch.support.CamelJobExecutionListener">
  <constructor-arg ref="producerTemplate"/>
  <constructor-arg value="jms:batchEventsBus"/>
</bean>

<batch:job id="myJob">
  <batch:step id="step">
    <batch:tasklet>
      <batch:chunk reader="someReader" writer="someWriter" commit-interval="100"/>
    </batch:tasklet>
  </batch:step>
  <batch:listeners>
    <batch:listener ref="camelJobExecutionListener"/>
  </batch:listeners>
</batch:job>
```

60.8. SPRING BOOT AUTO-CONFIGURATION

When using **spring-batch** with Spring Boot, use the following Maven dependency to enable support for auto-configuration:

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

The component supports 5 options, which are listed below.

Name	Description	Default	Type

Name	Description	Default	Type
camel.component.spring-batch.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.spring-batch.enabled	Whether to enable auto-configuration of the spring-batch component. This is enabled by default.		Boolean
camel.component.spring-batch.job-launcher	Explicitly specifies a JobLauncher to be used. The option is a org.springframework.batch.core.launch.JobLauncher type.		JobLauncher
camel.component.spring-batch.job-registry	Explicitly specifies a JobRegistry to be used. The option is a org.springframework.batch.core.configuration.JobRegistry type.		JobRegistry

Name	Description	Default	Type
camel.component.spring-batch.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 61. SPRING JDBC

Since Camel 3.10

Only producer is supported

The Spring JDBC component is an extension of the JDBC component with one additional feature to integrate with Spring Transaction Manager.

For general use of this component then see the JDBC Component.

Maven users must add the following dependency to their **pom.xml** for this component:

```
<dependency>
      <groupId>org.apache.camel.springboot</groupId>
      <artifactId>camel-spring-jdbc-starter</artifactId>
</dependency>
```

The version is specified using BOM in the following way.

```
<dependencyManagement>
      <dependencies>
        <dependency>
          <groupId>com.redhat.camel.springboot.platform</groupId>
          <artifactId>camel-spring-boot-bom</artifactId>
          <version>${camel-spring-boot-version}</version>
          <type>pom</type>
          <scope>import</scope>
        </dependency>
      </dependencies>
  </dependencyManagement>
```

61.1. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

```
<dependencyManagement>
      <dependencies>
        <dependency>
          <groupId>com.redhat.camel.springboot.platform</groupId>
          <artifactId>camel-spring-boot-bom</artifactId>
          <version>${camel-spring-boot-version}</version>
          <type>pom</type>
          <scope>import</scope>
        </dependency>
      </dependencies>
  </dependencyManagement>
```

61.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

61.3. CONFIGURING OPTIONS

Camel components are configured on two levels:

- Component level
- Endpoint level

61.3.1. Component Level Options

The **component level** is the highest level. The configurations you define at this level are inherited by all the endpoints. For example, a component can have security settings, credentials for authentication, urls for network connection, and so on.

Since components typically have pre-configured defaults for the most common cases, you may need to only configure a few component options, or maybe none at all.

You can configure components with Component DSL in a configuration file (application.properties|yaml), or directly with Java code.

61.3.2. Endpoint Level Options

At the **Endpoint level** you have many options, which you can use to configure what you want the endpoint to do. The options are categorized according to whether the endpoint is used as a consumer (from) or as a producer (to) or used for both.

You can configure endpoints directly in the endpoint URI as **path** and **query** parameters. You can also use Endpoint DSL and DataFormat DSL as *type safe* ways of configuring endpoints and data formats in Java.

When configuring options, use Property Placeholders for urls, port numbers, sensitive information, and other settings.

Placeholders allows you to externalize the configuration from your code, giving you more flexible and reusable code.

61.4. COMPONENT OPTIONS

The Spring JDBC component supports 4 options that are listed below.

Name	Description	Default	Type
dataSource (producer)	To use the DataSource instance instead of looking up the data source by name from the registry.		DataSource

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
connectionStrategy (advanced)	To use a custom strategy for working with connections. Do not use a custom strategy when using the spring-jdbc component because a special Spring ConnectionStrategy is used by default to support Spring Transactions.		ConnectionStrategy

61.5. ENDPOINT OPTIONS

The Spring JDBC endpoint is configured using URI syntax:

```
spring-jdbc:dataSourceName
```

Following are the path and query parameters:

61.5.1. Path Parameters (1 parameters)

Name	Description	Default	Type
dataSourceName (producer)	**Required** Name of DataSource to lookup in the Registry. If the name is dataSource or default, then Camel will attempt to lookup a default DataSource from the registry, meaning if there is a only one instance of DataSource found, then this DataSource will be used.		String

61.5.2. Query Parameters (14 parameters)

Name	Description	Default	Type
allowNamedParameters (producer)	Whether to allow using named parameters in the queries.	true	boolean

Name	Description	Default	Type
outputClass (producer)	Specify the full package and class name to use as conversion when outputType=SelectOne or SelectList.		String
outputType (producer)	Determines the output the producer should use. Enum values: • SelectOne • SelectList • StreamList	SelectList	JdbcOutputType
parameters (producer)	Optional parameters to the java.sql.Statement. For example to set maxRows, fetchSize etc.		Map
readSize (producer)	The default maximum number of rows that can be read by a polling query. The default value is 0.		int

Name	Description	Default	Type
resetAutoCommit (producer)	Camel will set the autoCommit on the JDBC connection to be false, commit the change after executed the statement and reset the autoCommit flag of the connection at the end, if the resetAutoCommit is true. If the JDBC connection doesn't support to reset the autoCommit flag, you can set the resetAutoCommit flag to be false, and Camel will not try to reset the autoCommit flag. When used with XA transactions you most likely need to set it to false so that the transaction manager is in charge of committing this tx.	true	boolean
transacted (producer)	Whether transactions are in use.	false	boolean
useGetBytesForBlob (producer)	To read BLOB columns as bytes instead of string data. This may be needed for certain databases such as Oracle where you must read BLOB columns as bytes.	false	boolean
useHeadersAsParameters (producer)	Set this option to true to use the prepareStatementStrategy with named parameters. This allows to define queries with named placeholders, and use headers with the dynamic values for the query placeholders.	false	boolean

Name	Description	Default	Type
useJDBC4ColumnNameAndLabelSemantics (producer)	Sets whether to use JDBC 4 or JDBC 3.0 or older semantic when retrieving column name. JDBC 4.0 uses columnLabel to get the column name where as JDBC 3.0 uses both columnName or columnLabel. Unfortunately JDBC drivers behave differently so you can use this option to work out issues around your JDBC driver if you get problem using this component This option is default true.	true	boolean
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
beanRowMapper (advanced)	To use a custom org.apache.camel.component.jdbc.BeanRowMapper when using outputClass. The default implementation will lower case the row names and skip underscores, and dashes. For example CUST_ID is mapped as custId.		BeanRowMapper
connectionStrategy (advanced)	To use a custom strategy for working with connections. Do not use a custom strategy when using the spring-jdbc component because a special Spring ConnectionStrategy is used by default to support Spring Transactions.		ConnectionStrategy
prepareStatementStrategy (advanced)	Allows the plugin to use a custom org.apache.camel.component.jdbc.JdbcPrepareStatementStrategy to control preparation of the query and prepared statement.		JdbcPrepareStatementStrategy

61.6. SPRING BOOT AUTO-CONFIGURATION

When using spring-jdbc with Spring Boot use the following Maven dependency to have support for auto configuration:

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

The component supports 4 options that are listed below.

Name	Description	Default	Type
camel.component.spring-jdbc.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.spring-jdbc.connection-strategy	To use a custom strategy for working with connections. Do not use a custom strategy when using the spring-jdbc component because a special Spring ConnectionStrategy is used by default to support Spring Transactions. The option is a org.apache.camel.component.jdbc.ConnectionStrategy type.		ConnectionStrategy
camel.component.spring-jdbc.enabled	Whether to enable auto configuration of the spring-jdbc component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.spring-jdbc.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 62. SPRING LDAP

Since Camel 2.11

Only producer is supported

The Spring LDAP component provides a Camel wrapper for Spring LDAP.

Maven users must add the following dependency to their **pom.xml** for this component:

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

62.1. URI FORMAT

```
spring-ldap:springLdapTemplate[?options]
```

Where **springLdapTemplate** is the name of the Spring LDAP Template bean. In this bean, you configure the URL and the credentials for your LDAP access.

62.2. CONFIGURING OPTIONS

Camel components are configured on two levels:

- Component level
- Endpoint level

62.2.1. Component Level Options

The **component level** is the highest level. The configurations you define at this level are inherited by all the endpoints. For example, a component can have security settings, credentials for authentication, urls for network connection, and so on.

Since components typically have pre-configured defaults for the most common cases, you may need to only configure a few component options, or maybe none at all.

You can configure components with Component DSL in a configuration file (application.properties|yaml), or directly with Java code.

62.2.2. Endpoint Level Options

At the **Endpoint level** you have many options, which you can use to configure what you want the endpoint to do. The options are categorized according to whether the endpoint is used as a consumer (from) or as a producer (to) or used for both.

You can configure endpoints directly in the endpoint URI as **path** and **query** parameters. You can also use Endpoint DSL and DataFormat DSL as *type safe* ways of configuring endpoints and data formats in Java.

When configuring options, use Property Placeholders for urls, port numbers, sensitive information, and other settings.

Placeholders allows you to externalize the configuration from your code, giving you more flexible and reusable code.

62.3. COMPONENT OPTIONS

The Spring LDAP component supports 2 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

62.4. ENDPOINT OPTIONS

The Spring LDAP endpoint is configured using URI syntax:

```
spring-ldap:templateName
```

Following are the path and query parameters:

62.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
templateName (producer)	**Required** Name of the Spring LDAP Template bean.		String

62.4.2. Query Parameters (3 parameters)

Name	Description	Default	Type
operation (producer)	**Required** The LDAP operation to be performed. Enum values: • SEARCH • BIND • UNBIND • AUTHENTICATE • MODIFY_ATTRIBUTES • FUNCTION_DRIVEN		LdapOperation
scope (producer)	The scope of the search operation. Enum values: • object • onelevel • subtree	subtree	String

Name	Description	Default	Type
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

62.5. USAGE

The component supports producer endpoints only. An attempt to create a consumer endpoint can result in an **UnsupportedOperationException**.
The body of the message must be a map (an instance of **java.util.Map**). Unless a base DN is specified in the configuration of your ContextSource, this map must contain at least an entry with the key **dn** (not needed for function_driven operation) that specifies the root node for the LDAP operation to be performed. Other entries of the map are operation-specific.

The body of the message remains unchanged for the **bind** and **unbind** operations. For the **search** and **function_driven** operations, the body is set to the result of the search, see http://static.springsource.org/spring-ldap/site/apidocs/org/springframework/ldap/core/LdapTemplate.html#search%28java.lang.String,%20ja

62.5.1. Search

The message body must have an entry with the key **filter**. The value must be a **String** representing a valid LDAP filter, see http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Search_and_Compare.

62.5.2. Bind

The message body must have an entry with the key **attributes**. The value must be an instance of javax.naming.directory.Attributes This entry specifies the LDAP node to be created.

62.5.3. Unbind

No further entries are necessary, the node with the specified **dn** is deleted.

62.5.4. Authenticate

The message body must have entries with the keys **filter** and **password**. The values must be an instance of **String** representing a valid LDAP filter and a user password, respectively.

62.5.5. Modify Attributes

The message body must have an entry with the key **modificationItems**. The value must be an instance of any array of type javax.naming.directory.ModificationItem

62.5.6. Function-Driven

The message body must have entries with the keys **function** and **request**. The **function** value must be of type **java.util.function.BiFunction<L, Q, S>**. The **L** type parameter must be of type **org.springframework.ldap.core.LdapOperations**. The **request** value must be the same type as the **Q** type parameter in the **function** and it must encapsulate the parameters expected by the LdapTemplate method being invoked within the **function**. The **S** type parameter represents the response type as returned by the LdapTemplate method being invoked. This operation allows dynamic invocation of LdapTemplate methods that are not covered by the operations mentioned above.

Key definitions

In order to avoid spelling errors, the following constants are defined in **org.apache.camel.springldap.SpringLdapProducer**:

- public static final String DN = "dn"
- public static final String FILTER = "filter"
- public static final String ATTRIBUTES = "attributes"
- public static final String PASSWORD = "password";
- public static final String MODIFICATION_ITEMS = "modificationItems";
- public static final String FUNCTION = "function";
- public static final String REQUEST = "request";

Following is an example of createMap function:

```
from(“direct:start”)
    .setBody(constant(createMap()))
    .to("spring-ldap:ldapTemplate?operation=BIND");
```

Here, createMap function returns Map object that contains information about attributes and domain name of ldap server.

```
private static Map<String, Object> createMap() {
    BasicAttributes basicAttributes = new BasicAttributes();
    basicAttributes.put("cn", "Name Surname");
```

```
    basicAttributes.put("sn", "Surname");
    basicAttributes.put("objectClass", "person");
    Map<String, Object> map = new HashMap<>();
    map.put(SpringLdapProducer.DN, "cn=LdapDN,dc=example,dc=org");
    map.put(SpringLdapProducer.ATTRIBUTES, basicAttributes);
    return map;
}
```

You must also configure ldap connection using Spring Boot auto-configuration or LdapTemplate Bean for the above example.

Example for Spring Boot auto-configuration:

```
spring.ldap.password=passwordforldapserver
spring.ldap.urls=urlForLdapServer
spring.ldap.username=usernameForLdapServer
```

62.6. SPRING BOOT AUTO-CONFIGURATION

When using spring-ldap with Spring Boot, use the following Maven dependency to enable support for auto configuration:

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

The component supports 3 options that are listed below.

Name	Description	Default	Type
camel.component.spring-ldap.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.spring-ldap.enabled	Whether to enable auto configuration of the spring-ldap component. This is enabled by default.		Boolean
camel.component.spring-ldap.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 63. SPRING RABBITMQ

Since Camel 3.8

Both producer and consumer are supported

The Spring RabbitMQ component allows you to produce and consume messages from RabbitMQ instances using the Spring RabbitMQ client.

Maven users must add the following dependency to their **pom.xml** for this component:

```
<dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-spring-rabbitmq-starter</artifactId>
</dependency>
```

The version is specified using BOM in the following way.

```
<dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>com.redhat.camel.springboot.platform</groupId>
        <artifactId>camel-spring-boot-bom</artifactId>
        <version>${camel-spring-boot-version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

63.1. URI FORMAT

```
spring-rabbitmq:exchangeName?[options]
```

The **exchangeName** determines the exchange to which the produced messages are sent to. In the case of consumers, the **exchangeName** determines the exchange the queue is bound to.

63.2. CONFIGURING OPTIONS

Camel components are configured on two levels:

- Component level
- Endpoint level

63.2.1. Component Level Options

The **component level** is the highest level. The configurations you define at this level are inherited by all the endpoints. For example, a component can have security settings, credentials for authentication, urls for network connection, and so on.

Since components typically have pre-configured defaults for the most common cases, you may need to only configure a few component options, or maybe none at all.

You can configure components with Component DSL in a configuration file (application.properties|yaml), or directly with Java code.

63.2.2. Endpoint Level Options

At the **Endpoint level** you have many options, which you can use to configure what you want the endpoint to do. The options are categorized according to whether the endpoint is used as a consumer (from) or as a producer (to) or used for both.

You can configure endpoints directly in the endpoint URI as **path** and **query** parameters. You can also use Endpoint DSL and DataFormat DSL as *type safe* ways of configuring endpoints and data formats in Java.

When configuring options, use Property Placeholders for urls, port numbers, sensitive information, and other settings.

Placeholders allows you to externalize the configuration from your code, giving you more flexible and reusable code.

63.3. COMPONENT OPTIONS

The Spring RabbitMQ component supports 29 options that are listed below.

Name	Description	Default	Type
amqpAdmin (common)	**Autowired** Optional AMQP Admin service to use for auto declaring elements (queues, exchanges, bindings).		AmqpAdmin
connectionFactory (common)	**Autowired** The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory

Name	Description	Default	Type
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean
autoDeclare (consumer)	Specifies whether the consumer should auto declare binding between exchange, queue and routing key when starting. Enabling this can be good for development to make it easy to standup exchanges, queues and bindings on the broker.	false	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
deadLetterExchange (consumer)	The name of the dead letter exchange.		String
deadLetterExchangeType (consumer)	The type of the dead letter exchange. Enum values: • direct • fanout • headers • topic	direct	String
deadLetterQueue (consumer)	The name of the dead letter queue.		String
deadLetterRoutingKey (consumer)	The routing key for the dead letter exchange.		String
maximumRetryAttempts (consumer)	How many times a Rabbitmq consumer will retry the same message if Camel failed to process the message.	5	int

Name	Description	Default	Type
rejectAndDontRequeue (consumer)	Whether a Rabbitmq consumer should reject the message without requeuing. This enables failed messages to be sent to a Dead Letter Exchange/Queue, if the broker is so configured.	true	boolean
retryDelay (consumer)	Delay in msec a Rabbitmq consumer will wait before redelivering a message that Camel failed to process.	1000	int
concurrentConsumers (consumer (advanced))	The number of consumers.	1	int
errorHandler (consumer (advanced))	To use a custom ErrorHandler for handling exceptions from the message listener (consumer).		ErrorHandler
listenerContainerFactory (consumer (advanced))	To use a custom factory for creating and configuring ListenerContainer to be used by the consumer for receiving messages.		ListenerContainerFactory
maxConcurrentConsumers (consumer (advanced))	The maximum number of consumers (available only with SMLC).		Integer
messageListenerContainerType (consumer (advanced))	The type of the MessageListenerContainer. Enum values: • DMLC • SMLC	DMLC	String

Name	Description	Default	Type
prefetchCount (consumer (advanced))	Tell the broker how many messages to send to each consumer in a single request. Often this can be set quite high to improve throughput.	250	int
retry (consumer (advanced))	Custom retry configuration to use. If this is configured then the other settings such as maximumRetryAttempts for retry are not in use.		RetryOperationsInterceptor
shutdownTimeout (consumer (advanced))	The time to wait for workers in milliseconds after the container is stopped. If any workers are active when the shutdown signal comes they will be allowed to finish processing as long as they can finish within this timeout.	5000	long
allowNullBody (producer)	Whether to allow sending messages with no body. If this option is false and the message body is null, then an MessageConversionException is thrown.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
replyTimeout (producer)	Specify the timeout in milliseconds to be used when waiting for a reply message when doing request/reply messaging. The default value is 5 seconds. A negative value indicates an indefinite timeout.	5000	long
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
ignoreDeclarationExceptions (advanced)	Switch on ignore exceptions such as mismatched properties when declaring.	false	boolean
messageConverter (advanced)	To use a custom MessageConverter so you can be in control how to map to/from a org.springframework.amqp.core.Message.		MessageConverter
messagePropertiesConverter (advanced)	To use a custom MessagePropertiesConverter so you can be in control how to map to/from a org.springframework.amqp.core.MessageProperties.		MessagePropertiesConverter
headerFilterStrategy (filter)	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message.		HeaderFilterStrategy

63.4. ENDPOINT OPTIONS

The Spring RabbitMQ endpoint is configured using URI syntax:

```
spring-rabbitmq:exchangeName
```

Following are the path and query parameters:

63.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
exchangeName (common)	**Required** The exchange name determines the exchange to which the produced messages will be sent to. In the case of consumers, the exchange name determines the exchange the queue will be bound to. Note: to use default exchange then do not use empty name, but use default instead.		String

63.4.2. Query Parameters (34 parameters)

Name	Description	Default	Type
connectionFactory (common)	The connection factory to be use. A connection factory must be configured either on the component or endpoint.		ConnectionFactory
disableReplyTo (common)	Specifies whether Camel ignores the ReplyTo header in messages. If true, Camel does not send a reply back to the destination specified in the ReplyTo header. You can use this option if you want Camel to consume from a route and you do not want Camel to automatically send back a reply message because another component in your code handles the reply message. You can also use this option if you want to use Camel as a proxy between different message brokers and you want to route message from one system to another.	false	boolean

Name	Description	Default	Type
routingKey (common)	The value of a routing key to use. Default is empty which is not helpful when using the default (or any direct) exchange, but fine if the exchange is a headers exchange for instance.		String
testConnectionOnStartup (common)	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	boolean

Name	Description	Default	Type
acknowledgeMode (consumer)	Flag controlling the behaviour of the container with respect to message acknowledgement. The most common usage is to let the container handle the acknowledgements (so the listener doesn't need to know about the channel or the message). Set to AcknowledgeMode.MANUAL if the listener will send the acknowledgements itself using Channel.basicAck(long, boolean). Manual acks are consistent with either a transactional or non-transactional channel, but if you are doing no other work on the channel at the same other than receiving a single message then the transaction is probably unnecessary. Set to AcknowledgeMode.NONE to tell the broker not to expect any acknowledgements, and it will assume all messages are acknowledged as soon as they are sent (this is autoack in native Rabbit broker terms). If AcknowledgeMode.NONE then the channel cannot be transactional (so the container will fail on start up if that flag is accidentally set). Enum values: • NONE • MANUAL • AUTO		AcknowledgeMode

Name	Description	Default	Type
asyncConsumer (consumer)	Whether the consumer processes the Exchange asynchronously. If enabled then the consumer may pickup the next message from the queue, while the previous message is being processed asynchronously (by the Asynchronous Routing Engine). This means that messages may be processed not 100% strictly in order. If disabled (as default) then the Exchange is fully processed before the consumer will pickup the next message from the queue.	false	boolean
autoDeclare (consumer)	Specifies whether the consumer should auto declare binding between exchange, queue and routing key when starting.	true	boolean
autoStartup (consumer)	Specifies whether the consumer container should auto-startup.	true	boolean
deadLetterExchange (consumer)	The name of the dead letter exchange.		String
deadLetterExchangeType (consumer)	The type of the dead letter exchange. Enum values: • direct • fanout • headers • topic	direct	String

Name	Description	Default	Type
deadLetterQueue (consumer)	The name of the dead letter queue.		String
deadLetterRoutingKey (consumer)	The routing key for the dead letter exchange.		String
exchangeType (consumer)	The type of the exchange. Enum values: • direct • fanout • headers • topic	direct	String
exclusive (consumer)	Set to true for an exclusive consumer.	false	boolean
maximumRetryAttempts (consumer)	How many times a Rabbitmq consumer will retry the same message if Camel failed to process the message.	5	int
noLocal (consumer)	Set to true for an no-local consumer.	false	boolean
queues (consumer)	The queue(s) to use for consuming messages. Multiple queue names can be separated by comma. If none has been configured then Camel will generate an unique id as the queue name for the consumer.		String
rejectAndDontRequeue (consumer)	Whether a Rabbitmq consumer should reject the message without requeuing. This enables failed messages to be sent to a Dead Letter Exchange/Queue, if the broker is so configured.	true	boolean

Name	Description	Default	Type
retryDelay (consumer)	Delay in msec a Rabbitmq consumer will wait before redelivering a message that Camel failed to process.	1000	int
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer (advanced))	The number of consumers.		Integer
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
maxConcurrentConsumers (consumer (advanced))	The maximum number of consumers (available only with SMLC).		Integer
messageListenerContainerType (consumer (advanced))	The type of the MessageListenerContainer. Enum values: • DMLC • SMLC	DMLC	String
prefetchCount (consumer (advanced))	Tell the broker how many messages to send in a single request. Often this can be set quite high to improve throughput.		Integer
retry (consumer (advanced))	Custom retry configuration to use. If this is configured then the other settings such as maximumRetryAttempts for retry are not in use.		RetryOperationsInterceptor
replyTimeout (producer)	Specify the timeout in milliseconds to be used when waiting for a reply message when doing request/reply messaging. The default value is 5 seconds. A negative value indicates an indefinite timeout.	5000	long

Name	Description	Default	Type
usePublisherConnection (producer)	Use a separate connection for publishers and consumers.	false	boolean
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
args (advanced)	Specify arguments for configuring the different RabbitMQ concepts, a different prefix is required for each element: arg.consumer. arg.exchange. arg.queue. arg.binding. arg.dlq.exchange. arg.dlq.queue. arg.dlq.binding. For example to declare a queue with message ttl argument: args=arg.queue.x-message-ttl=60000.		Map

Name	Description	Default	Type
messageConverter (advanced)	To use a custom MessageConverter so you can be in control how to map to/from a org.springframework.amqp.core.Message.		MessageConverter
messagePropertiesConverter (advanced)	To use a custom MessagePropertiesConverter so you can be in control how to map to/from a org.springframework.amqp.core.MessageProperties.		MessagePropertiesConverter
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean

63.5. MESSAGE HEADERS

The Spring RabbitMQ component supports 2 message headers that are listed below:

Name	Description	Default	Type
CamelSpringRabbitmqRoutingOverrideKey (common) Constant: **ROUTING_OVERRIDE_KEY**	The exchange key.		String
CamelSpringRabbitmqExchangeOverrideName (common) Constant: **EXCHANGE_OVERRIDE_NAME**	The exchange name.		String

63.6. USING A CONNECTION FACTORY

To connect to RabbitMQ you must setup a **ConnectionFactory** (same as JMS) with the login details as described below.

It is recommended to use **CachingConnectionFactory** from spring-rabbit as it comes with connection pooling out of the box.

```
<bean id="rabbitConnectionFactory"
class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
  <property name="uri" value="amqp://lolcalhost:5672"/>
</bean>
```

The **ConnectionFactory** is auto-detected by default, so you can just execute it.

```
<camelContext>
  <route>
    <from uri="direct:cheese"/>
    <to uri="spring-rabbitmq:foo?routingKey=cheese"/>
  </route>
</camelContext>
```

63.7. DEFAULT EXCHANGE NAME

To use default exchange name (which would be an empty exchange name in RabbitMQ) you must use **default** as name in the endpoint uri, like:

```
to("spring-rabbitmq:default?routingKey=foo")
```

63.8. AUTO DECLARE EXCHANGES, QUEUES AND BINDINGS

Before you can send or receive messages from RabbitMQ, you must first set up the exchanges, queues and bindings.

In development mode, Camel can automatically do this. You can enable this by setting **autoDeclare=true** on the **SpringRabbitMQComponent**.

Then Spring RabbitMQ automatically declares the elements and sets up the binding between the exchange, queue and routing keys.

The elements can be configured using the multi-valued **args** option.

For example to specify the queue as durable and exclusive, you can configure the endpoint uri with **arg.queue.durable=true&arg.queue.exclusive=true**.

Exchanges

Option	Type	Description	Default
autoDelete	boolean	True if the server should delete the exchange when it is no longer in use (if all bindings are deleted).	false
durable	boolean	A durable exchange will survive a server restart.	true

You can also configure any additional **x-** arguments. See details in the RabbitMQ documentation.

Queues

Option	Type	Description	Default
autoDelete	boolean	True if the server should delete the exchange when it is no longer in use (if all bindings are deleted).	false
durable	boolean	A durable queue will survive a server restart.	false
exclusive	boolean	Whether the queue is exclusive	false
x-dead-letter-exchange	String	The name of the dead letter exchange. If none configured then the component configured value is used.	
x-dead-letter-routing-key	String	The routing key for the dead letter exchange. If none configured then the component configured value is used.	

You can also configure any additional **x-** arguments, such as the message time to live using **x-message-ttl**, and many others. See details in the RabbitMQ documentation.

63.9. MAPPING FROM CAMEL TO RABBITMQ

The message body is mapped from Camel Message body to a **byte[]** which is the type that RabbitMQ uses for message body. Camel uses its type converter to convert the message body to byte array.

Spring Rabbit comes out of the box with support for mapping Java serialized objects but Camel Spring RabbitMQ does **not** support this due to security vulnerabilities and using Java objects is a bad design as it enforces strong coupling.

Custom message headers is mapped from Camel Message headers to RabbitMQ headers. This behaviour can be customized by configuring a new implementation of **HeaderFilterStrategy** on the Camel component.

63.10. REQUEST / REPLY

Request and reply messaging is supported using RabbitMQ direct reply-to .

The example below does request/reply, where the message is sent using the cheese exchange name and routing key *foo.bar*, which is being consumed by the 2nd Camel route, that prepends the message with `Hello `, and then sends back the message.

So if we send **World** as message body to *direct:start* then, we can see the message being logged

- log:request ⇒ World
- log:input ⇒ World
- log:response ⇒ Hello World

```
from("direct:start")
    .to("log:request")
    .to(ExchangePattern.InOut, "spring-rabbitmq:cheese?routingKey=foo.bar")
    .to("log:response");

from("spring-rabbitmq:cheese?queues=myqueue&routingKey=foo.bar")
    .to("log:input")
    .transform(body().prepend("Hello "));
```

63.11. REUSE ENDPOINT AND SEND TO DIFFERENT DESTINATIONS COMPUTED AT RUNTIME

If you need to send messages to a lot of different RabbitMQ exchanges, you must reuse an endpoint and specify the real destination in a message header. This allows Camel to reuse the same endpoint, but send to different exchanges. This greatly reduces the number of endpoints created and economizes on memory and thread resources.

Using toD is easier than specifying the dynamic destination with headers

You can specify using the following headers:

Header	Type	Description
CamelSpringRabbitmqExchangeOverrideName	**String**	The exchange name.
CamelSpringRabbitmqRoutingOverrideKey	**String**	The routing key.

For example, the following route shows how you can compute a destination at run time and use it to override the exchange appearing in the endpoint URL:

```
from("file://inbox")
  .to("bean:computeDestination")
  .to("spring-rabbitmq:dummy");
```

The exchange name, **dummy**, is just a placeholder. It must be provided as part of the RabbitMQ endpoint URL, but it is ignored in this example.

In the **computeDestination** bean, specify the real destination by setting the **CamelRabbitmqExchangeOverrideName** header as follows:

```
public void setExchangeHeader(Exchange exchange) {
   String region = ....
   exchange.getIn().setHeader("CamelSpringRabbitmqExchangeOverrideName", "order-" + region);
}
```

Camel reads this header and uses it as the exchange name instead of the one configured on the endpoint. So, in this example Camel sends the message to **spring-rabbitmq:order-emea**, assuming the **region** value was **emea**.

The producer removes both **CamelSpringRabbitmqExchangeOverrideName** and **CamelSpringRabbitmqRoutingOverrideKey** headers from the exchange and do not propagate them to the created Rabbitmq message in order to avoid the accidental loops in the routes (in scenarios when the message is forwarded to another RabbitMQ endpoint).

63.12. USING TOD

If you need to send messages to a lot of different exchanges, you must reuse an endpoint and specify the dynamic destinations with simple language using toD.

For example, you need to send messages to the exchange with order types, then you can use **toD** as follows:

```
from("direct:order")
  .toD("spring-rabbit:order-${header.orderType}");
```

63.13. SPRING BOOT AUTO-CONFIGURATION

When using spring-rabbitmq with Spring Boot use the following Maven dependency to enable support for auto configuration:

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

The component supports 30 options that are listed below.

Name	Description	Default	Type
camel.component.spring-rabbitmq.allow-null-body	Whether to allow sending messages with no body. If this option is false and the message body is null, then an MessageConversionException is thrown.	false	Boolean

Name	Description	Default	Type
camel.component.spring-rabbitmq.amqp-admin	Optional AMQP Admin service to use for auto declaring elements (queues, exchanges, bindings). The option is a org.springframework.amqp.core.AmqpAdmin type.		AmqpAdmin
camel.component.spring-rabbitmq.auto-declare	Specifies whether the consumer should auto declare binding between exchange, queue and routing key when starting. Enabling this can be good for development to make it easy to standup exchanges, queues and bindings on the broker.	false	Boolean
camel.component.spring-rabbitmq.auto-startup	Specifies whether the consumer container should auto-startup.	true	Boolean
camel.component.spring-rabbitmq.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.spring-rabbitmq.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.spring-rabbitmq.concurrent-consumers	The number of consumers.	1	Integer
camel.component.spring-rabbitmq.connection-factory	The connection factory to be use. A connection factory must be configured either on the component or endpoint. The option is a org.springframework.amqp.rabbit.connection.ConnectionFactory type.		ConnectionFactory
camel.component.spring-rabbitmq.dead-letter-exchange	The name of the dead letter exchange.		String
camel.component.spring-rabbitmq.dead-letter-exchange-type	The type of the dead letter exchange.	direct	String
camel.component.spring-rabbitmq.dead-letter-queue	The name of the dead letter queue.		String

Name	Description	Default	Type
camel.component.spring-rabbitmq.dead-letter-routing-key	The routing key for the dead letter exchange.		String
camel.component.spring-rabbitmq.enabled	Whether to enable auto configuration of the spring-rabbitmq component. This is enabled by default.		Boolean
camel.component.spring-rabbitmq.error-handler	To use a custom ErrorHandler for handling exceptions from the message listener (consumer). The option is a org.springframework.util.ErrorHandler type.		ErrorHandler
camel.component.spring-rabbitmq.header-filter-strategy	To use a custom org.apache.camel.spi.HeaderFilterStrategy to filter header to and from Camel message. The option is a org.apache.camel.spi.HeaderFilterStrategy type.		HeaderFilterStrategy
camel.component.spring-rabbitmq.ignore-declaration-exceptions	Switch on ignore exceptions such as mismatched properties when declaring.	false	Boolean

Name	Description	Default	Type
camel.component.spring-rabbitmq.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.spring-rabbitmq.listener-container-factory	To use a custom factory for creating and configuring ListenerContainer to be used by the consumer for receiving messages. The option is a org.apache.camel.component.springrabbit.ListenerContainerFactory type.		ListenerContainerFactory
camel.component.spring-rabbitmq.max-concurrent-consumers	The maximum number of consumers (available only with SMLC).		Integer
camel.component.spring-rabbitmq.maximum-retry-attempts	How many times a Rabbitmq consumer will retry the same message if Camel failed to process the message.	5	Integer

Name	Description	Default	Type
camel.component.spring-rabbitmq.message-converter	To use a custom MessageConverter so you can be in control how to map to/from a org.springframework.amqp.core.Message. The option is a org.springframework.amqp.support.converter.MessageConverter type.		MessageConverter
camel.component.spring-rabbitmq.message-listener-container-type	The type of the MessageListenerContainer.	DMLC	String
camel.component.spring-rabbitmq.message-properties-converter	To use a custom MessagePropertiesConverter so you can be in control how to map to/from a org.springframework.amqp.core.MessageProperties. The option is a org.apache.camel.component.springrabbit.MessagePropertiesConverter type.		MessagePropertiesConverter
camel.component.spring-rabbitmq.prefetch-count	Tell the broker how many messages to send to each consumer in a single request. Often this can be set quite high to improve throughput.	250	Integer
camel.component.spring-rabbitmq.reject-and-dont-requeue	Whether a Rabbitmq consumer should reject the message without requeuing. This enables failed messages to be sent to a Dead Letter Exchange/Queue, if the broker is so configured.	true	Boolean

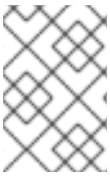
Name	Description	Default	Type
camel.component.spring-rabbitmq.reply-timeout	Specify the timeout in milliseconds to be used when waiting for a reply message when doing request/reply messaging. The default value is 5 seconds. A negative value indicates an indefinite timeout. The option is a long type.	5000	Long
camel.component.spring-rabbitmq.retry	Custom retry configuration to use. If this is configured then the other settings such as maximumRetryAttempts for retry are not in use. The option is a org.springframework.retry.interceptor.RetryOperationsInterceptor type.		RetryOperationsInterceptor
camel.component.spring-rabbitmq.retry-delay	Delay in msec a Rabbitmq consumer will wait before redelivering a message that Camel failed to process.	1000	Integer
camel.component.spring-rabbitmq.shutdown-timeout	The time to wait for workers in milliseconds after the container is stopped. If any workers are active when the shutdown signal comes they will be allowed to finish processing as long as they can finish within this timeout. The option is a long type.	5000	Long

Name	Description	Default	Type
camel.component.spring-rabbitmq.test-connection-on-startup	Specifies whether to test the connection on startup. This ensures that when Camel starts that all the JMS consumers have a valid connection to the JMS broker. If a connection cannot be granted then Camel throws an exception on startup. This ensures that Camel is not started with failed connections. The JMS producers is tested as well.	false	Boolean

CHAPTER 64. SPRING REDIS

The producer and consumer are supported.

This component allows sending and receiving messages from Redis. Redis is an advanced key-value store where keys can contain strings, hashes, lists, sets and sorted sets. In addition Redis provides pub/sub functionality for inter-app communications. Camel provides a producer for executing commands, a consumer for subscribing to pub/sub messages, and an idempotent repository for filtering out duplicate messages.



NOTE

Prerequisites

To use this component, you must have a Redis server running.

64.1. URI FORMAT

```
spring-redis://host:port[?options]
```

64.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

64.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example, a component may have security settings, credentials for authentication, URLs for network connection.

Some components only have a few options, and others may have many. Because components typically have pre-configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

64.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as a consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL and DataFormat DSL as a *type safe* way of configuring endpoints and data formats in Java.

A good practice when configuring options is to use Property Placeholders, which allows you to not hardcode URLs, port numbers, sensitive information, and other settings. In other words, placeholders allow you to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

64.3. COMPONENT OPTIONS

The Spring Redis component supports 4 options, which are listed below.

Name	Description	Default	Type
redisTemplate (common)	**Autowired** Reference to a pre-configured RedisTemplate instance to use.		RedisTemplate
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer uses the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy, you can use this to allow CamelContext and routes to start up in situations where a producer may otherwise fail during starting and cause the route to fail being started. By starting lazy, Camel's routing error handlers handle any startup failures while routing messages. Beware that when the first message is processed, creating, and starting, the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring of JDBC data sources, JMS connection factories, and AWS Clients.	true	boolean

64.4. ENDPOINT OPTIONS

The Spring Redis endpoint is configured using URI syntax:

```
spring-redis:host:port
```

with the following path and query parameters:

64.4.1. Path Parameters (2 parameters)

Name	Description	Default	Type
host (common)	**Required** The host where the Redis server is running.		String
port (common)	**Required** Redis server port number.		Integer

64.4.2. Query Parameters (10 parameters)

Name	Description	Default	Type
channels (common)	List of topic names or name patterns to subscribe to. Multiple names can be separated by a comma.		String
command (common)	Default command, which can be overridden by message header. Notice that the consumer only supports the following commands only: PSUBSCRIBE and SUBSCRIBE. Enum values: • PING • SET • GET • QUIT • EXISTS • DEL • TYPE • FLUSHDB • KEYS • RANDOMKEY • RENAME • RENAMENX • RENAMEX • DBSIZE • EXPIRE • EXPIREAT • TTL • SELECT	SET	Command

Name	Description	Default	Type
	• MOVE • FLUSHALL • GETSET • MGET • SETNX • SETEX • MSET • MSETNX • DECRBY • DECR • INCRBY • INCR • APPEND • SUBSTR • HSET • HGET • HSETNX • HMSET • HMGET • HINCRBY • HEXISTS • HDEL • HLEN • HKEYS • HVALS • HGETALL • RPUSH • LPUSH • LLEN • LRANGE • LTRIM • LINDEX • LSET		

Name	Description	Default	Type
	• LREM • LPOP • RPOP • RPOPLPUSH • SADD • SMEMBERS • SREM • SPOP • SMOVE • SCARD • SISMEMBER • SINTER • SINTERSTORE • SUNION • SUNIONSTORE • SDIFF • SDIFFSTORE • SRANDMEMBER • ZADD • ZRANGE • ZREM • ZINCRBY • ZRANK • ZREVRANK • ZREVRANGE • ZCARD • ZSCORE • MULTI • DISCARD • EXEC • WATCH • UNWATCH • SORT		

Name	Description	Default	Type
	• BLPOP • BRPOP • AUTH • SUBSCRIBE • PUBLISH • UNSUBSCRIBE • PSUBSCRIBE • PUNSUBSCRIBE • ZCOUNT • ZRANGEBYSCORE • ZREVRANGEBYSCORE • ZREMRANGEBYRANK • ZREMRANGEBYSCORE • ZUNIONSTORE • ZINTERSTORE • SAVE • BGSAVE • BGREWRITEAOF • LASTSAVE • SHUTDOWN • INFO • MONITOR • SLAVEOF • CONFIG • STRLEN • SYNC • LPUSHX • PERSIST • RPUSHX • ECHO • LINSERT • DEBUG • BRPOPLPUSH		

Name	Description	Default	Type
	• SETBIT • GETBIT • SETRANGE • GETRANGE • PEXPIRE • PEXPIREAT • GEOADD • GEODIST • GEOHASH • GEOPOS • GEORADIUS • GEORADIUSBYMEMBER		
connectionFactory (common)	Reference to a pre-configured RedisConnectionFactory instance to use.		RedisConnectionFactory
redisTemplate (common)	Reference to a pre-configured RedisTemplate instance to use.		RedisTemplate
serializer (common)	Reference to a pre-configured RedisSerializer instance to use.		RedisSerializer
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which means any exceptions that occurred while the consumer is trying to pick up incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. The consumer defaults to use the org.apache.camel.spi.ExceptionHandler to deal with exceptions. These exceptions log at WARN or ERROR level and ignored.	False	Boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. If you enable the bridgeErrorHandler option, this option is not used. By default, the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler

Name	Description	Default	Type
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
listenerContainer (consumer (advanced))	Reference to a pre-configured RedisMessageListenerContainer instance to use.		RedisMessageListenerContainer
lazyStartProducer (Producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy, you can use this to allow CamelContext and routes to start up in situations where a producer may otherwise fail during starting and cause the route startup to fail. By deferring this startup to be lazy, the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	False	Boolean

64.5. MESSAGE HEADERS

The Spring Redis component supports 29 message header(s), which is/are listed below:

Name	Description	Default	Type
CamelRedis.Command (producer) Constant: COMMAND	The command to perform.		String
CamelRedis.Key (common) Constant: KEY	The key.		String

Name	Description	Default	Type
CamelRedis.Keys (common) Constant: KEYS	The keys.		Collection
CamelRedis.Field (common) Constant: FIELD	The field.		String
CamelRedis.Fields (common) Constant: FIELDS	The fields.		Collection
CamelRedis.Value (common) Constant: VALUE	The value.		Object
CamelRedis.Values (common) Constant: VALUES	The values.		Map
CamelRedis.Start (common) Constant: START	Start.		Long
CamelRedis.End (common) Constant: END	End.		Long
CamelRedis.Timeout (common) Constant: TIMEOUT	The timeout.		Long
CamelRedis.Offset (common) Constant: OFFSET	The offset.		Long

Name	Description	Default	Type
CamelRedis.Destination (common) Constant: DESTINATION	The destination.		String
CamelRedis.Channel (common) Constant: CHANNEL	The channel.		byte[] or String
CamelRedis.Message (common) Constant: MESSAGE	The message.		Object
CamelRedis.Index (common) Constant: INDEX	The index.		Long
CamelRedis.Position (common) Constant: POSITION	The position.		String
CamelRedis.Pivot (common) Constant: PIVOT	The pivot.		String
CamelRedis.Count (common) Constant: COUNT	Count.		Long
CamelRedis.Timestamp (common) Constant: TIMESTAMP	The timestamp.		Long
CamelRedis.Pattern (common) Constant: PATTERN	The pattern.		byte[] or String

Name	Description	Default	Type
CamelRedis.Db (common) Constant: DB	The db.		Integer
CamelRedis.Score (common) Constant: SCORE	The score.		Double
CamelRedis.Min (common) Constant: MIN	The min.		Double
CamelRedis.Max (common) Constant: MAX	The max.		Double
CamelRedis.Increment (common) Constant: INCREMENT	Increment.		Double
CamelRedis.WithScore (common) Constant: WITHSCORE	WithScore.		Boolean
CamelRedis.Latitude (common) Constant: LATITUDE	Latitude.		Double
CamelRedis.Longitude (common) Constant: LONGITUDE	Latitude.		Double
CamelRedis.Radius (common) Constant: RADIUS	Radius.		Double

64.6. USAGE

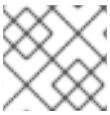
Also, see the available unit tests.

Redis Producer

```
from(“direct:start”)
    .setHeader("CamelRedis.Key", constant(key))
    .setHeader("CamelRedis.Value", constant(value))
    .to("spring-redis://host:port?command=SET&redisTemplate=#redisTemplate");
```

Redis Consumer

```
from("spring-redis://host:port?command=SUBSCRIBE&channels=myChannel")
   .log("Received message: ${body}");
```



NOTE

Where '//host:port' is URL address for running Redis server.

64.6.1. Message headers evaluated by the Redis producer

The producer issues commands to the server and each command has a different set of parameters with specific types. The result from the command execution is returned in the message body.

Hash Commands	Description	Parameters	Result
HSET	Set the string value of a hash field	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Void
HGET	Get the value of a hash field	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String)	String
HSETNX	Set the value of a hash field, only if the field does not exist	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Void
HMSET	Set multiple hash fields to multiple values	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUES**/"CamelRedis.Values" (Map<String, Object>)	Void

Hash Commands	Description	Parameters	Result
HMGET	Get the values of all the given hash fields	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELDS**/"CamelRedis.Filds" (Collection<String>)	Collection<Object>
HINCRBY	Increment the integer value of a hash field by the given number	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Long)	Long
HEXISTS	Determine if a hash field exists	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String)	Boolean
HDEL	Delete one or more hash fields	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.FIELD**/"CamelRedis.Field" (String)	Void
HLEN	Get the number of fields in a hash	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
HKEYS	Get all the fields in a hash	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Set<String>
HVALS	Get all the values in a hash	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Collection<Object>
HGETALL	Get all the fields and values in a hash	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Map<String, Object>

List Commands	Description	Parameters	Result
RPUSH	Append one or multiple values to a list	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long

List Commands	Description	Parameters	Result
RPUSHX	Append a value to a list, only if the list exists	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long
LPUSH	Prepend one or multiple values to a list	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long
LLEN	Get the length of a list	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
LRANGE	Get a range of elements from a list	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"Long), **RedisConstants.END**/"CamelRedis.End" (Long)	List<Object>
LTRIM	Trim a list to the specified range	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"Long), **RedisConstants.END**/"CamelRedis.End" (Long)	Void
LINDEX	Get an element from a list by its index	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.INDEX**/"CamelRedis.Index" (Long)	String
LINSERT	Insert an element before or after another element in a list	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.PIVOT**/"CamelRedis.Pivot" (String), **RedisConstants.POSITION**/"CamelRedis.Position" (String)	Long
LSET	Set the value of an element in a list by its index	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.INDEX**/"CamelRedis.Index" (Long)	Void

List Commands	Description	Parameters	Result
LREM	Remove elements from a list	**RedisConstants.KEY**/**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.COUNT**/"CamelRedis.Count" (Long)	Long
LPOP	Remove and get the first element in a list	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Object
RPOP	Remove and get the last element in a list	**RedisConstants.KEY**/"CamelRedis.Key" (String)	String
RPOPLPUSH	Remove the last element in a list, append it to another list and return it	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Object
BRPOPLPUSH	Pop a value from a list, push it to another list and return it; or block until one is available	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long)	Object
BLPOP	Remove and get the first element in a list, or block until one is available	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long)	Object
BRPOP	Remove and get the last element in a list, or block until one is available	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long)	String

Set Commands	Description	Parameters	Result
SADD	Add one or more members to a set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Boolean
SMEMBERS	Get all the members in a set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Set<Object>

Set Commands	Description	Parameters	Result
SREM	Remove one or more members from a set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Boolean
SPOP	Remove and return a random member from a set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	String
SMOVE	Move a member from one set to another	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Boolean
SCARD	Get the number of members in a set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
SISMEMBER	Determine if a given value is a member of a set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Boolean
SINTER	Intersect multiple sets	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String)	Set<Object>
SINTERSTORE	Intersect multiple sets and store the resulting set in a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Void
SUNION	Add multiple sets	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String)	Set<Object>
SUNIONSTORE	Add multiple sets and store the resulting set in a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Void

Set Commands	Description	Parameters	Result
SDIFF	Subtract multiple sets	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String)	Set<Object>
SDIFFSTORE	Subtract multiple sets and store the resulting set in a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Void
SRANDMEMBER	Get one or multiple random members from a set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	String

Ordered set Commands	Description	Parameters	Result
ZADD	Add one or more members to a sorted set, or update its score if it already exists	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.SCORE**/"CamelRedis.Score" (Double)	Boolean
ZRANGE	Return a range of members in a sorted set, by index	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"Long), **RedisConstants.END**/"CamelRedis.End" (Long), **RedisConstants.WITHSCORE**/"CamelRedis.WithScore" (Boolean)	Object
ZREM	Remove one or more members from a sorted set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Boolean
ZINCRBY	Increment the score of a member in a sorted set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.INCREMENT**/"CamelRedis.Increment" (Double)	Double

Ordered set Commands	Description	Parameters	Result
ZRANK	Determine the index of a member in a sorted set	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long
ZREVRANK	Determine the index of a member in a sorted set, with scores ordered from high to low	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long
ZREVRANGE	Return a range of members in a sorted set, by index, with scores ordered from high to low	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"Long), **RedisConstants.END**/"CamelRedis.End" (Long), **RedisConstants.WITHSCORE**/"CamelRedis.WithScore" (Boolean)	Object
ZCARD	Get the number of members in a sorted set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
ZCOUNT	Count the members in a sorted set with scores within the given values	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.MIN**/"CamelRedis.Min" (Double), **RedisConstants.MAX**/"CamelRedis.Max" (Double)	Long
ZRANGEBYSCORE	Return a range of members in a sorted set, by score	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.MIN**/"CamelRedis.Min" (Double), **RedisConstants.MAX**/"CamelRedis.Max" (Double)	Set<Object>
ZREVRANGEBYSCORE	Return a range of members in a sorted set, by score, with scores ordered from high to low	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.MIN**/"CamelRedis.Min" (Double), **RedisConstants.MAX**/"CamelRedis.Max" (Double)	Set<Object>

Ordered set Commands	Description	Parameters	Result
ZREMRANGEBYRANK	Remove all members in a sorted set within the given indexes	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"(Long), **RedisConstants.END**/"CamelRedis.End" (Long)	Void
ZREMRANGEBYSCORE	Remove all members in a sorted set within the given scores	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"(Long), **RedisConstants.END**/"CamelRedis.End" (Long)	Void
ZUNIONSTORE	Add multiple sorted sets and store the resulting sorted set in a new key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Void
ZINTERSTORE	Intersect multiple sorted sets and store the resulting sorted set in a new key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.KEYS**/"CamelRedis.Keys" (String), **RedisConstants.DESTINATION**/"CamelRedis.Destination" (String)	Void

String Commands	Description	Parameters	Result
SET	Set the string value of a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Void
GET	Get the value of a key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Object
STRLEN	Get the length of the value stored in a key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
APPEND	Append a value to a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (String)	Integer

String Commands	Description	Parameters	Result
SETBIT	Sets or clears the bit at offset in the string value stored at key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.OFFSET**/"CamelRedis.Offset" (Long), **RedisConstants.VALUE**/"CamelRedis.Value" (Boolean)	Void
GETBIT	Returns the bit value at offset in the string value stored at key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.OFFSET**/"CamelRedis.Offset" (Long)	Boolean
SETRANGE	Overwrite part of a string at key starting at the specified offset	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.OFFSET**/"CamelRedis.Offset" (Long)	Void
GETRANGE	Get a substring of the string stored at a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.START**/"CamelRedis.Start"(Long), **RedisConstants.END**/"CamelRedis.End" (Long)	String
SETNX	Set the value of a key, only if the key does not exist	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Boolean
SETEX	Set the value and expiration of a key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long), SECONDS	Void
DECRBY	Decrement the integer value of a key by the given number	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Long)	Long
DECR	Decrement the integer value of a key by one	**RedisConstants.KEY**/"CamelRedis.Key" (String),	Long
INCRBY	Increment the integer value of a key by the given amount	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Long)	Long

String Commands	Description	Parameters	Result
INCR	Increment the integer value of a key by one	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
MGET	Get the values of all the given keys	**RedisConstants.FIELDS**/"CamelRedis.Filds" (Collection<String>)	List<Object>
MSET	Set multiple keys to multiple values	**RedisConstants.VALUES**/"CamelRedis.Values" (Map<String, Object>)	Void
MSETNX	Set multiple keys to multiple values, only if none of the keys exist	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Void
GETSET	Set the string value of a key and return its old value	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Object

Key Commands	Description	Parameters	Result
EXISTS	Determine if a key exists	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Boolean
DEL	Delete a key	**RedisConstants.KEYS**/"CamelRedis.Keys" (String)	Void
TYPE	Determine the type stored at key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	DataType
KEYS	Find all keys matching the given pattern	**RedisConstants.PATERN**/"CamelRedis.Pattern" (String)	Collection<String>
RANDOMKEY	Return a random key from the keyspace	**RedisConstants.PATERN**/"CamelRedis.Pattern" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (String)	String
RENAME	Rename a key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Void
RENAMENX	Rename a key, only if the new key does not exist	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (String)	Boolean

Key Commands	Description	Parameters	Result
EXPIRE	Set a key's time to live in seconds	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long)	Boolean
SORT	Sort the elements in a list, set or sorted set	**RedisConstants.KEY**/"CamelRedis.Key" (String)	List<Object>
PERSIST	Remove the expiration from a key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Boolean
EXPIREAT	Set the expiration for a key as a UNIX timestamp	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMESTAMP**/"CamelRedis.Timestamp" (Long)	Boolean
PEXPIRE	Set a key's time to live in milliseconds	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMEOUT**/"CamelRedis.Timeout" (Long)	Boolean
PEXPIREAT	Set the expiration for a key as a UNIX timestamp specified in milliseconds	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.TIMESTAMP**/"CamelRedis.Timestamp" (Long)	Boolean
TTL	Get the time to live for a key	**RedisConstants.KEY**/"CamelRedis.Key" (String)	Long
MOVE	Move a key to another database	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.DB**/"CamelRedis.Db" (Integer)	Boolean

Geo Commands	Description	Parameters	Result
GEOADD	Adds the specified geospatial items (latitude, longitude, name) to the specified key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.LATITUDE**/"CamelRedis.Latitude" (Double), **RedisConstants.LONGITUDE**/"CamelRedis.Longitude" (Double), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	Long

Geo Commands	Description	Parameters	Result
GEODIST	Return the distance between two members in the geospatial index for the specified key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUES**/"CamelRedis.Values" (Object[])	Distance
GEOHASH	Return valid Geohash strings representing the position of an element in the geospatial index for the specified key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	List<String>
GEOPOS	Return the positions (longitude, latitude) of an element in the geospatial index for the specified key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object)	List<Point>
GEORADIUS	Return the element in the geospatial index for the specified key, which is within the borders of the area specified with the central location and the maximum distance from the center (the radius)	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.LATITUDE**/"CamelRedis.Latitude" (Double), **RedisConstants.LONGITUDE**/"CamelRedis.Longitude" (Double), **RedisConstants.RADIUS**/"CamelRedis.Radius" (Double), **RedisConstants.COUNT**/"CamelRedis.Count" (Integer)	GeoResults
GEORADIUSBYMEMBER	This command is exactly like GEORADIUS with the sole difference that instead of taking, as the center of the area to query, a longitude and latitude value, it takes the name of a member already existing inside the geospatial index for the specified key	**RedisConstants.KEY**/"CamelRedis.Key" (String), **RedisConstants.VALUE**/"CamelRedis.Value" (Object), **RedisConstants.RADIUS**/"CamelRedis.Radius" (Double), **RedisConstants.COUNT**/"CamelRedis.Count" (Integer)	GeoResults

Other Commands	Description	Parameters	Result
MULTI	Mark the start of a transaction block	none	Void
DISCARD	Discard all commands issued after MULTI	none	Void
EXEC	Execute all commands issued after MULTI	none	Void

Other Commands	Description	Parameters	Result
WATCH	Watch the given keys to determine the execution of the MULTI/EXEC block	**RedisConstants.KEYS**/"CamelRedis.Keys" (String)	Void
UNWATCH	Forget about all watched keys	none	Void
ECHO	Echo the given string	**RedisConstants.VALUE**/"CamelRedis.Value" (String)	String
PING	Ping the server	none	String
QUIT	Close the connection	none	Void
PUBLISH	Post a message to a channel	**RedisConstants.CHANNEL**/"CamelRedis.Channel" (String), **RedisConstants.MESSAGE**/"CamelRedis.Message" (Object)	Void

64.7. DEPENDENCIES

Maven users will need to add the following dependency to their pom.xml.

pom.xml

```
<dependency>
      <groupId>org.apache.camel.springboot</groupId>
      <artifactId>camel-spring-redis-starter</artifactId>
</dependency>
```

Use the BOM to get the version.

```
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>com.redhat.camel.springboot.platform</groupId>
         <artifactId>camel-spring-boot-bom</artifactId>
         <version>${camel-spring-boot-version}</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

64.8. SPRING BOOT AUTO-CONFIGURATION

When using spring-redis with Spring Boot, ensure you use the following Maven dependency to have support for auto-configuration:

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

The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.component.spring-redis.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatically configure JDBC data sources, JMS connection factories, AWS Clients, etc.	True	Boolean
camel.component.spring-redis.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which means any exceptions that occur while the consumer is trying to pick up incoming messages, or the likes, will be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	False	Boolean
camel.component.spring-redis.enabled	Whether to enable auto configuration of the spring-redis component. This is enabled by default.		Boolean
camel.component.spring-redis.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy, you can allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail start up. By deferring this startup to be lazy, the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	False	Boolean
camel.component.spring-redis.redis-template	Reference to a pre-configured RedisTemplate instance to use. The option is an org.springframework.data.redis.core.RedisTemplate type.		RedisTemplate

CHAPTER 65. SPRING WEBSERVICE

Since Camel 2.6

Both producer and consumer are supported

The Spring WS component allows you to integrate with Spring Web Services. It offers both, *client*-side support for accessing web services, and *server*-side support for creating your own contract-first web services.

Maven users must add the following dependency to their **pom.xml** for this component:

```
<dependency>
      <groupId>org.apache.camel.springboot</groupId>
      <artifactId>camel-spring-ws-starter</artifactId>
</dependency>
```

Use the BOM to get the version.

```
<dependencyManagement>
       <dependencies>
          <dependency>
             <groupId>com.redhat.camel.springboot.platform</groupId>
             <artifactId>camel-spring-boot-bom</artifactId>
             <version>${camel-spring-boot-version}</version>
             <type>pom</type>
             <scope>import</scope>
          </dependency>
       </dependencies>
    </dependencyManagement>
```

65.1. URI FORMAT

The URI scheme for this component is as follows

```
spring-ws:[mapping-type:]address[?options]
```

To expose a web service **mapping-type** needs to be set to any of the following:

Mapping type	Description
rootqname	Offers the option to map web service requests based on the qualified name of the root element contained in the message.
soapaction	Used to map web service requests based on the SOAP action specified in the header of the message.
uri	In order to map web service requests that target a specific URI.

Mapping type	Description
xpathresult	Used to map web service requests based on the evaluation of an XPath **expression** against the incoming message. The result of the evaluation should match the XPath result specified in the endpoint URI.
beanname	Allows you to reference an **org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher** object in order to integrate with existing (legacy) endpoint mappings like **PayloadRootQNameEndpointMapping**, **SoapActionEndpointMapping**, etc

As a consumer the **address** should contain a value relevant to the specified mapping-type (e.g. a SOAP action, XPath expression). As a producer the address should be set to the URI of the web service your calling upon.

65.2. CONFIGURING OPTIONS

Camel components are configured on two levels:

- Component level
- Endpoint level

65.2.1. Component Level Options

The **component level** is the highest level. The configurations you define at this level are inherited by all the endpoints. For example, a component can have security settings, credentials for authentication, urls for network connection, and so on.

Since components typically have pre-configured defaults for the most common cases, you may need to only configure a few component options, or maybe none at all.

You can configure components with Component DSL in a configuration file (application.properties|yaml), or directly with Java code.

65.2.2. Endpoint Level Options

At the **Endpoint level** you have many options, which you can use to configure what you want the endpoint to do. The options are categorized according to whether the endpoint is used as a consumer (from) or as a producer (to) or used for both.

You can configure endpoints directly in the endpoint URI as **path** and **query** parameters. You can also use Endpoint DSL and DataFormat DSL as *type safe* ways of configuring endpoints and data formats in Java.

When configuring options, use Property Placeholders for urls, port numbers, sensitive information, and other settings.

Placeholders allows you to externalize the configuration from your code, giving you more flexible and reusable code.

65.3. COMPONENT OPTIONS

The Spring WebService component supports 4 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
useGlobalSslContextParameters (security)	Enable usage of global SSL context parameters.	false	boolean

65.4. ENDPOINT OPTIONS

The Spring WebService endpoint is configured using URI syntax:

```
spring-ws:type:lookupKey:webServiceEndpointUri
```

Following are the path and query parameters:

65.4.1. Path Parameters (4 parameters)

Name	Description	Default	Type
type (consumer)	Endpoint mapping type if endpoint mapping is used. rootqname - Offers the option to map web service requests based on the qualified name of the root element contained in the message. soapaction - Used to map web service requests based on the SOAP action specified in the header of the message. uri - In order to map web service requests that target a specific URI. xpathresult - Used to map web service requests based on the evaluation of an XPath expression against the incoming message. The result of the evaluation should match the XPath result specified in the endpoint URI. beanname - Allows you to reference an org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher object in order to integrate with existing (legacy) endpoint mappings like PayloadRootQNameEndpointMapping, SoapActionEndpointMapping, etc. Enum values: • ROOT_QNAME • ACTION		EndpointMappingType

Name	Description	Default	Type
	• TO • SOAP_ACTION • XPATHRESULT • URI • URI_PATH • BEANNAME		
lookupKey (consumer)	Endpoint mapping key if endpoint mapping is used.		String
webServiceEndpointUri (producer)	The default Web Service endpoint uri to use for the producer.		String
expression (consumer)	The XPath expression to use when option type=xpathresult. Then this option is required to be configured.		String

65.4.2. Query Parameters (21 parameters)

Name	Description	Default	Type
messageFilter (common)	Option to provide a custom MessageFilter. For example when you want to process your headers or attachments by your own.		MessageFilter
messageIdStrategy (common)	Option to provide a custom MessageIdStrategy to control generation of WS-Addressing unique message ids.		MessageIdStrategy

Name	Description	Default	Type
endpointDispatcher (consumer)	Spring org.springframework.ws.server.endpoint.MessageEndpoint for dispatching messages received by Spring-WS to a Camel endpoint, to integrate with existing (legacy) endpoint mappings like PayloadRootQNameEndpointMapping, SoapActionEndpointMapping, etc.		CamelEndpointDispatcher
endpointMapping (consumer)	Reference to an instance of org.apache.camel.component.spring.ws.bean.CamelEndpointMapping in the Registry/ApplicationContext. Only one bean is required in the registry to serve all Camel/Spring-WS endpoints. This bean is auto-discovered by the MessageDispatcher and used to map requests to Camel endpoints based on characteristics specified on the endpoint (like root QName, SOAP action, etc).		CamelSpringWSEndpointMapping

Name	Description	Default	Type
bridgeErrorHandler (consumer (advanced))	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
allowResponseAttachmentOverride (producer)	Option to override soap response attachments in in/out exchange with attachments from the actual service layer. If the invoked service appends or rewrites the soap attachments this option when set to true, allows the modified soap attachments to be overwritten in in/out message attachments.	false	boolean
allowResponseHeaderOverride (producer)	Option to override soap response header in in/out exchange with header info from the actual service layer. If the invoked service appends or rewrites the soap header this option when set to true, allows the modified soap header to be overwritten in in/out message headers.	false	boolean
faultAction (producer)	Signifies the value for the faultAction response WS-Addressing Fault Action header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI
faultTo (producer)	Signifies the value for the faultAction response WS-Addressing FaultTo header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI

Name	Description	Default	Type
messageFactory (producer)	Option to provide a custom WebServiceMessageFactory. For example when you want Apache Axiom to handle web service messages instead of SAAJ.		WebServiceMessageFactory
messageSender (producer)	Option to provide a custom WebServiceMessageSender. For example to perform authentication or use alternative transports.		WebServiceMessageSender
outputAction (producer)	Signifies the value for the response WS-Addressing Action header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI
replyTo (producer)	Signifies the value for the replyTo response WS-Addressing ReplyTo header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI
soapAction (producer)	SOAP action to include inside a SOAP request when accessing remote web services.		String

Name	Description	Default	Type
timeout (producer)	Sets the socket read timeout (in milliseconds) while invoking a webservice using the producer, see URLConnection.setReadTimeout() and CommonsHttpMessageSender.setReadTimeout(). This option works when using the built-in message sender implementations: CommonsHttpMessageSender and HttpUrlConnectionMessageSender. One of these implementations will be used by default for HTTP based services unless you customize the Spring WS configuration options supplied to the component. If you are using a non-standard sender, it is assumed that you will handle your own timeout configuration. The built-in message sender HttpComponentsMessageSender is considered instead of CommonsHttpMessageSender which has been deprecated, see HttpComponentsMessageSender.setReadTimeout().		int
webServiceTemplate (producer)	Option to provide a custom WebServiceTemplate. This allows for full control over client-side web services handling; like adding a custom interceptor or specifying a fault resolver, message sender or message factory.		WebServiceTemplate

Name	Description	Default	Type
wsAddressingAction (producer)	WS-Addressing 1.0 action header to include when accessing web services. The To header is set to the address of the web service as specified in the endpoint URI (default Spring-WS behavior).		URI
lazyStartProducer (producer (advanced))	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
sslContextParameters (security)	To configure security using SSLContextParameters.		SSLContextParameters

65.5. MESSAGE HEADERS

The Spring WebService component supports 7 message headers that are listed below:

Name	Description	Default	Type
CamelSpringWebserviceEndpointUri (producer) Constant: **SPRING_WS_ENDPOINT_URI**	The endpoint URI.		String
CamelSpringWebserviceSoapAction (producer) Constant: **SPRING_WS_SOAP_ACTION**	SOAP action to include inside a SOAP request when accessing remote web services.		String
CamelSpringWebserviceSoapHeader (producer) Constant: **SPRING_WS_SOAP_HEADER**	The soap header source.		Source
CamelSpringWebserviceAddressingAction (producer) Constant: **SPRING_WS_ADDRESSING_ACTION**	WS-Addressing 1.0 action header to include when accessing web services. The To header is set to the address of the web service as specified in the endpoint URI (default Spring-WS behavior).		URI
CamelSpringWebserviceAddressingFaultTo (producer) Constant: **SPRING_WS_ADDRESSING_PRODUCER_FAULT_TO**	Signifies the value for the faultAction response WS-Addressing FaultTo header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI

Name	Description	Default	Type
CamelSpringWebserviceAddressingReplyTo (producer) Constant: **SPRING_WS_ADDRESSING_PRODUCER_REPLY_TO**	Signifies the value for the replyTo response WS-Addressing ReplyTo header that is provided by the method. See org.springframework.ws.soap.addressing.server.annotation.Action annotation for more details.		URI
breadcrumbId (consumer) Constant: **BREADCRUMB_ID**	The breadcrumb id.		String

65.6. ACCESSING WEB SERVICES

To call a web service at **http://foo.com/bar** simply define a route:

```
from("direct:example").to("spring-ws:http://foo.com/bar")
```

and send a message:

```
template.requestBody("direct:example", "<foobar xmlns=\"http://foo.com\"><msg>test message</msg></foobar>");
```

If you are calling a SOAP service, you must not include SOAP tags. Spring-WS performs the XML-to-SOAP marshaling.

65.7. SENDING SOAP AND WS-ADDRESSING ACTION HEADERS

When a remote web service requires a SOAP action or use of the WS-Addressing standard, you define your route as:

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?soapAction=http://foo.com&wsAddressingAction=http://bar.com")
```

You can also override the endpoint options with header values.

```
template.requestBodyAndHeader("direct:example",
"<foobar xmlns=\"http://foo.com\"><msg>test message</msg></foobar>",
SpringWebserviceConstants.SPRING_WS_SOAP_ACTION, "http://baz.com");
```

65.8. USING SOAP HEADERS

You can provide the SOAP header as a Camel Message header when sending a message to a spring-ws endpoint. For example given the following SOAP header in a String:

```
String body = ...
String soapHeader = "<h:Header xmlns:h=\"http://www.webserviceX.NET/\">
<h:MessageID>1234567890</h:MessageID><h:Nested><h:NestedID>1111</h:NestedID>
</h:Nested></h:Header>";
```

We can set the body and header on the Camel Message as follows:

```
exchange.getIn().setBody(body);
exchange.getIn().setHeader(SpringWebserviceConstants.SPRING_WS_SOAP_HEADER,
soapHeader);
```

And then send the Exchange to a **spring-ws** endpoint to call the Web Service.

Similarly, the spring-ws consumer also enriches the Camel Message with the SOAP header.

For an example see this unit test.

65.9. THE HEADER AND ATTACHMENT PROPAGATION

Spring WS Camel supports propagation of the headers and attachments into Spring-WS WebServiceMessage response. The endpoint uses a "hook" with the MessageFilter (default implementation is provided by BasicMessageFilter) to propagate the exchange headers and attachments into WebServiceMessage response.

```
exchange.getOut().getHeaders().put("myCustom","myHeaderValue")
exchange.getIn().addAttachment("myAttachment", new DataHandler(...))
```

If the exchange header in the pipeline contains text, it generates Qname(key)=value attribute in the soap header. You must create a QName class directly and put any key into header.

65.10. HOW TO TRANSFORM THE SOAP HEADER USING A STYLESHEET

The header transformation filter (HeaderTransformationMessageFilter.java) can be used to transform the soap header for a soap request. If you want to use the header transformation filter, see the below example:

```
<bean id="headerTransformationFilter"
class="org.apache.camel.component.spring.ws.filter.impl.HeaderTransformationMessageFilter">
    <constructor-arg index="0" value="org/apache/camel/component/spring/ws/soap-header-
transform.xslt"/>
</bean>
```

Use the bead defined above in the camel endpoint

```
<route>
    <from uri="direct:stockQuoteWebserviceHeaderTransformation"/>
    <to uri="spring-ws:http://localhost?
webServiceTemplate=#webServiceTemplate&amp;soapAction=http://www.stockquotes.edu/GetQuote&
amp;messageFilter=#headerTransformationFilter"/>
</route>
```

65.11. HOW TO USE MTOM ATTACHMENTS

The BasicMessageFilter provides all required information for Apache Axiom in order to produce MTOM message. If you want to use Apache Camel Spring WS within Apache Axiom, here is an example: - Define the **messageFactory** as shown below, and Spring-WS populates your SOAP message with optimized attachments through an MTOM strategy.

```
<bean id="axiomMessageFactory"
class="org.springframework.ws.soap.axiom.AxiomSoapMessageFactory">
<property name="payloadCaching" value="false" />
<property name="attachmentCaching" value="true" />
<property name="attachmentCacheThreshold" value="1024" />
</bean>
```

- Add into your pom.xml the following dependencies

```
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-api</artifactId>
<version>1.2.13</version>
</dependency>
<dependency>
<groupId>org.apache.ws.commons.axiom</groupId>
<artifactId>axiom-impl</artifactId>
<version>1.2.13</version>
<scope>runtime</scope>
</dependency>
```

- Add your attachment into the pipeline, for example using a Processor implementation.

```
private class Attachement implements Processor {
public void process(Exchange exchange) throws Exception
{ exchange.getOut().copyFrom(exchange.getIn()); File file = new File("testAttachment.txt");
exchange.getOut().addAttachment("test", new DataHandler(new FileDataSource(file)));  }
}
```

- Define endpoint (producer) as ussual, for example like this:

```
from("direct:send")
.process(new Attachement())
.to("spring-ws:http://localhost:8089/mySoapService?
soapAction=mySoap&messageFactory=axiomMessageFactory");
```

- Your producer now generates MTOM messages with optimized attachments.

65.12. THE CUSTOM HEADER AND ATTACHMENT FILTERING

If you need to provide your custom processing of either headers or attachments, extend existing BasicMessageFilter and override the appropriate methods or write a brand new implementation of the MessageFilter interface.
To use your custom filter, add either a global or a local message filter into your spring context.

- a) the global custom filter that provides the global configuration for all Spring-WS endpoints

```
<bean id="messageFilter" class="your.domain.myMessageFiler" scope="singleton" />
```

or

- b) the local messageFilter directly on the endpoint as follows:

```
to("spring-ws:http://yourdomain.com?messageFilter=#myEndpointSpecificMessageFilter");
```

For more information see CAMEL-5724

If you want to create your own MessageFilter, consider overriding the following methods in the default implementation of MessageFilter in class BasicMessageFilter:

```
protected void doProcessSoapHeader(Message inOrOut, SoapMessage soapMessage)
{your code /*no need to call super*/ }

protected void doProcessSoapAttachements(Message inOrOut, SoapMessage response)
{ your code /*no need to call super*/ }
```

65.13. USING A CUSTOM MESSAGESENDER AND MESSAGEFACTORY

A custom message sender or factory in the registry can be referenced like this:

```
from("direct:example")
.to("spring-ws:http://foo.com/bar?
messageFactory=#messageFactory&messageSender=#messageSender")
```

Spring configuration:

```
<!-- authenticate using HTTP Basic Authentication -->
<bean id="messageSender"
class="org.springframework.ws.transport.http.HttpComponentsMessageSender">
    <property name="credentials">
        <bean class="org.apache.commons.httpclient.UsernamePasswordCredentials">
            <constructor-arg index="0" value="admin"/>
            <constructor-arg index="1" value="secret"/>
        </bean>
    </property>
</bean>

<!-- force use of Sun SAAJ implementation, http://static.springsource.org/spring-
ws/sites/1.5/faq.html#saaj-jboss -->
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
    <property name="messageFactory">
        <bean class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl"/>
    </property>
</bean>
```

65.14. EXPOSING WEB SERVICES

To expose a web service using this component, you first must set up a MessageDispatcher to look for endpoint mappings in a Spring XML file. If you want to run inside a servlet container, you must use a **MessageDispatcherServlet** configured in **web.xml**.

By default the **MessageDispatcherServlet** will look for a Spring XML named **/WEB-INF/spring-ws-servlet.xml**. To use Camel with Spring-WS the only mandatory bean in that XML file is **CamelEndpointMapping**. This bean allows the **MessageDispatcher** to dispatch web service requests to your routes.

web.xml

```
<web-app>
   <servlet>
      <servlet-name>spring-ws</servlet-name>
      <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>spring-ws</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

spring-ws-servlet.xml

```
<bean id="endpointMapping"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointMapping" />

<bean id="wsdl" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">
   <property name="schema">
      <bean class="org.springframework.xml.xsd.SimpleXsdSchema">
         <property name="xsd" value="/WEB-INF/foobar.xsd"/>
      </bean>
   </property>
   <property name="portTypeName" value="FooBar"/>
   <property name="locationUri" value="/"/>
   <property name="targetNamespace" value="http://example.com/"/>
</bean>
```

More information on setting up Spring-WS can be found in Writing Contract-First Web Services. Basically paragraph 3.6 "Implementing the Endpoint" is handled by this component (specifically paragraph 3.6.2 "Routing the Message to the Endpoint" is where **CamelEndpointMapping** comes in). See the Spring Web Services Example included in the Camel distribution.

65.15. ENDPOINT MAPPING IN ROUTES

With the XML configuration in-place, you can now use Camel's DSL to define what web service requests are handled by your endpoint:

The following route receives all web service requests that have a root element named "GetFoo" within the **http://example.com/** namespace.

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

The following route receives web service requests containing the **http://example.com/GetFoo** SOAP action.

```
from("spring-ws:soapaction:http://example.com/GetFoo?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

The following route receives all requests sent to **http://example.com/foobar**.

```
from("spring-ws:uri:http://example.com/foobar?endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

The route below receives requests that contain the element **<foobar>abc</foobar>** anywhere inside the message (and the default namespace).

```
from("spring-ws:xpathresult:abc?expression=//foobar&endpointMapping=#endpointMapping")
.convertBodyTo(String.class).to(mock:example)
```

65.15.1. Alternative configuration, using existing endpoint mappings

For every endpoint with mapping-type **beanname** one bean of type **CamelEndpointDispatcher** with a corresponding name is required in the Registry/ApplicationContext. This bean acts as a bridge between the Camel endpoint and an existing endpoint mapping like **PayloadRootQNameEndpointMapping**.

The use of the **beanname** mapping-type is primarily meant for (legacy) situations where you are already using Spring-WS and have endpoint mappings defined in a Spring XML file. The **beanname** mapping-type allows you to wire your Camel route into an existing endpoint mapping. When you are starting from the beginning, you must define your endpoint mappings as Camel URI's (as illustrated above with **endpointMapping**) since it requires less configuration and is more expressive. You can also use vanilla Spring-WS with the help of annotations.

An example of a route using **beanname**:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="spring-ws:beanname:QuoteEndpointDispatcher" />
        <to uri="mock:example" />
    </route>
</camelContext>

<bean id="legacyEndpointMapping"
class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
    <property name="mappings">
        <props>
            <prop key="{http://example.com/}GetFuture">FutureEndpointDispatcher</prop>
            <prop key="{http://example.com/}GetQuote">QuoteEndpointDispatcher</prop>
        </props>
    </property>
</bean>

<bean id="QuoteEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
<bean id="FutureEndpointDispatcher"
class="org.apache.camel.component.spring.ws.bean.CamelEndpointDispatcher" />
```

65.16. POJO (UN)MARSHALLING

Camel's pluggable data formats offer support for pojo/xml marshalling using libraries such as JAXB, XStream, JibX, Castor and XMLBeans. You can use these data formats in your route to sent and receive pojo's to and from web services.

When *accessing* web services you can marshal the request and unmarshal the response message:

```
JaxbDataFormat jaxb = new JaxbDataFormat(false);
jaxb.setContextPath("com.example.model");

from("direct:example").marshal(jaxb).to("spring-ws:http://foo.com/bar").unmarshal(jaxb);
```

Similarly, when *providing* web services, you can unmarshal XML requests to POJOs and marshal the response message back to XML:

```
from("spring-ws:rootqname:{http://example.com/}GetFoo?
endpointMapping=#endpointMapping").unmarshal(jaxb)
.to("mock:example").marshal(jaxb);
```

65.17. SPRING BOOT AUTO-CONFIGURATION

When using spring-ws with Spring Boot use the following Maven dependency to have support for auto configuration:

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

The component supports 5 options that are listed below.

Name	Description	Default	Type
camel.component.spring-ws.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean

Name	Description	Default	Type
camel.component.spring-ws.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.spring-ws.enabled	Whether to enable auto configuration of the spring-ws component. This is enabled by default.		Boolean
camel.component.spring-ws.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages using Camel's routing error handlers. When the first message is processed then creating and starting the producer takes time and prolong the total processing time of the processing.	false	Boolean

Name	Description	Default	Type
camel.component.spring-ws.use-global-ssl-context-parameters	Enable usage of global SSL context parameters.	false	Boolean

CHAPTER 66. SQL

Both producer and consumer are supported

The SQL component allows you to work with databases using JDBC queries. The difference between this component and JDBC component is that in case of SQL the query is a property of the endpoint and it uses message payload as parameters passed to the query.

This component uses **spring-jdbc** behind the scenes for the actual SQL handling.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-sql</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

The SQL component also supports:

- a JDBC based repository for the Idempotent Consumer EIP pattern. See further below.
- a JDBC based repository for the Aggregator EIP pattern. See further below.

66.1. URI FORMAT

NOTE

This component can be used as a Transactional Client.

The SQL component uses the following endpoint URI notation:

```
sql:select * from table where id=# order by name[?options]
```

You can use named parameters by using :`#name_of_the_parameter` style as shown:

```
sql:select * from table where id=:#myId order by name[?options]
```

When using named parameters, Camel will lookup the names from, in the given precedence:

1. from message body if its a **java.util.Map**
2. from message headers

If a named parameter cannot be resolved, then an exception is thrown.

You can use Simple expressions as parameters as shown:

```
sql:select * from table where id=:#${exchangeProperty.myId} order by name[?options]
```

Notice that the standard ? symbol that denotes the parameters to an SQL query is substituted with the # symbol, because the ? symbol is used to specify options for the endpoint. The ? symbol replacement can be configured on endpoint basis.

You can externalize your SQL queries to files in the classpath or file system as shown:

```
sql:classpath:sql/myquery.sql[?options]
```

And the myquery.sql file is in the classpath and is just a plain text

```
-- this is a comment
select *
from table
where
  id = :#${exchangeProperty.myId}
order by
  name
```

In the file you can use multilines and format the SQL as you wish. And also use comments such as the – dash line.

66.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

66.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

66.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

66.3. COMPONENT OPTIONS

The SQL component supports 5 options, which are listed below.

Name	Description	Default	Type
dataSource (common)	**Autowired** Sets the DataSource to use to communicate with the database.		DataSource
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
usePlaceholder (advanced)	Sets whether to use placeholder and replace all placeholder characters with sign in the SQL queries. This option is default true.	true	boolean

66.4. ENDPOINT OPTIONS

The SQL endpoint is configured using URI syntax:

```
sql:query
```

with the following path and query parameters:

66.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
query (common)	**Required** Sets the SQL query to perform. You can externalize the query by using file: or classpath: as prefix and specify the location of the file.		String

66.4.2. Query Parameters (45 parameters)

Name	Description	Default	Type
allowNamedParameters (common)	Whether to allow using named parameters in the queries.	true	boolean
dataSource (common)	**Autowired** Sets the DataSource to use to communicate with the database at endpoint level.		DataSource
outputClass (common)	Specify the full package and class name to use as conversion when outputType=SelectOne.		String
outputHeader (common)	Store the query result in a header instead of the message body. By default, outputHeader == null and the query result is stored in the message body, any existing content in the message body is discarded. If outputHeader is set, the value is used as the name of the header to store the query result and the original message body is preserved.		String

Name	Description	Default	Type
outputType (common)	Make the output of consumer or producer to SelectList as List of Map, or SelectOne as single Java object in the following way: a) If the query has only single column, then that JDBC Column object is returned. (such as SELECT COUNT( ) FROM PROJECT will return a Long object. b) If the query has more than one column, then it will return a Map of that result. c) If the outputClass is set, then it will convert the query result into an Java bean object by calling all the setters that match the column names. It will assume your class has a default constructor to create instance with. d) If the query resulted in more than one rows, it throws an non-unique result exception. StreamList streams the result of the query using an Iterator. This can be used with the Splitter EIP in streaming mode to process the ResultSet in streaming fashion. Enum values: • SelectOne • SelectList • StreamList	SelectList	SqlOutputType
separator (common)	The separator to use when parameter values is taken from message body (if the body is a String type), to be inserted at # placeholders. Notice if you use named parameters, then a Map type is used instead. The default value is comma.	,	char
breakBatchOnConsumeFail (consumer)	Sets whether to break batch if onConsume failed.	false	boolean
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
expectedUpdateCount (consumer)	Sets an expected update count to validate when using onConsume.	-1	int
maxMessagesPerPoll (consumer)	Sets the maximum number of messages to poll.		int

Name	Description	Default	Type
onConsume (consumer)	After processing each row then this query can be executed, if the Exchange was processed successfully, for example to mark the row as processed. The query can have parameter.		String
onConsumeBatchComplete (consumer)	After processing the entire batch, this query can be executed to bulk update rows etc. The query cannot have parameters.		String
onConsumeFailed (consumer)	After processing each row then this query can be executed, if the Exchange failed, for example to mark the row as failed. The query can have parameter.		String
routeEmptyResultSet (consumer)	Sets whether empty resultset should be allowed to be sent to the next hop. Defaults to false. So the empty resultset will be filtered out.	false	boolean
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
transacted (consumer)	Enables or disables transaction. If enabled then if processing an exchange failed then the consumer breaks out processing any further exchanges to cause a rollback eager.	false	boolean
useIterator (consumer)	Sets how resultset should be delivered to route. Indicates delivery as either a list or individual object. defaults to true.	true	boolean
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

Name	Description	Default	Type
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy
processingStrategy (consumer (advanced))	Allows to plugin to use a custom org.apache.camel.component.sql.SqlProcessingStrategy to execute queries when the consumer has processed the rows/batch.		SqlProcessingStrategy
batch (producer)	Enables or disables batch mode.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
noop (producer)	If set, will ignore the results of the SQL query and use the existing IN message as the OUT message for the continuation of processing.	false	boolean
useMessageBodyForSql (producer)	Whether to use the message body as the SQL and then headers for parameters. If this option is enabled then the SQL in the uri is not used. Note that query parameters in the message body are represented by a question mark instead of a # symbol.	false	boolean
alwaysPopulateStatement (advanced)	If enabled then the populateStatement method from org.apache.camel.component.sql.SqlPrepareStatementStrategy is always invoked, also if there is no expected parameters to be prepared. When this is false then the populateStatement is only invoked if there is 1 or more expected parameters to be set; for example this avoids reading the message body/headers for SQL queries with no parameters.	false	boolean

Name	Description	Default	Type
parametersCount (advanced)	If set greater than zero, then Camel will use this count value of parameters to replace instead of querying via JDBC metadata API. This is useful if the JDBC vendor could not return correct parameters count, then user may override instead.		int
placeholder (advanced)	Specifies a character that will be replaced to in SQL query. Notice, that it is simple String.replaceAll() operation and no SQL parsing is involved (quoted strings will also change).	#	String
prepareStatementStrategy (advanced)	Allows to plugin to use a custom org.apache.camel.component.sql.SqlPrepareStatementStrategy to control preparation of the query and prepared statement.		SqlPrepareStatementStrategy
templateOptions (advanced)	Configures the Spring JdbcTemplate with the key/values from the Map.		Map
usePlaceholder (advanced)	Sets whether to use placeholder and replace all placeholder characters with sign in the SQL queries.	true	boolean
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long

Name	Description	Default	Type
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean

Name	Description	Default	Type
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean

66.5. TREATMENT OF THE MESSAGE BODY

The SQL component tries to convert the message body to an object of **java.util.Iterator** type and then uses this iterator to fill the query parameters (where each query parameter is represented by a # symbol (or configured placeholder) in the endpoint URI). If the message body is not an array or collection, the conversion results in an iterator that iterates over only one object, which is the body itself.

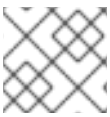
For example, if the message body is an instance of **java.util.List**, the first item in the list is substituted into the first occurrence of # in the SQL query, the second item in the list is substituted into the second occurrence of #, and so on.

If **batch** is set to **true**, then the interpretation of the inbound message body changes slightly – instead of an iterator of parameters, the component expects an iterator that contains the parameter iterators; the size of the outer iterator determines the batch size.

You can use the option **useMessageBodyForSql** that allows to use the message body as the SQL statement, and then the SQL parameters must be provided in a header with the key **SqlConstants.SQL_PARAMETERS**. This allows the SQL component to work more dynamically as the SQL query is from the message body. Use templating (such as Velocity, Freemarker) for conditional processing, e.g. to include or exclude **where** clauses depending on the presence of query parameters.

66.6. RESULT OF THE QUERY

For **select** operations, the result is an instance of **List<Map<String, Object>>** type, as returned by the JdbcTemplate.queryForList() method. For **update** operations, a **NULL** body is returned as the **update** operation is only set as a header and never as a body.



NOTE

See Header Values for more information on the **update** operation.

By default, the result is placed in the message body. If the outputHeader parameter is set, the result is placed in the header. This is an alternative to using a full message enrichment pattern to add headers, it provides a concise syntax for querying a sequence or some other small value into a header. It is convenient to use outputHeader and outputType together:

```
from("jms:order.inbox")
    .to("sql:select order_seq.nextval from dual?outputHeader=OrderId&outputType=SelectOne")
    .to("jms:order.booking");
```

66.7. USING STREAMLIST

The producer supports outputType=StreamList that uses an iterator to stream the output of the query. This allows to process the data in a streaming fashion which for example can be used by the Splitter EIP to process each row one at a time, and load data from the database as needed.

```
from("direct:withSplitModel")
        .to("sql:select * from projects order by id?
outputType=StreamList&outputClass=org.apache.camel.component.sql.ProjectModel")
        .to("log:stream")
        .split(body()).streaming()
            .to("log:row")
            .to("mock:result")
        .end();
```

66.8. HEADER VALUES

When performing **update** operations, the SQL Component stores the update count in the following message headers:

Header	Description
CamelSqlUpdateCount	The number of rows updated for **update** operations, returned as an **Integer** object. This header is not provided when using outputType=StreamList.
CamelSqlRowCount	The number of rows returned for **select** operations, returned as an **Integer** object. This header is not provided when using outputType=StreamList.
CamelSqlQuery	Query to execute. This query takes precedence over the query specified in the endpoint URI. Note that query parameters in the header *are* represented by a **?** instead of a # symbol

When performing **insert** operations, the SQL Component stores the rows with the generated keys and number of these rows in the following message headers:

Header	Description
CamelSqlGeneratedKeysRowCount	The number of rows in the header that contains generated keys.

Header	Description
CamelSqlGeneratedKeyRows	Rows that contains the generated keys (a list of maps of keys).

66.9. GENERATED KEYS

If you insert data using SQL INSERT, then the RDBMS may support auto generated keys. You can instruct the SQL producer to return the generated keys in headers.
To do that set the header **CamelSqlRetrieveGeneratedKeys=true**. Then the generated keys will be provided as headers with the keys listed in the table above.

To specify which generated columns should be retrieved, set the header **CamelSqlGeneratedColumns** to a **String[]** or **int[]**, indicating the column names or indexes, respectively. Some databases requires this, such as Oracle. It may also be necessary to use the **parametersCount** option if the driver cannot correctly determine the number of parameters.

66.10. DATASOURCE

You can set a reference to a **DataSource** in the URI directly:

```
sql:select * from table where id=# order by name?dataSource=#myDS
```

66.11. USING NAMED PARAMETERS

In the given route below, we want to get all the projects from the projects table. Notice the SQL query has 2 named parameters, :#lic and :#min.
Camel will then lookup for these parameters from the message body or message headers. Notice in the example above we set two headers with constant value
for the named parameters:

```
from("direct:projects")
  .setHeader("lic", constant("ASF"))
  .setHeader("min", constant(123))
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

Though if the message body is a **java.util.Map** then the named parameters will be taken from the body.

```
from("direct:projects")
  .to("sql:select * from projects where license = :#lic and id > :#min order by id")
```

66.12. USING EXPRESSION PARAMETERS IN PRODUCERS

In the given route below, we want to get all the project from the database. It uses the body of the exchange for defining the license and uses the value of a property as the second parameter.

```
from("direct:projects")
  .setBody(constant("ASF"))
```

```
    .setProperty("min", constant(123))
    .to("sql:select * from projects where license = :#${body} and id > :#${exchangeProperty.min} order
by id")
```

66.12.1. Using expression parameters in consumers

When using the SQL component as consumer, you can now also use expression parameters (simple language) to build dynamic query parameters, such as calling a method on a bean to retrieve an id, date or something.

For example in the sample below we call the nextId method on the bean myIdGenerator:

```
from("sql:select * from projects where id = :#${bean:myIdGenerator.nextId}")
    .to("mock:result");
```

And the bean has the following method:

```
public static class MyIdGenerator {

    private int id = 1;

    public int nextId() {
        return id++;
    }
```

Notice that there is no existing **Exchange** with message body and headers, so the simple expression you can use in the consumer are most useable for calling bean methods as in this example.

66.13. USING IN QUERIES WITH DYNAMIC VALUES

The SQL producer allows to use SQL queries with IN statements where the IN values is dynamic computed. For example from the message body or a header etc.

To use IN you need to:

- prefix the parameter name with **in:**
- add **( )** around the parameter

An example explains this better. The following query is used:

```
-- this is a comment
select *
from projects
where project in (:#in:names)
order by id
```

In the following route:

```
from("direct:query")
    .to("sql:classpath:sql/selectProjectsIn.sql")
    .to("log:query")
    .to("mock:query");
```

Then the IN query can use a header with the key names with the dynamic values such as:

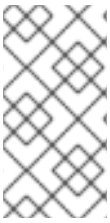
```
// use an array
template.requestBodyAndHeader("direct:query", "Hi there!", "names", new String[]{"Camel", "AMQ"});

// use a list
List<String> names = new ArrayList<String>();
names.add("Camel");
names.add("AMQ");

template.requestBodyAndHeader("direct:query", "Hi there!", "names", names);

// use a string separated values with comma
template.requestBodyAndHeader("direct:query", "Hi there!", "names", "Camel,AMQ");
```

The query can also be specified in the endpoint instead of being externalized (notice that externalizing makes maintaining the SQL queries easier)

```
from("direct:query")
    .to("sql:select * from projects where project in (:#in:names) order by id")
    .to("log:query")
    .to("mock:query");
```

66.14. USING THE JDBC BASED IDEMPOTENT REPOSITORY

In this section we will use the JDBC based idempotent repository.



NOTE

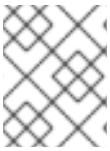
Abstract class
There is an abstract class **org.apache.camel.processor.idempotent.jdbc.AbstractJdbcMessageIdRepository** you can extend to build custom JDBC idempotent repository.

First we have to create the database table which will be used by the idempotent repository. We use the following schema:

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100) )
```

We added the createdAt column:

```
CREATE TABLE CAMEL_MESSAGEPROCESSED ( processorName VARCHAR(255),
messageId VARCHAR(100), createdAt TIMESTAMP )
```



NOTE

The SQL Server **TIMESTAMP** type is a fixed-length binary-string type. It does not map to any of the JDBC time types: **DATE**, **TIME**, or **TIMESTAMP**.

When working with concurrent consumers it is crucial to create a unique constraint on the columns processorName and messageId. Because the syntax for this constraint differs from database to database, we do not show it here.

66.14.1. Customize the JDBC idempotency repository

You have a few options to tune the **org.apache.camel.processor.idempotent.jdbc.JdbcMessageIdRepository** for your needs:

Parameter	Default Value	Description
createTableIfNotExists	true	Defines whether or not Camel should try to create the table if it doesn't exist.
tableName	CAMEL_MESSAGEPROCESSED	To use a custom table name instead of the default name: CAMEL_MESSAGEPROCESSED.
tableExistsString	SELECT 1 FROM CAMEL_MESSAGEPROCESSED WHERE 1 = 0	This query is used to figure out whether the table already exists or not. It must throw an exception to indicate the table doesn't exist.
createString	CREATE TABLE CAMEL_MESSAGEPROCESSED (processorName VARCHAR(255), messageId VARCHAR(100), createdAt TIMESTAMP)	The statement which is used to create the table.
queryString	SELECT COUNT(*) FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	The query which is used to figure out whether the message already exists in the repository (the result is not equals to '0'). It takes two parameters. This first one is the processor name (**String**) and the second one is the message id (**String**).
insertString	INSERT INTO CAMEL_MESSAGEPROCESSED (processorName, messageId, createdAt) VALUES (?, ?, ?)	The statement which is used to add the entry into the table. It takes three parameter. The first one is the processor name (**String**), the second one is the message id (**String**) and the third one is the timestamp (**java.sql.Timestamp**) when this entry was added to the repository.
deleteString	DELETE FROM CAMEL_MESSAGEPROCESSED WHERE processorName = ? AND messageId = ?	The statement which is used to delete the entry from the database. It takes two parameter. This first one is the processor name (**String**) and the second one is the message id (**String**).

The option **tableName** can be used to use the default SQL queries but with a different table name. However if you want to customize the SQL queries then you can configure each of them individually.

66.14.2. Orphan Lock aware Jdbc IdempotentRepository

One of the limitations of **org.apache.camel.processor.idempotent.jdbc.JdbcMessageIdRepository** is that it does not handle orphan locks resulting from JVM crash or non graceful shutdown. This can result in unprocessed files/messages if this is implementation is used with camel-file, camel-ftp etc. if you need to address orphan locks processing then use **org.apache.camel.processor.idempotent.jdbc.JdbcOrphanLockAwareIdempotentRepository**. This repository keeps track of the locks held by an instance of the application. For each lock held, the application will send keep alive signals to the lock repository resulting in updating the createdAt column with the current Timestamp. When an application instance tries to acquire a lock if the, then there are three possibilities exist :

- lock entry does not exist then the lock is provided using the base implementation of **JdbcMessageIdRepository**.
- lock already exists and the createdAt < System.currentTimeMillis() - lockMaxAgeMillis. In this case it is assumed that an active instance has the lock and the lock is not provided to the new instance requesting the lock
- lock already exists and the createdAt > = System.currentTimeMillis() - lockMaxAgeMillis. In this case it is assumed that there is no active instance which has the lock and the lock is provided to the requesting instance. The reason behind is that if the original instance which had the lock, if it was still running, it would have updated the Timestamp on createdAt using its keepAlive mechanism

This repository has two additional configuration parameters

Parameter	Description
lockMaxAgeMillis	This refers to the duration after which the lock is considered orphaned i.e. if the currentTimestamp - createdAt >= lockMaxAgeMillis then lock is orphaned.
lockKeepAliveIntervalMillis	The frequency at which keep alive updates are done to createdAt Timestamp column.

66.14.3. Caching Jdbc IdempotentRepository

Some SQL implementations are not fast on a per query basis. The **JdbcMessageIdRepository** implementation does its idempotent checks individually within SQL transactions. Checking a mere 100 keys can take minutes. The **JdbcCachedMessageIdRepository** preloads an in-memory cache on start with the entire list of keys. This cache is then checked first before passing through to the original implementation.

As with all cache implementations, there are considerations that should be made with regard to stale data and your specific usage.

66.15. USING THE JDBC BASED AGGREGATION REPOSITORY

JdbcAggregationRepository is an **AggregationRepository** which on the fly persists the aggregated messages. This ensures that you will not loose messages, as the default aggregator will use an in memory only **AggregationRepository**. The **JdbcAggregationRepository** allows together with Camel to provide persistent support for the Aggregator.

Only when an Exchange has been successfully processed it will be marked as complete which happens when the **confirm** method is invoked on the **AggregationRepository**. This means if the same Exchange fails again it will be kept retried until it success.

You can use option **maximumRedeliveries** to limit the maximum number of redelivery attempts for a given recovered Exchange. You must also set the **deadLetterUri** option so Camel knows where to send the Exchange when the **maximumRedeliveries** was hit.

You can see some examples in the unit tests of camel-sql, for example **JdbcAggregateRecoverDeadLetterChannelTest.java**

66.15.1. Database

To be operational, each aggregator uses two table: the aggregation and completed one. By convention the completed has the same name as the aggregation one suffixed with **"_COMPLETED"**. The name must be configured in the Spring bean with the **RepositoryName** property. In the following example aggregation will be used.

The table structure definition of both table are identical: in both case a String value is used as key (**id**) whereas a Blob contains the exchange serialized in byte array.
However one difference should be remembered: the **id** field does not have the same content depending on the table.
In the aggregation table **id** holds the correlation Id used by the component to aggregate the messages.
In the completed table, **id** holds the id of the exchange stored in corresponding the blob field.

Here is the SQL query used to create the tables, just replace **"aggregation"** with your aggregator repository name.

```
CREATE TABLE aggregation (
 id varchar(255) NOT NULL,
 exchange blob NOT NULL,
 version BIGINT NOT NULL,
 constraint aggregation_pk PRIMARY KEY (id)
);
CREATE TABLE aggregation_completed (
 id varchar(255) NOT NULL,
 exchange blob NOT NULL,
 version BIGINT NOT NULL,
 constraint aggregation_completed_pk PRIMARY KEY (id)
);
```

66.16. STORING BODY AND HEADERS AS TEXT

You can configure the **JdbcAggregationRepository** to store message body and select(ed) headers as String in separate columns. For example to store the body, and the following two headers **companyName** and **accountName** use the following SQL:

```
CREATE TABLE aggregationRepo3 (
 id varchar(255) NOT NULL,
 exchange blob NOT NULL,
 version BIGINT NOT NULL,
 body varchar(1000),
 companyName varchar(1000),
 accountName varchar(1000),
 constraint aggregationRepo3_pk PRIMARY KEY (id)
```

```
);
CREATE TABLE aggregationRepo3_completed (
 id varchar(255) NOT NULL,
 exchange blob NOT NULL,
 version BIGINT NOT NULL,
 body varchar(1000),
 companyName varchar(1000),
 accountName varchar(1000),
 constraint aggregationRepo3_completed_pk PRIMARY KEY (id)
);
```

And then configure the repository to enable this behavior as shown below:

```
<bean id="repo3"
  class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
  <property name="repositoryName" value="aggregationRepo3"/>
  <property name="transactionManager" ref="txManager3"/>
  <property name="dataSource" ref="dataSource3"/>
  <!-- configure to store the message body and following headers as text in the repo -->
  <property name="storeBodyAsText" value="true"/>
  <property name="headersToStoreAsText">
    <list>
      <value>companyName</value>
      <value>accountName</value>
    </list>
  </property>
</bean>
```

66.16.1. Codec (Serialization)

Since they can contain any type of payload, Exchanges are not serializable by design. It is converted into a byte array to be stored in a database BLOB field. All those conversions are handled by the **JdbcCodec** class. One detail of the code requires your attention: the **ClassLoadingAwareObjectInputStream**.

The **ClassLoadingAwareObjectInputStream** has been reused from the Apache ActiveMQ project. It wraps an **ObjectInputStream** and use it with the **ContextClassLoader** rather than the **currentThread** one. The benefit is to be able to load classes exposed by other bundles. This allows the exchange body and headers to have custom types object references.

66.16.2. Transaction

A Spring **PlatformTransactionManager** is required to orchestrate transaction.

66.16.2.1. Service (Start/Stop)

The **start** method verify the connection of the database and the presence of the required tables. If anything is wrong it will fail during starting.

66.16.3. Aggregator configuration

Depending on the targeted environment, the aggregator might need some configuration. As you already know, each aggregator should have its own repository (with the corresponding pair of table created in the database) and a data source. If the default lobHandler is not adapted to your database system, it can be injected with the **lobHandler** property.

Here is the declaration for Oracle:

```
<bean id="lobHandler" class="org.springframework.jdbc.support.lob.OracleLobHandler">
  <property name="nativeJdbcExtractor" ref="nativeJdbcExtractor"/>
</bean>
<bean id="nativeJdbcExtractor"
  class="org.springframework.jdbc.support.nativejdbc.CommonsDbcpNativeJdbcExtractor"/>
<bean id="repo"
  class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
  <property name="transactionManager" ref="transactionManager"/>
  <property name="repositoryName" value="aggregation"/>
  <property name="dataSource" ref="dataSource"/>
  <!-- Only with Oracle, else use default -->
  <property name="lobHandler" ref="lobHandler"/>
</bean>
```

66.16.4. Optimistic locking

You can turn on **optimisticLocking** and use this JDBC based aggregation repository in a clustered environment where multiple Camel applications shared the same database for the aggregation repository. If there is a race condition there JDBC driver will throw a vendor specific exception which the **JdbcAggregationRepository** can react upon. To know which caused exceptions from the JDBC driver is regarded as an optimistick locking error we need a mapper to do this. Therefore there is a **org.apache.camel.processor.aggregate.jdbc.JdbcOptimisticLockingExceptionMapper** allows you to implement your custom logic if needed. There is a default implementation **org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper** which works as follows:

The following check is done:

- If the caused exception is an **SQLException** then the SQLState is checked if starts with 23.
- If the caused exception is a **DataIntegrityViolationException**
- If the caused exception class name has "ConstraintViolation" in its name.
- Optional checking for FQN class name matches if any class names has been configured.

You can in addition add FQN classnames, and if any of the caused exception (or any nested) equals any of the FQN class names, then its an optimistick locking error.

Here is an example, where we define 2 extra FQN class names from the JDBC vendor.

```
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
  <property name="transactionManager" ref="transactionManager"/>
  <property name="repositoryName" value="aggregation"/>
  <property name="dataSource" ref="dataSource"/>
  <property name="jdbcOptimisticLockingExceptionMapper" ref="myExceptionMapper"/>
</bean>
<!-- use the default mapper with extraFQN class names from our JDBC driver -->
<bean id="myExceptionMapper"
class="org.apache.camel.processor.aggregate.jdbc.DefaultJdbcOptimisticLockingExceptionMapper">
  <property name="classNames">
    <util:set>
```

```
      <value>com.foo.sql.MyViolationExceptoion</value>
      <value>com.foo.sql.MyOtherViolationExceptoion</value>
    </util:set>
  </property>
</bean>
```

66.16.5. Propagation behavior

JdbcAggregationRepository uses two distinct *transaction templates* from Spring-TX. One is read-only and one is used for read-write operations.

However, when using **JdbcAggregationRepository** within a route that itself uses **<transacted />** and there's common **PlatformTransactionManager** used, there may be a need to configure *propagation behavior* used by transaction templates inside **JdbcAggregationRepository**.

Here's a way to do it:

```
<bean id="repo"
class="org.apache.camel.processor.aggregate.jdbc.JdbcAggregationRepository">
  <property name="propagationBehaviorName" value="PROPAGATION_NESTED" />
</bean>
```

Propagation is specified by constants of **org.springframework.transaction.TransactionDefinition** interface, so **propagationBehaviorName** is convenient setter that allows to use names of the constants.

66.16.6. PostgreSQL case

There's special database that may cause problems with optimistic locking used by **JdbcAggregationRepository**. PostgreSQL marks connection as invalid in case of data integrity violation exception (the one with SQLState 23505). This makes the connection effectively unusable within nested transaction. Details can be found in the document..

org.apache.camel.processor.aggregate.jdbc.PostgresAggregationRepository extends **JdbcAggregationRepository** and uses special **INSERT .. ON CONFLICT ..** statement to provide optimistic locking behavior.

This statement is (with default aggregation table definition):

```
INSERT INTO aggregation (id, exchange) values (?, ?) ON CONFLICT DO NOTHING
```

Details can be found in PostgreSQL documentation.

When this clause is used, **java.sql.PreparedStatement.executeUpdate()** call returns **0** instead of throwing SQLException with SQLState=23505. Further handling is exactly the same as with generic **JdbcAggregationRepository**, but without marking PostgreSQL connection as invalid.

66.17. CAMEL SQL STARTER

A starter module is available to spring-boot users. When using the starter, the **DataSource** can be directly configured using spring-boot properties.

```
# Example for a mysql datasource
spring.datasource.url=jdbc:mysql://localhost/test
```

```
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

To use this feature, add the following dependencies to your spring boot pom.xml file:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-sql-starter</artifactId>
  <version>${camel.version}</version> <!-- use the same version as your Camel core version -->
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>${spring-boot-version}</version>
</dependency>
```

You should also include the specific database driver, if needed.

66.18. SPRING BOOT AUTO-CONFIGURATION

When using sql with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-sql-starter</artifactId>
</dependency>
```

The component supports 8 options, which are listed below.

Name	Description	Default	Type
camel.component.sql-stored.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.sql-stored.enabled	Whether to enable auto configuration of the sql-stored component. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.component.sql-stored.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.sql.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.sql.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.sql.enabled	Whether to enable auto configuration of the sql component. This is enabled by default.		Boolean
camel.component.sql.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.sql.use-placeholder	Sets whether to use placeholder and replace all placeholder characters with sign in the SQL queries. This option is default true.	true	Boolean

CHAPTER 67. STUB

Both producer and consumer are supported

The Stub component provides a simple way to stub out any physical endpoints while in development or testing, allowing you for example to run a route without needing to actually connect to a specific specific SMTP or HTTP endpoint. Just add **stub:** in front of any endpoint URI to stub out the endpoint.

Internally the Stub component creates VM endpoints. The main difference between Stub and VM is that VM will validate the URI and parameters you give it, so putting vm: in front of a typical URI with query arguments will usually fail. Stub won't though, as it basically ignores all query parameters to let you quickly stub out one or more endpoints in your route temporarily.

67.1. URI FORMAT

```
stub:someUri
```

Where **someUri** can be any URI with any query parameters.

67.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

67.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

67.2.1.1. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

67.3. COMPONENT OPTIONS

The Stub component supports 10 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Sets the default number of concurrent threads processing exchanges.	1	int
defaultPollTimeout (consumer (advanced))	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	int
defaultBlockWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	boolean
defaultDiscardWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	boolean
defaultOfferTimeout (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, where a configured timeout can be added to the block case. Utilizing the .offer(timeout) method of the underlining java queue.		long

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
defaultQueueFactory (advanced)	Sets the default queue factory.		BlockingQueueFactory
queueSize (advanced)	Sets the default maximum capacity of the SEDA queue (i.e., the number of messages it can hold).	1000	int

67.4. ENDPOINT OPTIONS

The Stub endpoint is configured using URI syntax:

```
stub:name
```

with the following path and query parameters:

67.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
name (common)	**Required** Name of queue.		String

67.4.2. Query Parameters (18 parameters)

Name	Description	Default	Type
size (common)	The maximum capacity of the SEDA queue (i.e., the number of messages it can hold). Will by default use the defaultSize set on the SEDA component.	1000	int
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
concurrentConsumers (consumer)	Number of concurrent threads processing exchanges.	1	int
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
limitConcurrentConsumers (consumer (advanced))	Whether to limit the number of concurrentConsumers to the maximum of 500. By default, an exception will be thrown if an endpoint is configured with a greater number. You can disable that check by turning this option off.	true	boolean
multipleConsumers (consumer (advanced))	Specifies whether multiple consumers are allowed. If enabled, you can use SEDA for Publish-Subscribe messaging. That is, you can send a message to the SEDA queue and have each consumer receive a copy of the message. When enabled, this option should be specified on every consumer endpoint.	false	boolean

Name	Description	Default	Type
pollTimeout (consumer (advanced))	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	int
purgeWhenStopping (consumer (advanced))	Whether to purge the task queue when stopping the consumer/route. This allows to stop faster, as any pending messages on the queue is discarded.	false	boolean
blockWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	boolean
discardIfNoConsumers (producer)	Whether the producer should discard the message (do not add the message to the queue), when sending to a queue with no active consumers. Only one of the options discardIfNoConsumers and failIfNoConsumers can be enabled at the same time.	false	boolean
discardWhenFull (producer)	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	boolean
failIfNoConsumers (producer)	Whether the producer should fail by throwing an exception, when sending to a queue with no active consumers. Only one of the options discardIfNoConsumers and failIfNoConsumers can be enabled at the same time.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean

Name	Description	Default	Type
offerTimeout (producer)	Offer timeout (in milliseconds) can be added to the block case when queue is full. You can disable timeout by using 0 or a negative value.		long
timeout (producer)	Timeout (in milliseconds) before a SEDA producer will stop waiting for an asynchronous task to complete. You can disable timeout by using 0 or a negative value.	30000	long
waitForTaskToComplete (producer)	Option to specify whether the caller should wait for the async task to complete or not before continuing. The following three options are supported: Always, Never or IfReplyExpected. The first two values are self-explanatory. The last value, IfReplyExpected, will only wait if the message is Request Reply based. The default option is IfReplyExpected. Enum values: • Never • IfReplyExpected • Always	IfReplyExpected	WaitForTaskToComplete
queue (advanced)	Define the queue instance which will be used by the endpoint.		BlockingQueue

67.5. EXAMPLES

Here are a few samples of stubbing endpoint uris

```
stub:smtp://somehost.foo.com?user=whatnot&something=else
stub:http://somehost.bar.com/something
```

67.6. SPRING BOOT AUTO-CONFIGURATION

When using stub with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-stub-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.component.stub.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.stub.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.stub.concurrent-consumers	Sets the default number of concurrent threads processing exchanges.	1	Integer
camel.component.stub.default-block-when-full	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will instead block and wait until the message can be accepted.	false	Boolean
camel.component.stub.default-discard-when-full	Whether a thread that sends messages to a full SEDA queue will be discarded. By default, an exception will be thrown stating that the queue is full. By enabling this option, the calling thread will give up sending and continue, meaning that the message was not sent to the SEDA queue.	false	Boolean
camel.component.stub.default-offer-timeout	Whether a thread that sends messages to a full SEDA queue will block until the queue's capacity is no longer exhausted. By default, an exception will be thrown stating that the queue is full. By enabling this option, where a configured timeout can be added to the block case. Utilizing the .offer(timeout) method of the underlining java queue.		Long
camel.component.stub.default-poll-timeout	The timeout (in milliseconds) used when polling. When a timeout occurs, the consumer can check whether it is allowed to continue running. Setting a lower value allows the consumer to react more quickly upon shutdown.	1000	Integer

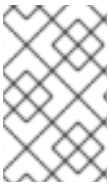
Name	Description	Default	Type
camel.component.stub.default-queue-factory	Sets the default queue factory. The option is a org.apache.camel.component.seda.BlockingQueueFactory<org.apache.camel.Exchange> type.		BlockingQueueFactory
camel.component.stub.enabled	Whether to enable auto configuration of the stub component. This is enabled by default.		Boolean
camel.component.stub.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.stub.queue-size	Sets the default maximum capacity of the SEDA queue (i.e., the number of messages it can hold).	1000	Integer

CHAPTER 68. TELEGRAM

Both producer and consumer are supported

The Telegram component provides access to the Telegram Bot API. It allows a Camel-based application to send and receive messages by acting as a Bot, participating in direct conversations with normal users, private and public groups or channels.

A Telegram Bot must be created before using this component, following the instructions at the Telegram Bot developers home. When a new Bot is created, the BotFather provides an **authorization token** corresponding to the Bot. The authorization token is a mandatory parameter for the camel-telegram endpoint.



NOTE

In order to allow the Bot to receive all messages exchanged within a group or channel (not just the ones starting with a '/' character), ask the BotFather to **disable the privacy mode**, using the **/setprivacy** command.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-telegram</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

68.1. URI FORMAT

```
telegram:type[?options]
```

68.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

68.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

68.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

68.3. COMPONENT OPTIONS

The Telegram component supports 7 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
baseUri (advanced)	Can be used to set an alternative base URI, e.g. when you want to test the component against a mock Telegram API.		String
client (advanced)	To use a custom AsyncHttpClient.		AsyncHttpClient
clientConfig (advanced)	To configure the AsyncHttpClient to use a custom com.ning.http.client.AsyncHttpClientConfig instance.		AsyncHttpClientConfig
authorizationToken (security)	The default Telegram authorization token to be used when the information is not provided in the endpoints.		String

68.4. ENDPOINT OPTIONS

The Telegram endpoint is configured using URI syntax:

```
telegram:type
```

with the following path and query parameters:

68.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
type (common)	**Required** The endpoint type. Currently, only the 'bots' type is supported. Enum values: • bots		String

68.4.2. Query Parameters (30 parameters)

Name	Description	Default	Type

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
limit (consumer)	Limit on the number of updates that can be received in a single polling request.	100	Integer
sendEmptyMessageWhenIdle (consumer)	If the polling consumer did not poll any files, you can enable this option to send an empty message (no body) instead.	false	boolean
timeout (consumer)	Timeout in seconds for long polling. Put 0 for short polling or a bigger number for long polling. Long polling produces shorter response time.	30	Integer
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
pollStrategy (consumer (advanced))	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 have been created and being routed in Camel.		PollingConsumerPollStrategy

Name	Description	Default	Type
chatId (producer)	The identifier of the chat that will receive the produced messages. Chat ids can be first obtained from incoming messages (eg. when a telegram user starts a conversation with a bot, its client sends automatically a '/start' message containing the chat id). It is an optional parameter, as the chat id can be set dynamically for each outgoing message (using body or headers).		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
baseUri (advanced)	Can be used to set an alternative base URI, e.g. when you want to test the component against a mock Telegram API.		String
bufferSize (advanced)	The initial in-memory buffer size used when transferring data between Camel and AHC Client.	4096	int
clientConfig (advanced)	To configure the AsyncHttpClient to use a custom com.ning.http.client.AsyncHttpClientConfig instance.		AsyncHttpClientConfig
proxyHost (proxy)	HTTP proxy host which could be used when sending out the message.		String
proxyPort (proxy)	HTTP proxy port which could be used when sending out the message.		Integer
proxyType (proxy)	HTTP proxy type which could be used when sending out the message. Enum values: • HTTP • SOCKS4 • SOCKS5	HTTP	TelegramProxyType

Name	Description	Default	Type
backoffErrorThreshold (scheduler)	The number of subsequent error polls (failed due some error) that should happen before the backoffMultipler should kick-in.		int
backoffIdleThreshold (scheduler)	The number of subsequent idle polls that should happen before the backoffMultipler should kick-in.		int
backoffMultiplier (scheduler)	To let the scheduled polling consumer backoff 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.		int
delay (scheduler)	Milliseconds before the next poll.	500	long
greedy (scheduler)	If greedy is enabled, then the ScheduledPollConsumer will run immediately again, if the previous run polled 1 or more messages.	false	boolean
initialDelay (scheduler)	Milliseconds before the first poll starts.	1000	long
repeatCount (scheduler)	Specifies a maximum limit of number of fires. So if you set it to 1, the scheduler will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.	0	long
runLoggingLevel (scheduler)	The consumer logs a start/complete log line when it polls. This option allows you to configure the logging level for that. Enum values: • TRACE • DEBUG • INFO • WARN • ERROR • OFF	TRACE	LoggingLevel

Name	Description	Default	Type
scheduledExecutorService (scheduler)	Allows for configuring a custom/shared thread pool to use for the consumer. By default each consumer has its own single threaded thread pool.		ScheduledExecutorService
scheduler (scheduler)	To use a cron scheduler from either camel-spring or camel-quartz component. Use value spring or quartz for built in scheduler.	none	Object
schedulerProperties (scheduler)	To configure additional properties when using a custom scheduler or any of the Quartz, Spring based scheduler.		Map
startScheduler (scheduler)	Whether the scheduler should be auto started.	true	boolean
timeUnit (scheduler)	Time unit for initialDelay and delay options. Enum values: • NANOSECONDS • MICROSECONDS • MILLISECONDS • SECONDS • MINUTES • HOURS • DAYS	MILLISECONDS	TimeUnit
useFixedDelay (scheduler)	Controls if fixed delay or fixed rate is used. See ScheduledExecutorService in JDK for details.	true	boolean
authorizationToken (security)	**Required** The authorization token for using the bot (ask the BotFather).		String

68.4.3. Message Headers

Name	Description
CamelTelegramChatId	This header is used by the producer endpoint in order to resolve the chat id that will receive the message. The recipient chat id can be placed (in order of priority) in message body, in the **CamelTelegramChatId** header or in the endpoint configuration (**chatId** option). This header is also present in all incoming messages.

Name	Description
CamelTelegramMediaType	This header is used to identify the media type when the outgoing message is composed of pure binary data. Possible values are strings or enum values belonging to the **org.apache.camel.component.telegram.TelegramMediaType** enumeration.
CamelTelegramMediaTitleCaption	This header is used to provide a caption or title for outgoing binary messages.
CamelTelegramParseMode	This header is used to format text messages using HTML or Markdown (see **org.apache.camel.component.telegram.TelegramParseMode**).

68.5. USAGE

The Telegram component supports both consumer and producer endpoints. It can also be used in **reactive chat-bot mode** (to consume, then produce messages).

68.6. PRODUCER EXAMPLE

The following is a basic example of how to send a message to a Telegram chat through the Telegram Bot API.

in Java DSL

```
from("direct:start").to("telegram:bots?
authorizationToken=123456789:insertYourAuthorizationTokenHere");
```

or in Spring XML

```
<route>
    <from uri="direct:start"/>
    <to uri="telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere"/>
<route>
```

The code **123456789:insertYourAuthorizationTokenHere** is the **authorization token** corresponding to the Bot.

When using the producer endpoint without specifying the **chat id** option, the target chat will be identified using information contained in the body or headers of the message. The following message bodies are allowed for a producer endpoint (messages of type **OutgoingXXXMessage** belong to the package **org.apache.camel.component.telegram.model**)

Java Type	Description
OutgoingTextMessage	To send a text message to a chat
OutgoingPhotoMessage	To send a photo (JPG, PNG) to a chat

Java Type	Description
OutgoingAudioMessage	To send a mp3 audio to a chat
OutgoingVideoMessage	To send a mp4 video to a chat
OutgoingDocumentMessage	To send a file to a chat (any media type)
OutgoingStickerMessage	To send a sticker to a chat (WEBP)
OutgoingAnswerInlineQuery	To send answers to an inline query
EditMessageTextMessage	To edit text and game messages (editMessageText)
EditMessageCaptionMessage	To edit captions of messages (editMessageCaption)
EditMessageMediaMessage	To edit animation, audio, document, photo, or video messages. (editMessageMedia)
EditMessageReplyMarkupMessage	To edit only the reply markup of message. (editMessageReplyMarkup)
EditMessageDelete	To delete a message, including service messages. (deleteMessage)
SendLocationMessage	To send a location (setSendLocation)
EditMessageLiveLocationMessage	To send changes to a live location (editMessageLiveLocation)
StopMessageLiveLocationMessage	To stop updating a live location message sent by the bot or via the bot (for inline bots) before live_period expires (stopMessageLiveLocation)
SendVenueMessage	To send information about a venue (sendVenue)
byte[]	To send any media type supported. It requires the **CamelTelegramMediaType** header to be set to the appropriate media type
String	To send a text message to a chat. It gets converted automatically into a **OutgoingTextMessage**

68.7. CONSUMER EXAMPLE

The following is a basic example of how to receive all messages that telegram users are sending to the configured Bot. In Java DSL

```
from("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere")
.bean(ProcessorBean.class)
```

or in Spring XML

```
<route>
   <from uri="telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere"/>
   <bean ref="myBean" />
<route>

<bean id="myBean" class="com.example.MyBean"/>
```

The **MyBean** is a simple bean that will receive the messages

```
public class MyBean {

   public void process(String message) {
      // or Exchange, or org.apache.camel.component.telegram.model.IncomingMessage (or both)

      // do process
   }

}
```

Supported types for incoming messages are

Java Type	Description
IncomingMessage	The full object representation of an incoming message
String	The content of the message, for text messages only

68.8. REACTIVE CHAT-BOT EXAMPLE

The reactive chat-bot mode is a simple way of using the Camel component to build a simple chat bot that replies directly to chat messages received from the Telegram users.

The following is a basic configuration of the chat-bot in Java DSL

```
from("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere")
.bean(ChatBotLogic.class)
.to("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere");
```

or in Spring XML

```
<route>
   <from uri="telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere"/>
   <bean ref="chatBotLogic" />
   <to uri="telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere"/>
<route>

<bean id="chatBotLogic" class="com.example.ChatBotLogic"/>
```

The **ChatBotLogic** is a simple bean that implements a generic String-to-String method.

```
public class ChatBotLogic {

    public String chatBotProcess(String message) {
        if( "do-not-reply".equals(message) ) {
            return null; // no response in the chat
        }

        return "echo from the bot: " + message; // echoes the message
    }

}
```

Every non-null string returned by the **chatBotProcess** method is automatically routed to the chat that originated the request (as the **CamelTelegramChatId** header is used to route the message).

68.9. GETTING THE CHAT ID

If you want to push messages to a specific Telegram chat when an event occurs, you need to retrieve the corresponding chat ID. The chat ID is not currently shown in the telegram client, but you can obtain it using a simple route.

First, add the bot to the chat where you want to push messages, then run a route like the following one.

```
from("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere")
.to("log:INFO?showHeaders=true");
```

Any message received by the bot will be dumped to your log together with information about the chat (**CamelTelegramChatId** header).

Once you get the chat ID, you can use the following sample route to push message to it.

```
from("timer:tick")
.setBody().constant("Hello")
to("telegram:bots?
authorizationToken=123456789:insertYourAuthorizationTokenHere&chatId=123456")
```

Note that the corresponding URI parameter is simply **chatId**.

68.10. CUSTOMIZING KEYBOARD

You can customize the user keyboard instead of asking him to write an option. **OutgoingTextMessage** has the property **ReplyMarkup** which can be used for such thing.

```
from("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere")
    .process(exchange -> {

        OutgoingTextMessage msg = new OutgoingTextMessage();
        msg.setText("Choose one option!");

        InlineKeyboardButton buttonOptionOneI = InlineKeyboardButton.builder()
                .text("Option One - I").build();
```

```
        InlineKeyboardButton buttonOptionOneII = InlineKeyboardButton.builder()
                .text("Option One - II").build();

        InlineKeyboardButton buttonOptionTwoI = InlineKeyboardButton.builder()
                .text("Option Two - I").build();

        ReplyKeyboardMarkup replyMarkup = ReplyKeyboardMarkup.builder()
                .keyboard()
                    .addRow(Arrays.asList(buttonOptionOneI, buttonOptionOneII))
                    .addRow(Arrays.asList(buttonOptionTwoI))
                    .close()
                .oneTimeKeyboard(true)
                .build();

        msg.setReplyMarkup(replyMarkup);

        exchange.getIn().setBody(msg);
    })
    .to("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere");
```

If you want to disable it the next message must have the property **removeKeyboard** set on **ReplyKeyboardMarkup** object.

```
from("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere")
    .process(exchange -> {

        OutgoingTextMessage msg = new OutgoingTextMessage();
        msg.setText("Your answer was accepted!");

        ReplyKeyboardMarkup replyMarkup = ReplyKeyboardMarkup.builder()
                .removeKeyboard(true)
                .build();

        msg.setReplyKeyboardMarkup(replyMarkup);

        exchange.getIn().setBody(msg);
    })
    .to("telegram:bots?authorizationToken=123456789:insertYourAuthorizationTokenHere");
```

68.11. WEBHOOK MODE

The Telegram component supports usage in the **webhook mode** using the **camel-webhook** component.

In order to enable webhook mode, users need first to add a REST implementation to their application. Maven users, for example, can add **netty-http** to their **pom.xml** file:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-netty-http</artifactId>
    <version>{CamelSBVersion}</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

Once done, you need to prepend the webhook URI to the telegram URI you want to use.

In Java DSL:

```
from("webhook:telegram:bots?
authorizationToken=123456789:insertYourAuthorizationTokenHere").to("log:info");
```

Some endpoints will be exposed by your application and Telegram will be configured to send messages to them. You need to ensure that your server is exposed to the internet and to pass the right value of the **camel.component.webhook.configuration.webhook-external-url** property.

Refer to the **camel-webhook** component documentation for instructions on how to set it.

68.12. SPRING BOOT AUTO-CONFIGURATION

When using telegram with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-telegram-starter</artifactId>
</dependency>
```

The component supports 8 options, which are listed below.

Name	Description	Default	Type
camel.component.telegram.authorization-token	The default Telegram authorization token to be used when the information is not provided in the endpoints.		String
camel.component.telegram.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.telegram.base-uri	Can be used to set an alternative base URI, e.g. when you want to test the component against a mock Telegram API.		String

Name	Description	Default	Type
camel.component.telegram.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.telegram.client	To use a custom AsyncHttpClient. The option is a org.asynchttpclient.AsyncHttpClient type.		AsyncHttpClient
camel.component.telegram.client-config	To configure the AsyncHttpClient to use a custom com.ning.http.client.AsyncHttpClientConfig instance. The option is a org.asynchttpclient.AsyncHttpClientConfig type.		AsyncHttpClientConfig
camel.component.telegram.enabled	Whether to enable auto configuration of the telegram component. This is enabled by default.		Boolean
camel.component.telegram.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean

CHAPTER 69. TIMER

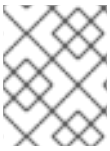
Only consumer is supported

The Timer component is used to generate message exchanges when a timer fires You can only consume events from this endpoint.

69.1. URI FORMAT

```
timer:name[?options]
```

Where **name** is the name of the **Timer** object, which is created and shared across endpoints. So if you use the same name for all your timer endpoints, only one **Timer** object and thread will be used.



NOTE

The IN body of the generated exchange is **null**. So **exchange.getIn().getBody()** returns **null**.

NOTE

Advanced Scheduler
See also the Quartz component that supports much more advanced scheduling.

69.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

69.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

69.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

69.3. COMPONENT OPTIONS

The Timer component supports 2 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

69.4. ENDPOINT OPTIONS

The Timer endpoint is configured using URI syntax:

```
timer:timerName
```

with the following path and query parameters:

69.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
timerName (consumer)	**Required** The name of the timer.		String

69.4.2. Query Parameters (13 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
delay (consumer)	Delay before first event is triggered.	1000	long
fixedRate (consumer)	Events take place at approximately regular intervals, separated by the specified period.	false	boolean
includeMetadata (consumer)	Whether to include metadata in the exchange such as fired time, timer name, timer count etc. This information is default included.	true	boolean
period (consumer)	If greater than 0, generate periodic events every period.	1000	long
repeatCount (consumer)	Specifies a maximum limit of number of fires. So if you set it to 1, the timer will only fire once. If you set it to 5, it will only fire five times. A value of zero or negative means fire forever.		long
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern
daemon (advanced)	Specifies whether or not the thread associated with the timer endpoint runs as a daemon. The default value is true.	true	boolean

Name	Description	Default	Type
pattern (advanced)	Allows you to specify a custom Date pattern to use for setting the time option using URI syntax.		String
synchronous (advanced)	Sets whether synchronous processing should be strictly used.	false	boolean
time (advanced)	A java.util.Date the first event should be generated. If using the URI, the pattern expected is: yyyy-MM-dd HH:mm:ss or yyyy-MM-dd'T'HH:mm:ss.		Date
timer (advanced)	To use a custom Timer.		Timer

69.5. EXCHANGE PROPERTIES

When the timer is fired, it adds the following information as properties to the **Exchange**:

Name	Type	Description
Exchange.TIMER_NAME	**String**	The value of the **name** option.
Exchange.TIMER_TIME	**Date**	The value of the **time** option.
Exchange.TIMER_PERIOD	**long**	The value of the **period** option.
Exchange.TIMER_FIRED_TIME	**Date**	The time when the consumer fired.
Exchange.TIMER_COUNTER	**Long**	The current fire counter. Starts from 1.

69.6. SAMPLE

To set up a route that generates an event every 60 seconds:

```
from("timer://foo?fixedRate=true&period=60000").to("bean:myBean?method=someMethodName");
```

The above route will generate an event and then invoke the **someMethodName** method on the bean called **myBean** in the Registry.

And the route in Spring DSL:

```
<route>
  <from uri="timer://foo?fixedRate=true&amp;period=60000"/>
  <to uri="bean:myBean?method=someMethodName"/>
```

```
</route>
```

69.7. FIRING AS SOON AS POSSIBLE

Since Camel 2.17

You may want to fire messages in a Camel route as soon as possible you can use a negative delay:

```
<route>
  <from uri="timer://foo?delay=-1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

In this way the timer will fire messages immediately.

You can also specify a repeatCount parameter in conjunction with a negative delay to stop firing messages after a fixed number has been reached.

If you don't specify a repeatCount then the timer will continue firing messages until the route will be stopped.

69.8. FIRING ONLY ONCE

You may want to fire a message in a Camel route only once, such as when starting the route. To do that you use the repeatCount option as shown:

```
<route>
  <from uri="timer://foo?repeatCount=1"/>
  <to uri="bean:myBean?method=someMethodName"/>
</route>
```

69.9. SPRING BOOT AUTO-CONFIGURATION

When using timer with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-timer-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type

Name	Description	Default	Type
camel.component.timer.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.timer.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.timer.enabled	Whether to enable auto configuration of the timer component. This is enabled by default.		Boolean

CHAPTER 70. VALIDATOR

Only producer is supported

The Validation component performs XML validation of the message body using the JAXP Validation API and based on any of the supported XML schema languages, which defaults to XML Schema

Note that the component also supports the following useful schema languages:

- RelaxNG Compact Syntax
- RelaxNG XML Syntax

The MSV component also supports RelaxNG XML Syntax.

70.1. URI FORMAT

```
validator:someLocalOrRemoteResource
```

Where **someLocalOrRemoteResource** is some URL to a local resource on the classpath or a full URL to a remote resource or resource on the file system which contains the XSD to validate against. For example:

- **msv:org/foo/bar.xsd**
- **msv:file:../foo/bar.xsd**
- **msv:http://acme.com/cheese.xsd**
- **validator:com/mypackage/myschema.xsd**

The Validation component is provided directly in the camel-core.

70.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

70.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

70.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

70.3. COMPONENT OPTIONS

The Validator component supports 3 options, which are listed below.

Name	Description	Default	Type
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
resourceResolverFactory (advanced)	To use a custom LSResourceResolver which depends on a dynamic endpoint resource URI.		ValidatorResourceResolverFactory

70.4. ENDPOINT OPTIONS

The Validator endpoint is configured using URI syntax:

```
validator:resourceUri
```

with the following path and query parameters:

70.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
resourceUri (producer)	**Required** URL to a local resource on the classpath, or a reference to lookup a bean in the Registry, or a full URL to a remote resource or resource on the file system which contains the XSD to validate against.		String

70.4.2. Query Parameters (10 parameters)

Name	Description	Default	Type
failOnNullBody (producer)	Whether to fail if no body exists.	true	boolean
failOnNullHeader (producer)	Whether to fail if no header exists when validating against a header.	true	boolean
headerName (producer)	To validate against a header instead of the message body.		String
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
errorHandler (advanced)	To use a custom org.apache.camel.processor.validation.ValidatorErrorHandler. The default error handler captures the errors and throws an exception.		ValidatorErrorHandler
resourceResolver (advanced)	To use a custom LSResourceResolver. Do not use together with resourceResolverFactory.		LSResourceResolver

Name	Description	Default	Type
resourceResolverFactory (advanced)	To use a custom LSResourceResolver which depends on a dynamic endpoint resource URI. The default resource resolver factory resturns a resource resolver which can read files from the class path and file system. Do not use together with resourceResolver.		ValidatorResourceResolverFactory
schemaFactory (advanced)	To use a custom javax.xml.validation.SchemaFactory.		SchemaFactory
schemaLanguage (advanced)	Configures the W3C XML Schema Namespace URI.	http://www.w3.org/2001/XMLSchema	String
useSharedSchema (advanced)	Whether the Schema instance should be shared or not. This option is introduced to work around a JDK 1.6.x bug. Xerces should not have this issue.	true	boolean

70.5. EXAMPLE

The following example shows how to configure a route from endpoint **direct:start** which then goes to one of two endpoints, either **mock:valid** or **mock:invalid** based on whether or not the XML matches the given schema (which is supplied on the classpath).

70.6. ADVANCED: JMX METHOD CLEARCACHEDSCHEMA

You can force that the cached schema in the validator endpoint is cleared and reread with the next process call with the JMX operation **clearCachedSchema**. You can also use this method to programmatically clear the cache. This method is available on the **ValidatorEndpoint** class.

70.7. SPRING BOOT AUTO-CONFIGURATION

When using validator with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-validator-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.component.validator.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.validator.enabled	Whether to enable auto configuration of the validator component. This is enabled by default.		Boolean
camel.component.validator.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.validator.resource-resolver-factory	To use a custom LSResourceResolver which depends on a dynamic endpoint resource URI. The option is a org.apache.camel.component.validator.ValidatorResourceResolverFactory type.		ValidatorResourceResolverFactory

CHAPTER 71. WEBHOOK

Only consumer is supported

The Webhook meta component allows other Camel components to configure webhooks on a remote webhook provider and listening for them.

The following components currently provide webhook endpoints:

- **Telegram**

Maven users can add the following dependency to their **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-webhook</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

Typically, other components that support webhook will bring this dependency transitively.

71.1. URI FORMAT

```
webhook:endpoint[?options]
```

71.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

71.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

71.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

71.3. COMPONENT OPTIONS

The Webhook component supports 8 options, which are listed below.

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
webhookAutoRegister (consumer)	Automatically register the webhook at startup and unregister it on shutdown.	true	boolean
webhookBasePath (consumer)	The first (base) path element where the webhook will be exposed. It's a good practice to set it to a random string, so that it cannot be guessed by unauthorized parties.		String
webhookComponentName (consumer)	The Camel Rest component to use for the REST transport, such as netty-http.		String
webhookExternalUrl (consumer)	The URL of the current service as seen by the webhook provider.		String
webhookPath (consumer)	The path where the webhook endpoint will be exposed (relative to basePath, if any).		String
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean

Name	Description	Default	Type
configuration (advanced)	Set the default configuration for the webhook meta-component.		WebhookConfiguration

71.4. ENDPOINT OPTIONS

The Webhook endpoint is configured using URI syntax:

```
webhook:endpointUri
```

with the following path and query parameters:

71.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
endpointUri (consumer)	**Required** The delegate uri. Must belong to a component that supports webhooks.		String

71.4.2. Query Parameters (8 parameters)

Name	Description	Default	Type
bridgeErrorHandler (consumer)	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	boolean
webhookAutoRegister (consumer)	Automatically register the webhook at startup and unregister it on shutdown.	true	boolean
webhookBasePath (consumer)	The first (base) path element where the webhook will be exposed. It's a good practice to set it to a random string, so that it cannot be guessed by unauthorized parties.		String

Name	Description	Default	Type
webhookComponentName (consumer)	The Camel Rest component to use for the REST transport, such as netty-http.		String
webhookExternalUrl (consumer)	The URL of the current service as seen by the webhook provider.		String
webhookPath (consumer)	The path where the webhook endpoint will be exposed (relative to basePath, if any).		String
exceptionHandler (consumer (advanced))	To let the consumer use a custom ExceptionHandler. Notice if the option bridgeErrorHandler is enabled then this option is not in use. By default the consumer will deal with exceptions, that will be logged at WARN or ERROR level and ignored.		ExceptionHandler
exchangePattern (consumer (advanced))	Sets the exchange pattern when the consumer creates an exchange. Enum values: • InOnly • InOut • InOptionalOut		ExchangePattern

71.5. EXAMPLES

Examples of webhook component are provided in the documentation of the delegate components that support it.

71.6. SPRING BOOT AUTO-CONFIGURATION

When using webhook with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-webhook-starter</artifactId>
</dependency>
```

The component supports 9 options, which are listed below.

Name	Description	Default	Type
camel.component.webhook.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.webhook.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.webhook.configuration	Set the default configuration for the webhook meta-component. The option is a org.apache.camel.component.webhook.WebhookConfiguration type.		WebhookConfiguration
camel.component.webhook.enabled	Whether to enable auto configuration of the webhook component. This is enabled by default.		Boolean
camel.component.webhook.webhook-auto-register	Automatically register the webhook at startup and unregister it on shutdown.	true	Boolean
camel.component.webhook.webhook-base-path	The first (base) path element where the webhook will be exposed. It's a good practice to set it to a random string, so that it cannot be guessed by unauthorized parties.		String
camel.component.webhook.webhook-component-name	The Camel Rest component to use for the REST transport, such as netty-http.		String
camel.component.webhook.webhook-external-url	The URL of the current service as seen by the webhook provider.		String
camel.component.webhook.webhook-path	The path where the webhook endpoint will be exposed (relative to basePath, if any).		String

CHAPTER 72. XSLT

Only producer is supported

The XSLT component allows you to process a message using an XSLT template. This can be ideal when using Templating to generate response for requests.

72.1. URI FORMAT

```
xslt:templateName[?options]
```

The URI format contains **templateName**, which can be one of the following:

- the classpath-local URI of the template to invoke
- the complete URL of the remote template.

You can append query options to the URI in the following format:

?option=value&option=value&…

Table 72.1. Table 1. Example URIs

URI	Description
xslt:com/acme/mytransform.xsl	Refers to the file com/acme/mytransform.xsl on the classpath
xslt:file:///foo/bar.xsl	Refers to the file /foo/bar.xsl
xslt:http://acme.com/cheese/foo.xsl	Refers to the remote http resource

72.2. CONFIGURING OPTIONS

Camel components are configured on two separate levels:

- component level
- endpoint level

72.2.1. Configuring Component Options

The component level is the highest level which holds general and common configurations that are inherited by the endpoints. For example a component may have security settings, credentials for authentication, urls for network connection and so forth.

Some components only have a few options, and others may have many. Because components typically have pre configured defaults that are commonly used, then you may often only need to configure a few options on a component; or none at all.

Configuring components can be done with the Component DSL, in a configuration file (application.properties|yaml), or directly with Java code.

72.2.2. Configuring Endpoint Options

Where you find yourself configuring the most is on endpoints, as endpoints often have many options, which allows you to configure what you need the endpoint to do. The options are also categorized into whether the endpoint is used as consumer (from) or as a producer (to), or used for both.

Configuring endpoints is most often done directly in the endpoint URI as path and query parameters. You can also use the Endpoint DSL as a type safe way of configuring endpoints.

A good practice when configuring options is to use Property Placeholders, which allows to not hardcode urls, port numbers, sensitive information, and other settings. In other words placeholders allows to externalize the configuration from your code, and gives more flexibility and reuse.

The following two sections lists all the options, firstly for the component followed by the endpoint.

72.3. COMPONENT OPTIONS

The XSLT component supports 7 options, which are listed below.

Name	Description	Default	Type
contentCache (producer)	Cache for the resource content (the stylesheet file) when it is loaded. If set to false Camel will reload the stylesheet file on each message processing. This is good for development. A cached stylesheet can be forced to reload at runtime via JMX using the clearCachedStylesheet operation.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
autowiredEnabled (advanced)	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	boolean
transformerFactoryClass (advanced)	To use a custom XSLT transformer factory, specified as a FQN class name.		String

Name	Description	Default	Type
transformerFactoryConfigurationStrategy (advanced)	A configuration strategy to apply on freshly created instances of TransformerFactory.		TransformerFactoryConfigurationStrategy
uriResolver (advanced)	To use a custom UriResolver. Should not be used together with the option 'uriResolverFactory'.		URIResolver
uriResolverFactory (advanced)	To use a custom UriResolver which depends on a dynamic endpoint resource URI. Should not be used together with the option 'uriResolver'.		XsltUriResolverFactory

72.4. ENDPOINT OPTIONS

The XSLT endpoint is configured using URI syntax:

```
xslt:resourceUri
```

with the following path and query parameters:

72.4.1. Path Parameters (1 parameters)

Name	Description	Default	Type
resourceUri (producer)	**Required** Path to the template. The following is supported by the default URIResolver. You can prefix with: classpath, file, http, ref, or bean. classpath, file and http loads the resource using these protocols (classpath is default). ref will lookup the resource in the registry. bean will call a method on a bean to be used as the resource. For bean you can specify the method name after dot, eg bean:myBean.myMethod.		String

72.4.2. Query Parameters (13 parameters)

Name	Description	Default	Type
contentCache (producer)	Cache for the resource content (the stylesheet file) when it is loaded. If set to false Camel will reload the stylesheet file on each message processing. This is good for development. A cached stylesheet can be forced to reload at runtime via JMX using the clearCachedStylesheet operation.	true	boolean

Name	Description	Default	Type
deleteOutputFile (producer)	If you have output=file then this option dictates whether or not the output file should be deleted when the Exchange is done processing. For example suppose the output file is a temporary file, then it can be a good idea to delete it after use.	false	boolean
failOnNullBody (producer)	Whether or not to throw an exception if the input body is null.	true	boolean
lazyStartProducer (producer)	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	boolean
output (producer)	Option to specify which output type to use. Possible values are: string, bytes, DOM, file. The first three options are all in memory based, where as file is streamed directly to a java.io.File. For file you must specify the filename in the IN header with the key Exchange.XSLT_FILE_NAME which is also CamelXsltFileName. Also any paths leading to the filename must be created beforehand, otherwise an exception is thrown at runtime. Enum values: • string • bytes • DOM • file	string	XsltOutput
transformerCacheSize (producer)	The number of javax.xml.transform.Transformer object that are cached for reuse to avoid calls to Template.newTransformer().	0	int
entityResolver (advanced)	To use a custom org.xml.sax.EntityResolver with javax.xml.transform.sax.SAXSource.		EntityResolver

Name	Description	Default	Type
errorListener (advanced)	Allows to configure to use a custom javax.xml.transform.ErrorListener. Beware when doing this then the default error listener which captures any errors or fatal errors and store information on the Exchange as properties is not in use. So only use this option for special use-cases.		ErrorListener
resultHandlerFactory (advanced)	Allows you to use a custom org.apache.camel.builder.xml.ResultHandlerFactory which is capable of using custom org.apache.camel.builder.xml.ResultHandler types.		ResultHandlerFactory
transformerFactory (advanced)	To use a custom XSLT transformer factory.		TransformerFactory
transformerFactoryClass (advanced)	To use a custom XSLT transformer factory, specified as a FQN class name.		String
transformerFactoryConfigurationStrategy (advanced)	A configuration strategy to apply on freshly created instances of TransformerFactory.		TransformerFactoryConfigurationStrategy
uriResolver (advanced)	To use a custom javax.xml.transform.URIResolver.		URIResolver

72.5. USING XSLT ENDPOINTS

The following format is an example of using an XSLT template to formulate a response for a message for InOut message exchanges (where there is a **JMSReplyTo** header)

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl");
```

If you want to use InOnly and consume the message and send it to another destination you could use the following route:

```
from("activemq:My.Queue").
  to("xslt:com/acme/mytransform.xsl").
  to("activemq:Another.Queue");
```

72.6. GETTING USEABLE PARAMETERS INTO THE XSLT

By default, all headers are added as parameters which are then available in the XSLT.
To make the parameters useable, you will need to declare them.

```
<setHeader name="myParam"><constant>42</constant></setHeader>
<to uri="xslt:MyTransform.xsl"/>
```

The parameter also needs to be declared in the top level of the XSLT for it to be available:

```
<xsl: ..... >

  <xsl:param name="myParam"/>

   <xsl:template ...>
```

72.7. SPRING XML VERSIONS

To use the above examples in Spring XML you would use something like the following code:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="activemq:My.Queue"/>
    <to uri="xslt:org/apache/camel/spring/processor/example.xsl"/>
    <to uri="activemq:Another.Queue"/>
  </route>
</camelContext>
```

72.8. USING XSL:INCLUDE

Camel provides its own implementation of **URIResolver**. This allows Camel to load included files from the classpath.

For example the include file in the following code will be located relative to the starting endpoint.

```
<xsl:include href="staff_template.xsl"/>
```

This means that Camel will locate the file in the **classpath** as **org/apache/camel/component/xslt/staff_template.xsl**

You can use **classpath:** or **file:** to instruct Camel to look either in the classpath or file system. If you omit the prefix then Camel uses the prefix from the endpoint configuration. If no prefix is specified in the endpoint configuration, the default is **classpath:**.

You can also refer backwards in the include paths. In the following example, the xsl file will be resolved under **org/apache/camel/component**.

```
    <xsl:include href="../staff_other_template.xsl"/>
```

72.9. USING XSL:INCLUDE AND DEFAULT PREFIX

Camel will use the prefix from the endpoint configuration as the default prefix.

You can explicitly specify **file:** or **classpath:** loading. The two loading types can be mixed in a XSLT script, if necessary.

72.10. DYNAMIC STYLESHEETS

To provide a dynamic stylesheet at runtime you can define a dynamic URI. See How to use a dynamic URI in to() for more information.

72.11. ACCESSING WARNINGS, ERRORS AND FATALERRORS FROM XSLT ERRORLISTENER

Any warning/error or fatalError is stored on the current Exchange as a property with the keys **Exchange.XSLT_ERROR**, **Exchange.XSLT_FATAL_ERROR**, or **Exchange.XSLT_WARNING** which allows end users to get hold of any errors happening during transformation.

For example in the stylesheet below, we want to terminate if a staff has an empty dob field. And to include a custom error message using xsl:message.

```
<xsl:template match="/">
  <html>
    <body>
      <xsl:for-each select="staff/programmer">
        <p>Name: <xsl:value-of select="name"/><br />
          <xsl:if test="dob=''">
            <xsl:message terminate="yes">Error: DOB is an empty string!</xsl:message>
          </xsl:if>
        </p>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
```

The exception is stored on the Exchange as a warning with the key **Exchange.XSLT_WARNING.**

72.12. SPRING BOOT AUTO-CONFIGURATION

When using xslt with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-xslt-starter</artifactId>
</dependency>
```

The component supports 8 options, which are listed below.

Name	Description	Default	Type

Name	Description	Default	Type
camel.component.xslt.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.xslt.content-cache	Cache for the resource content (the stylesheet file) when it is loaded. If set to false Camel will reload the stylesheet file on each message processing. This is good for development. A cached stylesheet can be forced to reload at runtime via JMX using the clearCachedStylesheet operation.	true	Boolean
camel.component.xslt.enabled	Whether to enable auto configuration of the xslt component. This is enabled by default.		Boolean
camel.component.xslt.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.xslt.transformer-factory-class	To use a custom XSLT transformer factory, specified as a FQN class name.		String
camel.component.xslt.transformer-factory-configuration-strategy	A configuration strategy to apply on freshly created instances of TransformerFactory. The option is a org.apache.camel.component.xslt.TransformerFactoryConfigurationStrategy type.		TransformerFactoryConfigurationStrategy
camel.component.xslt.uri-resolver	To use a custom UriResolver. Should not be used together with the option 'uriResolverFactory'. The option is a javax.xml.transform.URIResolver type.		URIResolver
camel.component.xslt.uri-resolver-factory	To use a custom UriResolver which depends on a dynamic endpoint resource URI. Should not be used together with the option 'uriResolver'. The option is a org.apache.camel.component.xslt.XsltUriResolverFactory type.		XsltUriResolverFactory

CHAPTER 73. AVRO

This component provides a dataformat for avro, which allows serialization and deserialization of messages using Apache Avro's binary dataformat. Since Camel 3.2 rpc functionality was moved into separate **camel-avro-rpc** component.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-avro</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

You can easily generate classes from a schema, using maven, ant etc. More details can be found at the Apache Avro documentation.

73.1. AVRO DATAFORMAT OPTIONS

The Avro dataformat supports 1 options, which are listed below.

Name	Default	Java Type	Description
instanceClassName		**String**	Class name to use for marshal and unmarshalling.

73.2. AVRO DATA FORMAT USAGE

Using the avro data format is as easy as specifying that the class that you want to marshal or unmarshal in your route.

```
AvroDataFormat format = new AvroDataFormat(Value.SCHEMA$);

from("direct:in").marshal(format).to("direct:marshal");
from("direct:back").unmarshal(format).to("direct:unmarshal");
```

Where Value is an Avro Maven Plugin Generated class.

or in XML

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:in"/>
    <marshal>
      <avro instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </marshal>
    <to uri="log:out"/>
  </route>
</camelContext>
```

An alternative can be to specify the dataformat inside the context and reference it from your route.

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <dataFormats>
      <avro id="avro" instanceClass="org.apache.camel.dataformat.avro.Message"/>
    </dataFormats>
    <route>
      <from uri="direct:in"/>
      <marshal><custom ref="avro"/></marshal>
      <to uri="log:out"/>
    </route>
</camelContext>
```

In the same manner you can umarshal using the avro data format.

73.3. SPRING BOOT AUTO-CONFIGURATION

When using avro with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

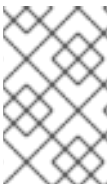
```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-avro-starter</artifactId>
</dependency>
```

The component supports 2 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.avro.enabled	Whether to enable auto configuration of the avro data format. This is enabled by default.		Boolean
camel.dataformat.avro.instance-class-name	Class name to use for marshal and unmarshalling.		String

CHAPTER 74. AVRO JACKSON

Jackson Avro is a Data Format which uses the Jackson library with the Avro extension to unmarshal an Avro payload into Java objects or to marshal Java objects into an Avro payload.



NOTE

If you are familiar with Jackson, this Avro data format behaves in the same way as its JSON counterpart, and thus can be used with classes annotated for JSON serialization/deserialization.

```
from("kafka:topic").
  unmarshal().avro(AvroLibrary.Jackson, JsonNode.class).
  to("log:info");
```

74.1. CONFIGURING THE SCHEMARESOLVER

Since Avro serialization is schema-based, this data format requires that you provide a SchemaResolver object that is able to lookup the schema for each exchange that is going to be marshalled/unmarshalled.

You can add a single SchemaResolver to the registry and it will be looked up automatically. Or you can explicitly specify the reference to a custom SchemaResolver.

74.2. AVRO JACKSON OPTIONS

The Avro Jackson dataformat supports 18 options, which are listed below.

Name	Default	Java Type	Description
objectMapper		**String**	Lookup and use the existing ObjectMapper with the given id when using Jackson.
useDefaultObjectMapper		**Boolean**	Whether to lookup and use default Jackson ObjectMapper from the registry.
unmarshalType		**String**	Class name of the java type to use when unmarshalling.
jsonView		**String**	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.
include		**String**	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.
allowJmsType		**Boolean**	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.

Name	Default	Java Type	Description
collectionType		String	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.
useList		Boolean	To unmarshal to a List of Map or a List of Pojo.
moduleClassNames		String	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.
moduleRefs		String	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.
enableFeatures		String	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
disableFeatures		String	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
allowUnmarshallType		Boolean	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.
timezone		String	If set then Jackson will use the Timezone when marshalling/unmarshalling.
autoDiscoverObjectMapper		Boolean	If set to true then Jackson will lookup for an objectMapper into the registry.
contentTypeHeader		Boolean	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.

Name	Default	Java Type	Description
schemaResolver		String	Optional schema resolver used to lookup schemas for the data in transit.
autoDiscoverSchemaResolver		Boolean	When not disabled, the SchemaResolver will be looked up into the registry.

74.3. USING CUSTOM AVROMAPPER

You can configure **JacksonAvroDataFormat** to use a custom **AvroMapper** in case you need more control of the mapping configuration.

If you setup a single **AvroMapper** in the registry, then Camel will automatic lookup and use this **AvroMapper**.

74.4. DEPENDENCIES

To use Avro Jackson in your camel routes you need to add the dependency on **camel-jackson-avro** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson-avro</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

74.5. SPRING BOOT AUTO-CONFIGURATION

When using avro-jackson with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jackson-avro-starter</artifactId>
</dependency>
```

The component supports 19 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.avro-jackson.allow-jms-type	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.	false	Boolean
camel.dataformat.avro-jackson.allow-unmarshall-type	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.	false	Boolean
camel.dataformat.avro-jackson.auto-discover-object-mapper	If set to true then Jackson will lookup for an objectMapper into the registry.	false	Boolean
camel.dataformat.avro-jackson.auto-discover-schema-resolver	When not disabled, the SchemaResolver will be looked up into the registry.	true	Boolean
camel.dataformat.avro-jackson.collection-type	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.		String
camel.dataformat.avro-jackson.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.avro-jackson.disable-features	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.avro-jackson.enable-features	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String

Name	Description	Default	Type
camel.dataformat.avro-jackson.enabled	Whether to enable auto configuration of the avro-jackson data format. This is enabled by default.		Boolean
camel.dataformat.avro-jackson.include	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.		String
camel.dataformat.avro-jackson.json-view	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.		String
camel.dataformat.avro-jackson.module-class-names	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.		String
camel.dataformat.avro-jackson.module-refs	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.		String
camel.dataformat.avro-jackson.object-mapper	Lookup and use the existing ObjectMapper with the given id when using Jackson.		String
camel.dataformat.avro-jackson.schema-resolver	Optional schema resolver used to lookup schemas for the data in transit.		String
camel.dataformat.avro-jackson.timezone	If set then Jackson will use the Timezone when marshalling/unmarshalling.		String
camel.dataformat.avro-jackson.unmarshal-type	Class name of the java type to use when unmarshalling.		String

Name	Description	Default	Type
camel.dataformat.avro-jackson.use-default-object-mapper	Whether to lookup and use default Jackson ObjectMapper from the registry.	true	Boolean
camel.dataformat.avro-jackson.use-list	To unmarshal to a List of Map or a List of Pojo.	false	Boolean

CHAPTER 75. BINDY

The goal of this component is to allow the parsing/binding of non-structured data (or to be more precise non-XML data) to/from Java Beans that have binding mappings defined with annotations. Using Bindy, you can bind data from sources such as :

- CSV records,
- Fixed-length records,
- FIX messages,
- or almost any other non-structured data

to one or many Plain Old Java Object (POJO). Bindy converts the data according to the type of the java property. POJOs can be linked together with one-to-many relationships available in some cases. Moreover, for data type like Date, Double, Float, Integer, Short, Long and BigDecimal, you can provide the pattern to apply during the formatting of the property.

For the BigDecimal numbers, you can also define the precision and the decimal or grouping separators.

Type	Format Type	Pattern example	Link
Date	**DateFormat**	**dd-MM-yyyy**	https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/SimpleDateFormat.html
Decimal*	**DecimalFormat**	**..##**	https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/DecimalFormat.html

Where Decimal = Double, Integer, Float, Short, Long

Format supported

This first release only support comma separated values fields and key value pair fields (e.g. : FIX messages).

To work with camel-bindy, you must first define your model in a package (e.g. com.acme.model) and for each model class (e.g. Order, Client, Instrument, ...) add the required annotations (described hereafter) to the Class or field.

Multiple models

As you configure bindy using class names instead of package names you can put multiple models in the same package.

75.1. OPTIONS

The Bindy dataformat supports 5 options, which are listed below.

Name	Default	Java Type	Description
type		**Enum**	**Required** Whether to use Csv, Fixed, or KeyValue. Enum values: • Csv • Fixed • KeyValue
classType		**String**	Name of model class to use.
locale		**String**	To configure a default locale to use, such as us for united states. To use the JVM platform default locale then use the name default.
unwrapSingleInstance		**Boolean**	When unmarshalling should a single instance be unwrapped and returned instead of wrapped in a java.util.List.
allowEmptyStream		**Boolean**	Whether to allow empty streams in the unmarshal process. If true, no exception will be thrown when a body without records is provided.

75.2. ANNOTATIONS

The annotations created allow to map different concept of your model to the POJO like:

- Type of record (CSV, key value pair (e.g. FIX message), fixed length ...),
- Link (to link object in another object),
- DataField and their properties (int, type, ...),
- KeyValuePairField (for key = value format like we have in FIX financial messages),
- Section (to identify header, body and footer section),
- OneToMany,
- BindyConverter,
- FormatFactories

This section will describe them.

75.2.1. 1. CsvRecord

The CsvRecord annotation is used to identified the root class of the model. It represents a record = "a line of a CSV file" and can be linked to several children model classes.

Annotation name	Record type	Level
CsvRecord	CSV	Class

Parameter name	Type	Required	Default value	Info
separator	String	✓		Separator used to split a record in tokens (mandatory) - can be ',' or ';' or 'anything'. The only whitespace character supported is tab (\t). No other whitespace characters (spaces) are not supported. This value is interpreted as a regular expression. If you want to use a sign which has a special meaning in regular expressions, e.g. the '|' sign, then you have to mask it, like '|'.
allowEmptyStream	boolean		false	The allowEmptyStream parameter will allow to prcoess the unavaiable stream for CSV file.
autospanLine	boolean		false	Last record spans rest of line (optional) - if enabled then the last column is auto spanned to end of line, for example if its a comment, etc this allows the line to contain all characters, also the delimiter char.
crlf	String		WINDOWS	Character to be used to add a carriage return after each record (optional) - allow to define the carriage return character to use. If you specify a value other than the three listed before, the value you enter (custom) will be used as the CRLF character(s). Three values can be used : WINDOWS, UNIX, MAC, or custom.
endWithLineBreak	boolean		true	The endWithLineBreak parameter flags if the CSV file should end with a line break or not (optional)
generateHeaderColumns	boolean		false	The generateHeaderColumns parameter allow to add in the CSV generated the header containing names of the columns
isOrdered	boolean		false	Indicates if the message must be ordered in output
name	String			Name describing the record (optional)

Parameter name	Type	Required	Default value	Info
quote	String		"	Whether to marshal columns with the given quote character (optional) - allow to specify a quote character of the fields when CSV is generated. This annotation is associated to the root class of the model and must be declared one time.
quoting	boolean		false	Indicate if the values (and headers) must be quoted when marshaling (optional)
quotingEscaped	boolean		false	Indicate if the values must be escaped when quoting (optional)
removeQuotes	boolean		true	The remove quotes parameter flags if unmarshalling should try to remove quotes for each field
skipField	boolean		false	The skipField parameter will allow to skip fields of a CSV file. If some fields are not necessary, they can be skipped.
skipFirstLine	boolean		false	The skipFirstLine parameter will allow to skip or not the first line of a CSV file. This line often contains columns definition

case 1 : separator = ','

The separator used to segregate the fields in the CSV record is **,** :

```
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500, USD, 08-01-2009
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

case 2 : separator = ';'

Compare to the previous case, the separator here is **;** instead of **,** :

```
10; J; Pauline; M; XD12345678; Fortis Dynamic 15/15; 2500; USD; 08-01-2009
```

```
@CsvRecord( separator = ";" )
public Class Order {

}
```

case 3 : separator = '|'

Compare to the previous case, the separator here is | instead of ; :

```
10| J| Pauline| M| XD12345678| Fortis Dynamic 15/15| 2500| USD| 08-01-2009
```

```
@CsvRecord( separator = "\\|" )
public Class Order {

}
```

case 4 : separator = '\",\"'

Applies for Camel 2.8.2 or older

When the field to be parsed of the CSV record contains , or ; which is also used as separator, we should find another strategy to tell camel bindy how to handle this case. To define the field containing the data with a comma, you will use single or double quotes as delimiter (e.g : '10', 'Street 10, NY', 'USA' or "10", "Street 10, NY", "USA").

—	In this case, the first and last character of the line which are a single or double quotes will be removed by bindy.

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15","2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "\",\"" )
public Class Order {

}
```

Bindy automatically detects if the record is enclosed with either single or double quotes and automatic remove those quotes when unmarshalling from CSV to Object. Therefore do **not** include the quotes in the separator, but simply do as below:

```
"10","J","Pauline"," M","XD12345678","Fortis Dynamic 15,15","2500","USD","08-01-2009"
```

```
@CsvRecord( separator = "," )
public Class Order {

}
```

Notice that if you want to marshal from Object to CSV and use quotes, then you need to specify which quote character to use, using the **quote** attribute on the **@CsvRecord** as shown below:

```
@CsvRecord( separator = ",", quote = "\"" )
public Class Order {

}
```

case 5 : separator & skipFirstLine

The feature is interesting when the client wants to have in the first line of the file, the name of the data fields :

```
order id, client id, first name, last name, isin code, instrument name, quantity, currency, date
```

To inform bindy that this first line must be skipped during the parsing process, then we use the attribute :

```
@CsvRecord(separator = ",", skipFirstLine = true)
public Class Order {

}
```

case 6 : generateHeaderColumns

To add at the first line of the CSV generated, the attribute generateHeaderColumns must be set to true in the annotation like this :

```
@CsvRecord( generateHeaderColumns = true )
public Class Order {

}
```

As a result, Bindy during the unmarshaling process will generate CSV like this :

```
order id, client id, first name, last name, isin code, instrument name, quantity, currency, date
10, J, Pauline, M, XD12345678, Fortis Dynamic 15/15, 2500, USD, 08-01-2009
```

case 7 : carriage return

If the platform where camel-bindy will run is not Windows but Macintosh or Unix, then you can change the crlf property like this. Three values are available : WINDOWS, UNIX or MAC

```
@CsvRecord(separator = ",", crlf="MAC")
public Class Order {

}
```

Additionally, if for some reason you need to add a different line ending character, you can opt to specify it using the crlf parameter. In the following example, we can end the line with a comma followed by the newline character:

```
@CsvRecord(separator = ",", crlf=",\n")
public Class Order {

}
```

case 8 : isOrdered

Sometimes, the order to follow during the creation of the CSV record from the model is different from the order used during the parsing. Then, in this case, we can use the attribute **isOrdered = true** to indicate this in combination with attribute **position** of the DataField annotation.

```
@CsvRecord(isOrdered = true)
```

```
public Class Order {

   @DataField(pos = 1, position = 11)
   private int orderNr;

   @DataField(pos = 2, position = 10)
   private String clientNr;

}
```

—	**pos** is used to parse the file stream, while**position** is used to generate the CSV.

75.2.2. 2. Link

The link annotation will allow to link objects together.

Annotation name	Record type	Level
Link	all	Class & Property

Parameter name	Type	Required	Default value	Info
linkType	LinkType		OneToOne	Type of link identifying the relation between the classes

Only one-to-one relation is allowed as of the current version.

E.g : If the model class Client is linked to the Order class, then use annotation Link in the Order class like this :

Property Link

```
@CsvRecord(separator = ",")
public class Order {

   @DataField(pos = 1)
   private int orderNr;

   @Link
   private Client client;
}
```

And for the class Client :

Class Link

```
@Link
public class Client {

}
```

75.2.3. 3. DataField

The DataField annotation defines the property of the field. Each datafield is identified by its position in the record, a type (string, int, date, ...) and optionally of a pattern.

Annotation name	Record type	Level
DataField	all	Property

Parameter name	Type	Required	Default value	Info
pos	int	✓		Position of the data in the input record, must start from 1 (mandatory). See the position parameter.
align	String		R	Align the text to the right or left. Use values <tt>R</tt> or <tt>L</tt>.
clip	boolean		false	Indicates to clip data in the field if it exceeds the allowed length when using fixed length.
columnName	String			Name of the header column (optional). Uses the name of the property as default. Only applicable when **CsvRecord** has **generateHeaderColumns = true**
decimalSeparator	String			Decimal Separator to be used with BigDecimal number
defaultValue	String			Field's default value in case no value is set
delimiter	String			Optional delimiter to be used if the field has a variable length
groupingSeparator	String			Grouping Separator to be used with BigDecimal number when we would like to format/parse to number with grouping e.g. 123,456.789

Parameter name	Type	Required	Default value	Info
impliedDecimalSeparator	boolean		false	Indicates if there is a decimal point implied at a specified location
length	int		0	Length of the data block (number of characters) if the record is set to a fixed length
lengthPos	int		0	Identifies a data field in the record that defines the expected fixed length for this field
method	String			Method name to call to apply such customization on DataField. This must be the method on the datafield itself or you must provide static fully qualified name of the class's method e.g: see unit test org.apache.camel.dataformat.bindy.csv.BindySimpleCsvFunctionWithExternalMethodTest.replaceToBar
name	String			Name of the field (optional)
paddingChar	char			The char to pad with if the record is set to a fixed length
pattern	String			Pattern that the Java formatter (SimpleDateFormat by example) will use to transform the data (optional). If using pattern, then setting locale on bindy data format is recommended. Either set to a known locale such as "us" or use "default" to use platform default locale.
position	int		0	Position of the field in the output message generated (should start from 1). Must be used when the position of the field in the CSV generated (output message) must be different compare to input position (pos). See the pos parameter.
precision	int		0	precision of the \{@link java.math.BigDecimal} number to be created
required	boolean		false	Indicates if the field is mandatory
rounding	String		CEILING	Round mode to be used to round/scale a BigDecimal Values : UP, DOWN, CEILING, FLOOR, HALF_UP, HALF_DOWN,HALF_EVEN, UNNECESSARY e.g : Number = 123456.789, Precision = 2, Rounding = CEILING Result : 123456.79

Parameter name	Type	Required	Default value	Info
timezone	String			Timezone to be used.
trim	boolean		false	Indicates if the value should be trimmed

case 1 : pos

This parameter/attribute represents the position of the field in the CSV record.

Position

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5)
    private String isinCode;

}
```

As you can see in this example the position starts at **1** but continues at **5** in the class Order. The numbers from **2** to **4** are defined in the class Client (see here after).

Position continues in another model class

```
public class Client {

    @DataField(pos = 2)
    private String clientNr;

    @DataField(pos = 3)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

case 2 : pattern

The pattern allows to enrich or validates the format of your data

Pattern

```
@CsvRecord(separator = ",")
public class Order {
```

```
    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    // pattern used during parsing or when the date is created
    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 3 : precision

The precision is helpful when you want to define the decimal part of your number.

Precision

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @Link
    private Client client;

    @DataField(pos = 5)
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;

    @DataField(pos = 7, precision = 2)
    private BigDecimal amount;

    @DataField(pos = 8)
    private String currency;

    @DataField(pos = 9, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 4 : Position is different in output

The position attribute will inform bindy how to place the field in the CSV record generated. By default, the position used corresponds to the position defined with the attribute **pos**. If the position is different (that means that we have an asymetric processus comparing marshaling from unmarshaling) then we

can use **position** to indicate this.

Here is an example:

Position is different in output

```
@CsvRecord(separator = ",", isOrdered = true)
public class Order {

    // Positions of the fields start from 1 and not from 0

    @DataField(pos = 1, position = 11)
    private int orderNr;

    @DataField(pos = 2, position = 10)
    private String clientNr;

    @DataField(pos = 3, position = 9)
    private String firstName;

    @DataField(pos = 4, position = 8)
    private String lastName;

    @DataField(pos = 5, position = 7)
    private String instrumentCode;

    @DataField(pos = 6, position = 6)
    private String instrumentNumber;
}
```

This attribute of the annotation **@DataField** must be used in combination with attribute **isOrdered = true** of the annotation **@CsvRecord**.

case 5 : required

If a field is mandatory, simply use the attribute **required** set to true.

Required

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2, required = true)
    private String clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, required = true)
    private String lastName;
}
```

If this field is not present in the record, then an error will be raised by the parser with the following information :

```
Some fields are missing (optional or mandatory), line :
```

case 6 : trim

If a field has leading and/or trailing spaces which should be removed before they are processed, simply use the attribute **trim** set to true.

Trim

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1, trim = true)
    private int orderNr;

    @DataField(pos = 2, trim = true)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4)
    private String lastName;
}
```

case 7 : defaultValue

If a field is not defined then uses the value indicated by the **defaultValue** attribute.

Default value

```
@CsvRecord(separator = ",")
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 2)
    private Integer clientNr;

    @DataField(pos = 3, required = true)
    private String firstName;

    @DataField(pos = 4, defaultValue = "Barin")
    private String lastName;
}
```

case 8 : columnName

Specifies the column name for the property only if **@CsvRecord** has annotation **generateHeaderColumns = true**.

Column Name

```
@CsvRecord(separator = ",", generateHeaderColumns = true)
public class Order {

    @DataField(pos = 1)
    private int orderNr;

    @DataField(pos = 5, columnName = "ISIN")
    private String isinCode;

    @DataField(name = "Name", pos = 6)
    private String instrumentName;
}
```

This attribute is only applicable to optional fields.

75.2.4. 4. FixedLengthRecord

The FixedLengthRecord annotation is used to identified the root class of the model. It represents a record = "a line of a file/message containing data fixed length (number of characters) formatted" and can be linked to several children model classes. This format is a bit particular because data of a field can be aligned to the right or to the left.

When the size of the data does not fill completely the length of the field, we can then add 'pad' characters.

Annotation name	Record type	Level
FixedLengthRecord	fixed	Class

Parameter name	Type	Required	Default value	Info
countGrapheme	boolean		false	Indicates how chars are counted
crlf	String		WINDOWS	Character to be used to add a carriage return after each record (optional). Possible values: WINDOWS, UNIX, MAC, or custom. This option is used only during marshalling, whereas unmarshalling uses system default JDK provided line delimiter unless eol is customized.

Parameter name	Type	Required	Default value	Info
eol	String			Character to be used to process considering end of line after each record while unmarshalling (optional - default: "", which help default JDK provided line delimiter to be used unless any other line delimiter provided) This option is used only during unmarshalling, where marshalling uses system default provided line delimiter as "WINDOWS" unless any other value is provided.
footer	Class		void	Indicates that the record(s) of this type may be followed by a single footer record at the end of the file
header	Class		void	Indicates that the record(s) of this type may be preceded by a single header record at the beginning of in the file
ignoreMissingChars	boolean		false	Indicates whether too short lines will be ignored
ignoreTrailingChars	boolean		false	Indicates that characters beyond the last mapped filed can be ignored when unmarshalling / parsing. This annotation is associated to the root class of the model and must be declared one time.
length	int		0	The fixed length of the record (number of characters). It means that the record will always be that long padded with \ {#paddingChar()}'s
name	String			Name describing the record (optional)
paddingChar	char			The char to pad with.
skipFooter	boolean		false	Configures the data format to skip marshalling / unmarshalling of the footer record. Configure this parameter on the primary record (e.g., not the header or footer).
skipHeader	boolean		false	Configures the data format to skip marshalling / unmarshalling of the header record. Configure this parameter on the primary record (e.g., not the header or footer).

A record may not be both a header/footer and a primary fixed-length record.

case 1 : Simple fixed length record

This simple example shows how to design the model to parse/format a fixed message

```
10A9PaulineMISINXD12345678BUYShare2500.45USD01-08-2009
```

Fixed-simple

```
@FixedLengthRecord(length=54, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;

    @DataField(pos = 3, length=2)
    private String clientNr;

    @DataField(pos = 5, length=7)
    private String firstName;

    @DataField(pos = 12, length=1, align="L")
    private String lastName;

    @DataField(pos = 13, length=4)
    private String instrumentCode;

    @DataField(pos = 17, length=10)
    private String instrumentNumber;

    @DataField(pos = 27, length=3)
    private String orderType;

    @DataField(pos = 30, length=5)
    private String instrumentType;

    @DataField(pos = 35, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 42, length=3)
    private String currency;

    @DataField(pos = 45, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 2 : Fixed length record with alignment and padding

This more elaborated example show how to define the alignment for a field and how to assign a padding character which is **' '** here:

```
10A9 PaulineM ISINXD12345678BUYShare2500.45USD01-08-2009
```

Fixed-padding-align

```
@FixedLengthRecord(length=60, paddingChar=' ')
public static class Order {

    @DataField(pos = 1, length=2)
    private int orderNr;
```

```
    @DataField(pos = 3, length=2)
    private String clientNr;

    @DataField(pos = 5, length=9)
    private String firstName;

    @DataField(pos = 14, length=5, align="L")   // align text to the LEFT zone of the block
    private String lastName;

    @DataField(pos = 19, length=4)
    private String instrumentCode;

    @DataField(pos = 23, length=10)
    private String instrumentNumber;

    @DataField(pos = 33, length=3)
    private String orderType;

    @DataField(pos = 36, length=5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length=7)
    private BigDecimal amount;

    @DataField(pos = 48, length=3)
    private String currency;

    @DataField(pos = 51, length=10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 3 : Field padding

Sometimes, the default padding defined for record cannnot be applied to the field as we have a number format where we would like to pad with '0' instead of ' '. In this case, you can use in the model the attribute **paddingChar** on **@DataField** to set this value.

```
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-padding-field

```
@FixedLengthRecord(length = 65, paddingChar = ' ')
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    @DataField(pos = 5, length = 9)
    private String firstName;

    @DataField(pos = 14, length = 5, align = "L")
```

```
    private String lastName;

    @DataField(pos = 19, length = 4)
    private String instrumentCode;

    @DataField(pos = 23, length = 10)
    private String instrumentNumber;

    @DataField(pos = 33, length = 3)
    private String orderType;

    @DataField(pos = 36, length = 5)
    private String instrumentType;

    @DataField(pos = 41, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 53, length = 3)
    private String currency;

    @DataField(pos = 56, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 4: Fixed length record with delimiter

Fixed-length records sometimes have delimited content within the record. The firstName and lastName fields are delimited with the **^** character in the following example:

```
10A9Pauline^M^ISINXD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-delimited

```
@FixedLengthRecord
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
```

```
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;

    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;

    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

The **pos** value(s) in a fixed-length record may optionally be defined using ordinal, sequential values instead of precise column numbers.

case 5 : Fixed length record with record-defined field length

Occasionally a fixed-length record may contain a field that define the expected length of another field within the same record. In the following example the length of the **instrumentNumber** field value is defined by the value of **instrumentNumberLen** field in the record.

```
10A9Pauline^M^ISIN10XD12345678BUYShare000002500.45USD01-08-2009
```

Fixed-delimited

```
@FixedLengthRecord
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, delimiter = "^")
    private String firstName;

    @DataField(pos = 4, delimiter = "^")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 2, align = "R", paddingChar = '0')
    private int instrumentNumberLen;

    @DataField(pos = 7, lengthPos=6)
    private String instrumentNumber;

    @DataField(pos = 8, length = 3)
    private String orderType;
```

```
    @DataField(pos = 9, length = 5)
    private String instrumentType;

    @DataField(pos = 10, precision = 2, length = 12, paddingChar = '0')
    private BigDecimal amount;

    @DataField(pos = 11, length = 3)
    private String currency;

    @DataField(pos = 12, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}
```

case 6 : Fixed length record with header and footer

Bindy will discover fixed-length header and footer records that are configured as part of the model – provided that the annotated classes exist either in the same package as the primary **@FixedLengthRecord** class, or within one of the configured scan packages. The following text illustrates two fixed-length records that are bracketed by a header record and footer record.

```
101-08-2009
10A9 PaulineM ISINXD12345678BUYShare000002500.45USD01-08-2009
10A9 RichN ISINXD12345678BUYShare000002700.45USD01-08-2009
9000000002
```

Fixed-header-and-footer-main-class

```
@FixedLengthRecord(header = OrderHeader.class, footer = OrderFooter.class)
public class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 2, length = 2)
    private String clientNr;

    @DataField(pos = 3, length = 9)
    private String firstName;

    @DataField(pos = 4, length = 5, align = "L")
    private String lastName;

    @DataField(pos = 5, length = 4)
    private String instrumentCode;

    @DataField(pos = 6, length = 10)
    private String instrumentNumber;

    @DataField(pos = 7, length = 3)
    private String orderType;

    @DataField(pos = 8, length = 5)
    private String instrumentType;

    @DataField(pos = 9, precision = 2, length = 12, paddingChar = '0')
```

```
    private BigDecimal amount;

    @DataField(pos = 10, length = 3)
    private String currency;

    @DataField(pos = 11, length = 10, pattern = "dd-MM-yyyy")
    private Date orderDate;
}

@FixedLengthRecord
public  class OrderHeader {
    @DataField(pos = 1, length = 1)
    private int recordType = 1;

    @DataField(pos = 2, length = 10, pattern = "dd-MM-yyyy")
    private Date recordDate;
}

@FixedLengthRecord
public class OrderFooter {

    @DataField(pos = 1, length = 1)
    private int recordType = 9;

    @DataField(pos = 2, length = 9, align = "R", paddingChar = '0')
    private int numberOfRecordsInTheFile;
}
```

case 7 : Skipping content when parsing a fixed length record

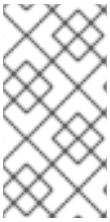
It is common to integrate with systems that provide fixed-length records containing more information than needed for the target use case. It is useful in this situation to skip the declaration and parsing of those fields that we do not need. To accomodate this, Bindy will skip forward to the next mapped field within a record if the **pos** value of the next declared field is beyond the cursor position of the last parsed field. Using absolute **pos** locations for the fields of interest (instead of ordinal values) causes Bindy to skip content between two fields.

Similarly, it is possible that none of the content beyond some field is of interest. In this case, you can tell Bindy to skip parsing of everything beyond the last mapped field by setting the **ignoreTrailingChars** property on the **@FixedLengthRecord** declaration.

```
@FixedLengthRecord(ignoreTrailingChars = true)
public static class Order {

    @DataField(pos = 1, length = 2)
    private int orderNr;

    @DataField(pos = 3, length = 2)
    private String clientNr;

    // any characters that appear beyond the last mapped field will be ignored

}
```

75.2.5. 5. Message

The Message annotation is used to identified the class of your model who will contain key value pairs fields. This kind of format is used mainly in Financial Exchange Protocol Messages (FIX). Nevertheless, this annotation can be used for any other format where data are identified by keys. The key pair values are separated each other by a separator which can be a special character like a tab delimitor (unicode representation : **u0009**) or a start of heading (unicode representation : **u0001**)



NOTE

To work with FIX messages, the model must contain a Header and Trailer classes linked to the root message class which could be a Order class. This is not mandatory but will be very helpful when you will use camel-bindy in combination with camel-fix which is a Fix gateway based on quickFix project .

Annotation name	Record type	Level
Message	key value pair	Class

Parameter name	Type	Required	Default value	Info
keyValuePairSeparator	String	✓		Key value pair separator is used to split the values from their keys (mandatory). Can be '\u0001', '\u0009', '#', or 'anything'.
pairSeparator	String	✓		Pair separator used to split the key value pairs in tokens (mandatory). Can be '=', ';', or 'anything'.
crlf	String		WINDOWS	Character to be used to add a carriage return after each record (optional). Possible values = WINDOWS, UNIX, MAC, or custom. If you specify a value other than the three listed before, the value you enter (custom) will be used as the CRLF character(s).
isOrdered	boolean		false	Indicates if the message must be ordered in output. This annotation is associated to the message class of the model and must be declared one time.
name	String			Name describing the message (optional)
type	String		FIX	type is used to define the type of the message (e.g. FIX, EMX, ...) (optional)
version	String		4.1	version defines the version of the message (e.g. 4.1, ...) (optional)

case 1 : separator = 'u0001'

The separator used to segregate the key value pair fields in a FIX message is the ASCII **01** character or in unicode format **\u0001**. This character must be escaped a second time to avoid a java runtime error. Here is an example :

```
8=FIX.4.1 9=20 34=1 35=0 49=INVMGR 56=BRKR 1=BE.CHM.001 11=CHM0001-01 22=4 ...
```

and how to use the annotation:

FIX - message

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

}
```

Look at test cases

The ASCII character like tab, ... cannot be displayed in WIKI page. So, have a look to the test case of camel-bindy to see exactly how the FIX message looks like (https://github.com/apache/camel/blob/main/components/camel-bindy/src/test/data/fix/fix.txt) and the Order, Trailer, Header classes (https://github.com/apache/camel/blob/main/components/camel-bindy/src/test/java/org/apache/camel/dataformat/bindy/model/fix/simple/Order.java).

75.2.6. 6. KeyValuePairField

The KeyValuePairField annotation defines the property of a key value pair field. Each KeyValuePairField is identified by a tag (= key) and its value associated, a type (string, int, date, ...), optionaly a pattern and if the field is required.

Annotation name	Record type	Level
KeyValuePairField	Key Value Pair - FIX	Property

Parameter name	Type	Required	Default value	Info
tag	int	✓		tag identifying the field in the message (mandatory) - must be unique
impliedDecimalSeparator	boolean		false	Camel 2.11: Indicates if there is a decimal point implied at a specified location
name	String			name of the field (optional)
pattern	String			pattern that the formater will use to transform the data (optional)

Parameter name	Type	Required	Default value	Info
position	int		0	Position of the field in the message generated - must be used when the position of the key/tag in the FIX message must be different
precision	int		0	precision of the BigDecimal number to be created
required	boolean		false	Indicates if the field is mandatory
timezone	String			Timezone to be used.

case 1 : tag

This parameter represents the key of the field in the message:

FIX message - Tag

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\u0001", type="FIX", version="4.1")
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String Account;

    @KeyValuePairField(tag = 11) // Order reference
    private String ClOrdId;

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String IDSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String SecurityId;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String Side;

    @KeyValuePairField(tag = 58) // Free text
    private String Text;
}
```

case 2 : Different position in output

If the tags/keys that we will put in the FIX message must be sorted according to a predefine order, then use the attribute **position** of the annotation **@KeyValuePairField**.

FIX message - Tag - sort

```
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version = "4.1",
isOrdered = true)
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
}
```

75.2.7. 7. Section

In FIX message of fixed length records, it is common to have different sections in the representation of the information : header, body and section. The purpose of the annotation **@Section** is to inform bindy about which class of the model represents the header (= section 1), body (= section 2) and footer (= section 3)

Only one attribute/parameter exists for this annotation.

Annotation name	Record type	Level
Section	FIX	Class

Parameter name	Type	Required	Default value	Info
number	int	✓		Number of the section

case 1 : Section

Definition of the header section:

FIX message - Section - Header

```
@Section(number = 1)
public class Header {

    @KeyValuePairField(tag = 8, position = 1) // Message Header
    private String beginString;
```

```
    @KeyValuePairField(tag = 9, position = 2) // Checksum
    private int bodyLength;
}
```

Definition of the body section:

FIX message - Section - Body

```
@Section(number = 2)
@Message(keyValuePairSeparator = "=", pairSeparator = "\\u0001", type = "FIX", version = "4.1",
isOrdered = true)
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1, position = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11, position = 3) // Order reference
    private String clOrdId;
```

Definition of the footer section:

FIX message - Section - Footer

```
@Section(number = 3)
public class Trailer {

    @KeyValuePairField(tag = 10, position = 1)
    // CheckSum
    private int checkSum;

    public int getCheckSum() {
        return checkSum;
    }
```

75.2.8. 8. OneToMany

The purpose of the annotation **@OneToMany** is to allow to work with a **List<?>** field defined a POJO class or from a record containing repetitive groups.

NOTE

Restrictions for OneToMany
Be careful, the one to many of bindy does not allow to handle repetitions defined on several levels of the hierarchy.

The relation OneToMany ONLY WORKS in the following cases :

- Reading a FIX message containing repetitive groups (= group of tags/keys)
- Generating a CSV with repetitive data

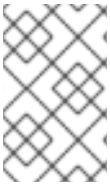
Annotation name	Record type	Level
OneToMany	all	Property

Parameter name	Type	Required	Default value	Info
mappedTo	String			Class name associated to the type of the List<Type of the Class>

case 1 : Generating CSV with repetitive data

Here is the CSV output that we want :

```
Claus,Ibsen,Camel in Action 1,2010,35
Claus,Ibsen,Camel in Action 2,2012,35
Claus,Ibsen,Camel in Action 3,2013,35
Claus,Ibsen,Camel in Action 4,2014,35
```



NOTE

The repetitive data concern the title of the book and its publication date while first, last name and age are common and the classes used to modeling this. The Author class contains a List of Book.

Generate CSV with repetitive data

```
@CsvRecord(separator=",")
public class Author {

    @DataField(pos = 1)
    private String firstName;

    @DataField(pos = 2)
    private String lastName;

    @OneToMany
    private List<Book> books;

    @DataField(pos = 5)
    private String Age;
}

public class Book {

    @DataField(pos = 3)
    private String title;
```

```
    @DataField(pos = 4)
    private String year;
}
```

case 2 : Reading FIX message containing group of tags/keys

Here is the message that we would like to process in our model :

```
8=FIX 4.19=2034=135=049=INVMGR56=BRKR
1=BE.CHM.00111=CHM0001-0158=this is a camel - bindy test
22=448=BE000124567854=1
22=548=BE000987654354=2
22=648=BE000999999954=3
10=220
```

Tags 22, 48 and 54 are repeated.

And the code:

Reading FIX message containing group of tags/keys

```
public class Order {

    @Link Header header;

    @Link Trailer trailer;

    @KeyValuePairField(tag = 1) // Client reference
    private String account;

    @KeyValuePairField(tag = 11) // Order reference
    private String clOrdId;

    @KeyValuePairField(tag = 58) // Free text
    private String text;

    @OneToMany(mappedTo =
"org.apache.camel.dataformat.bindy.model.fix.complex.onetomany.Security")
    List<Security> securities;
}

public class Security {

    @KeyValuePairField(tag = 22) // Fund ID type (Sedol, ISIN, ...)
    private String idSource;

    @KeyValuePairField(tag = 48) // Fund code
    private String securityCode;

    @KeyValuePairField(tag = 54) // Movement type ( 1 = Buy, 2 = sell)
    private String side;
}
```

75.2.9. 9. BindyConverter

The purpose of the annotation **@BindyConverter** is define a converter to be used on field level. The provided class must implement the Format interface.

```
@FixedLengthRecord(length = 10, paddingChar = ' ')
public static class DataModel {
    @DataField(pos =  1, length = 10, trim = true)
    @BindyConverter(CustomConverter.class)
    public String field1;
}

public static class CustomConverter implements Format<String> {
    @Override
    public String format(String object) throws Exception {
        return (new StringBuilder(object)).reverse().toString();
    }

    @Override
    public String parse(String string) throws Exception {
        return (new StringBuilder(string)).reverse().toString();
    }
}
```

75.2.10. 10. FormatFactories

The purpose of the annotation **@FormatFactories** is to define a set of converters at record-level. The provided classes must implement the **FormatFactoryInterface** interface.

```
@CsvRecord(separator = ",")
@FormatFactories({OrderNumberFormatFactory.class})
public static class Order {

    @DataField(pos = 1)
    private OrderNumber orderNr;

    @DataField(pos = 2)
    private String firstName;
}

public static class OrderNumber {
    private int orderNr;

    public static OrderNumber ofString(String orderNumber) {
        OrderNumber result = new OrderNumber();
        result.orderNr = Integer.valueOf(orderNumber);
        return result;
    }
}

public static class OrderNumberFormatFactory extends AbstractFormatFactory {

    {
        supportedClasses.add(OrderNumber.class);
    }

    @Override
```

```
    public Format<?> build(FormattingOptions formattingOptions) {
        return new Format<OrderNumber>() {
            @Override
            public String format(OrderNumber object) throws Exception {
                return String.valueOf(object.orderNr);
            }

            @Override
            public OrderNumber parse(String string) throws Exception {
                return OrderNumber.ofString(string);
            }
        };
    }
}
```

75.3. SUPPORTED DATATYPES

The DefaultFormatFactory makes formatting of the following datatype available by returning an instance of the interface FormatFactoryInterface based on the provided FormattingOptions:

- BigDecimal
- BigInteger
- Boolean
- Byte
- Character
- Date
- Double
- Enums
- Float
- Integer
- LocalDate
- LocalDateTime
- LocalTime
- Long
- Short
- String

The DefaultFormatFactory can be overridden by providing an instance of FactoryRegistry in the registry in use (e.g. spring or JNDI).

75.4. USING THE JAVA DSL

The next step instantiates the DataFormat *bindy* class associated with this record type and providing a class as a parameter.

For example the following uses the class **BindyCsvDataFormat** (which corresponds to the class associated with the CSV record type) which is configured with *com.acme.model.MyModel.class* to initialize the model objects configured in this package.

```
DataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);
```

75.4.1. Setting locale

Bindy supports configuring the locale on the dataformat, such as

```
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("us");
```

Or to use the platform default locale then use "default" as the locale name.

```
BindyCsvDataFormat bindy = new BindyCsvDataFormat(com.acme.model.MyModel.class);

bindy.setLocale("default");
```

75.4.2. Unmarshaling

```
from("file://inbox")
  .unmarshal(bindy)
  .to("direct:handleOrders");
```

Alternatively, you can use a named reference to a data format which can then be defined in your Registry e.g. your Spring XML file:

```
from("file://inbox")
  .unmarshal("myBindyDataFormat")
  .to("direct:handleOrders");
```

The Camel route will pick-up files in the inbox directory, unmarshall CSV records into a collection of model objects and send the collection
to the route referenced by **handleOrders**.

The collection returned is a **List of Map** objects. Each Map within the list contains the model objects that were marshalled out of each line of the CSV. The reason behind this is that *each line can correspond to more than one object*. This can be confusing when you simply expect one object to be returned per line.

Each object can be retrieve using its class name.

```
List<Map<String, Object>> unmarshaledModels = (List<Map<String, Object>>)
exchange.getIn().getBody();

int modelCount = 0;
for (Map<String, Object> model : unmarshaledModels) {
  for (String className : model.keySet()) {
```

```
        Object obj = model.get(className);
        LOG.info("Count : " + modelCount + ", " + obj.toString());
      }
     modelCount++;
    }

    LOG.info("Total CSV records received by the csv bean : " + modelCount);
```

Assuming that you want to extract a single Order object from this map for processing in a route, you could use a combination of a Splitter and a Processor as per the following:

```
from("file://inbox")
    .unmarshal(bindy)
    .split(body())
        .process(new Processor() {
            public void process(Exchange exchange) throws Exception {
                Message in = exchange.getIn();
                Map<String, Object> modelMap = (Map<String, Object>) in.getBody();
                in.setBody(modelMap.get(Order.class.getCanonicalName()));
            }
        })
        .to("direct:handleSingleOrder")
    .end();
```

Take care of the fact that Bindy uses CHARSET_NAME property or the CHARSET_NAME header as define in the Exchange interface to do a characterset conversion of the inputstream received for unmarshalling. In some producers (e.g. file-endpoint) you can define a characterset. The characterset conversion can already been done by this producer. Sometimes you need to remove this property or header from the exchange before sending it to the unmarshal. If you don't remove it the conversion might be done twice which might lead to unwanted results.

```
from("file://inbox?charset=Cp922")
  .removeProperty(Exchange.CHARSET_NAME)
  .unmarshal("myBindyDataFormat")
  .to("direct:handleOrders");
```

75.4.3. Marshaling

To generate CSV records from a collection of model objects, you create the following route :

```
from("direct:handleOrders")
    .marshal(bindy)
    .to("file://outbox")
```

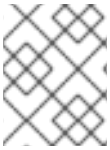
75.5. USING SPRING XML

This is really easy to use Spring as your favorite DSL language to declare the routes to be used for camel-bindy. The following example shows two routes where the first will pick-up records from files, unmarshal the content and bind it to their model. The result is then send to a pojo (doing nothing special) and place them into a queue.

The second route will extract the pojos from the queue and marshal the content to generate a file containing the CSV record.

Spring DSL

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   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">

   <!-- Queuing engine - ActiveMq - work locally in mode virtual memory -->
   <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
      <property name="brokerURL" value="vm://localhost:61616"/>
   </bean>

   <camelContext xmlns="http://camel.apache.org/schema/spring">
      <dataFormats>
        <bindy id="bindyDataformat" type="Csv" classType="org.apache.camel.bindy.model.Order"/>
      </dataFormats>

      <route>
         <from uri="file://src/data/csv/?noop=true" />
         <unmarshal ref="bindyDataformat" />
         <to uri="bean:csv" />
         <to uri="activemq:queue:in" />
      </route>

      <route>
         <from uri="activemq:queue:in" />
         <marshal ref="bindyDataformat" />
         <to uri="file://src/data/csv/out/" />
      </route>
   </camelContext>
</beans>
```



NOTE

Please verify that your model classes implements serializable otherwise the queue manager will raise an error.

75.6. DEPENDENCIES

To use Bindy in your camel routes you need to add the a dependency on **camel-bindy** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-bindy</artifactId>
```

```
  <version>{CamelSBVersion}</version>
</dependency>
```

75.7. SPRING BOOT AUTO-CONFIGURATION

When using bindy-csv with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-bindy-starter</artifactId>
</dependency>
```

The component supports 18 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.bindy-csv.allow-empty-stream	Whether to allow empty streams in the unmarshal process. If true, no exception will be thrown when a body without records is provided.	false	Boolean
camel.dataformat.bindy-csv.class-type	Name of model class to use.		String
camel.dataformat.bindy-csv.enabled	Whether to enable auto configuration of the bindy-csv data format. This is enabled by default.		Boolean
camel.dataformat.bindy-csv.locale	To configure a default locale to use, such as us for united states. To use the JVM platform default locale then use the name default.		String
camel.dataformat.bindy-csv.type	Whether to use Csv, Fixed, or KeyValue.		String
camel.dataformat.bindy-csv.unwrap-single-instance	When unmarshalling should a single instance be unwrapped and returned instead of wrapped in a java.util.List.	true	Boolean
camel.dataformat.bindy-fixed.allow-empty-stream	Whether to allow empty streams in the unmarshal process. If true, no exception will be thrown when a body without records is provided.	false	Boolean
camel.dataformat.bindy-fixed.class-type	Name of model class to use.		String

Name	Description	Default	Type
camel.dataformat.bindy-fixed.enabled	Whether to enable auto configuration of the bindy-fixed data format. This is enabled by default.		Boolean
camel.dataformat.bindy-fixed.locale	To configure a default locale to use, such as us for united states. To use the JVM platform default locale then use the name default.		String
camel.dataformat.bindy-fixed.type	Whether to use Csv, Fixed, or KeyValue.		String
camel.dataformat.bindy-fixed.unwrap-single-instance	When unmarshalling should a single instance be unwrapped and returned instead of wrapped in a java.util.List.	true	Boolean
camel.dataformat.bindy-kvp.allow-empty-stream	Whether to allow empty streams in the unmarshal process. If true, no exception will be thrown when a body without records is provided.	false	Boolean
camel.dataformat.bindy-kvp.class-type	Name of model class to use.		String
camel.dataformat.bindy-kvp.enabled	Whether to enable auto configuration of the bindy-kvp data format. This is enabled by default.		Boolean
camel.dataformat.bindy-kvp.locale	To configure a default locale to use, such as us for united states. To use the JVM platform default locale then use the name default.		String
camel.dataformat.bindy-kvp.type	Whether to use Csv, Fixed, or KeyValue.		String
camel.dataformat.bindy-kvp.unwrap-single-instance	When unmarshalling should a single instance be unwrapped and returned instead of wrapped in a java.util.List.	true	Boolean

CHAPTER 76. HL7

The HL7 component is used for working with the HL7 MLLP protocol and HL7 v2 messages using the HAPI library.

This component supports the following:

- HL7 MLLP codec for Mina
- HL7 MLLP codec for Netty
- Type Converter from/to HAPI and String
- HL7 DataFormat using the HAPI library

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-hl7</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

76.1. HL7 MLLP PROTOCOL

HL7 is often used with the HL7 MLLP protocol, which is a text based TCP socket based protocol. This component ships with a Mina and Netty Codec that conforms to the MLLP protocol so you can easily expose an HL7 listener accepting HL7 requests over the TCP transport layer. To expose a HL7 listener service, the camel-mina or link:camel-netty component is used with the HL7MLLPCodec (mina) or HL7MLLPNettyDecoder/HL7MLLPNettyEncoder (Netty).

HL7 MLLP codec can be configured as follows:

Name	Default Value	Description
startByte	**0x0b**	The start byte spanning the HL7 payload.
endByte1	**0x1c**	The first end byte spanning the HL7 payload.
endByte2	**0x0d**	The 2nd end byte spanning the HL7 payload.
charset	JVM Default	The encoding (a charset name) to use for the codec. If not provided, Camel will use the JVM default Charset
produceString	**true**	If true, the codec creates a string using the defined charset. If false, the codec sends a plain byte array into the route, so that the HL7 Data Format can determine the actual charset from the HL7 message content.
convertLFtoCR	**false**	Will convert **\n** to **\r** (**0x0d**, 13 decimal) as HL7 stipulates **\r** as segment terminators. The HAPI library requires the use of **\r**.

76.1.1. Exposing an HL7 listener using Mina

In the Spring XML file, we configure a mina endpoint to listen for HL7 requests using TCP on port **8888**:

```
<endpoint id="hl7MinaListener" uri="mina:tcp://localhost:8888?sync=true&amp;codec=#hl7codec"/>
```

sync=true indicates that this listener is synchronous and therefore will return a HL7 response to the caller. The HL7 codec is setup with **codec=#hl7codec**. Note that **hl7codec** is just a Spring bean ID, so it could be named **mygreatcodecforhl7** or whatever. The codec is also set up in the Spring XML file:

```
<bean id="hl7codec" class="org.apache.camel.component.hl7.HL7MLLPCodec">
    <property name="charset" value="iso-8859-1"/>
</bean>
```

The endpoint **hl7MinaLlistener** can then be used in a route as a consumer, as this Java DSL example illustrates:

```
from("hl7MinaListener")
  .bean("patientLookupService");
```

This is a very simple route that will listen for HL7 and route it to a service named **patientLookupService**. This is also Spring bean ID, configured in the Spring XML as:

```
<bean id="patientLookupService"
class="com.mycompany.healthcare.service.PatientLookupService"/>
```

The business logic can be implemented in POJO classes that do not depend on Camel, as shown here:

```
import ca.uhn.hl7v2.HL7Exception;
import ca.uhn.hl7v2.model.Message;
import ca.uhn.hl7v2.model.v24.segment.QRD;

public class PatientLookupService {
    public Message lookupPatient(Message input) throws HL7Exception {
        QRD qrd = (QRD)input.get("QRD");
        String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();

        // find patient data based on the patient id and create a HL7 model object with the response
        Message response = ... create and set response data
        return response
    }
```

76.1.2. Exposing an HL7 listener using Netty (available from Camel 2.15 onwards)

In the Spring XML file, we configure a netty endpoint to listen for HL7 requests using TCP on port **8888**:

```
<endpoint id="hl7NettyListener" uri="netty:tcp://localhost:8888?
sync=true&amp;encoders=#hl7encoder&amp;decoders=#hl7decoder"/>
```

sync=true indicates that this listener is synchronous and therefore will return a HL7 response to the caller. The HL7 codec is setup with **encoders=#hl7encoder*and*decoders=#hl7decoder**. Note that **hl7encoder** and **hl7decoder** are just bean IDs, so they could be named differently. The beans can be set in the Spring XML file:

```
<bean id="hl7decoder" class="org.apache.camel.component.hl7.HL7MLLPNettyDecoderFactory"/>
<bean id="hl7encoder" class="org.apache.camel.component.hl7.HL7MLLPNettyEncoderFactory"/>
```

The endpoint **hl7NettyListener** can then be used in a route as a consumer, as this Java DSL example illustrates:

```
from("hl7NettyListener")
  .bean("patientLookupService");
```

76.2. HL7 MODEL USING JAVA.LANG.STRING OR BYTE[]

The HL7 MLLP codec uses plain String as its data format. Camel uses its Type Converter to convert to/from strings to the HAPI HL7 model objects, but you can use the plain String objects if you prefer, for instance if you wish to parse the data yourself.

You can also let both the Mina and Netty codecs use a plain **byte[]** as its data format by setting the **produceString** property to false. The Type Converter is also capable of converting the **byte[]** to/from HAPI HL7 model objects.

76.3. HL7V2 MODEL USING HAPI

The HL7v2 model uses Java objects from the HAPI library. Using this library, you can encode and decode from the EDI format (ER7) that is mostly used with HL7v2.

The sample below is a request to lookup a patient with the patient ID **0101701234**.

```
MSH|^~\\&|MYSENDER|MYRECEIVER|MYAPPLICATION||200612211200||QRY^A19|1234|P|2.4
QRD|200612211200|R|I|GetPatient|||1^RD|0101701234|DEM||
```

Using the HL7 model you can work with a **ca.uhn.hl7v2.model.Message** object, e.g. to retrieve a patient ID:

```
Message msg = exchange.getIn().getBody(Message.class);
QRD qrd = (QRD)msg.get("QRD");
String patientId = qrd.getWhoSubjectFilter(0).getIDNumber().getValue();  // 0101701234
```

This is powerful when combined with the HL7 listener, because you don't have to work with **byte[]**, **String** or any other simple object formats. You can just use the HAPI HL7v2 model objects. If you know the message type in advance, you can be more type-safe:

```
QRY_A19 msg = exchange.getIn().getBody(QRY_A19.class);
String patientId = msg.getQRD().getWhoSubjectFilter(0).getIDNumber().getValue();
```

76.4. HL7 DATAFORMAT

The **camel-hl7** JAR ships with a HL7 data format that can be used to marshal or unmarshal HL7 model objects.

The HL7 dataformat supports 1 options, which are listed below.

Name	Default	Java Type	Description
validate		**Boolean**	Whether to validate the HL7 message Is by default true.

- **marshal** = from Message to byte stream (can be used when responding using the HL7 MLLP codec)
- **unmarshal** = from byte stream to Message (can be used when receiving streamed data from the HL7 MLLP

To use the data format, simply instantiate an instance and invoke the marshal or unmarshal operation in the route builder:

```
DataFormat hl7 = new HL7DataFormat();

from("direct:hl7in")
  .marshal(hl7)
  .to("jms:queue:hl7out");
```

In the sample above, the HL7 is marshalled from a HAPI Message object to a byte stream and put on a JMS queue.
The next example is the opposite:

```
DataFormat hl7 = new HL7DataFormat();

from("jms:queue:hl7out")
  .unmarshal(hl7)
  .to("patientLookupService");
```

Here we unmarshal the byte stream into a HAPI Message object that is passed to our patient lookup service.

76.4.1. Segment separators

Unmarshalling does not automatically fix segment separators anymore by converting **\n** to **\r**. If you need this conversion, **org.apache.camel.component.hl7.HL7#convertLFToCR** provides a handy **Expression** for this purpose.

76.4.2. Charset

Both **marshal and unmarshal** evaluate the charset provided in the field **MSH-18**. If this field is empty, by default the charset contained in the corresponding Camel charset property/header is assumed. You can even change this default behavior by overriding the **guessCharsetName** method when inheriting from the **HL7DataFormat** class.

There is a shorthand syntax in Camel for well-known data formats that are commonly used. Then you don't need to create an instance of the **HL7DataFormat** object:

```
from("direct:hl7in")
  .marshal().hl7()
  .to("jms:queue:hl7out");
```

```
from("jms:queue:hl7out")
  .unmarshal().hl7()
  .to("patientLookupService");
```

76.5. MESSAGE HEADERS

The unmarshal operation adds these fields from the MSH segment as headers on the Camel message:

Key	MSH field	Example
CamelHL7SendingApplication	**MSH-3**	**MYSERVER**
CamelHL7SendingFacility	**MSH-4**	**MYSERVERAPP**
CamelHL7ReceivingApplication	**MSH-5**	**MYCLIENT**
CamelHL7ReceivingFacility	**MSH-6**	**MYCLIENTAPP**
CamelHL7Timestamp	**MSH-7**	**20071231235900**
CamelHL7Security	**MSH-8**	**null**
CamelHL7MessageType	**MSH-9-1**	**ADT**
CamelHL7TriggerEvent	**MSH-9-2**	**A01**
CamelHL7MessageControl	**MSH-10**	**1234**
CamelHL7ProcessingId	**MSH-11**	**P**
CamelHL7VersionId	**MSH-12**	**2.4**
CamelHL7Context	``	**contains the that was used to parse the message**

Key	MSH field	Example
CamelHL7Charset	**MSH-18**	**UNICODE UTF-8**

All headers except **CamelHL7Context** are **String** types. If a header value is missing, its value is **null**.

76.6. DEPENDENCIES

To use HL7 in your Camel routes you'll need to add a dependency on **camel-hl7** listed above, which implements this data format.

The HAPI library is split into a base library and several structure libraries, one for each HL7v2 message version:

- v2.1 structures library
- v2.2 structures library
- v2.3 structures library
- v2.3.1 structures library
- v2.4 structures library
- v2.5 structures library
- v2.5.1 structures library
- v2.6 structures library

By default **camel-hl7** only references the HAPI base library. Applications are responsible for including structure libraries themselves. For example, if an application works with HL7v2 message versions 2.4 and 2.5 then the following dependencies must be added:

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v24</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
<dependency>
  <groupId>ca.uhn.hapi</groupId>
  <artifactId>hapi-structures-v25</artifactId>
  <version>2.2</version>
  <!-- use the same version as your hapi-base version -->
</dependency>
```

Alternatively, an OSGi bundle containing the base library, all structures libraries and required dependencies (on the bundle classpath) can be downloaded from the central Maven repository.

```
<dependency>
  <groupId>ca.uhn.hapi</groupId>
```

```
    <artifactId>hapi-osgi-base</artifactId>
    <version>2.2</version>
</dependency>
```

76.7. SPRING BOOT AUTO-CONFIGURATION

When using hl7 with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-hl7-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.hl7.enabled	Whether to enable auto configuration of the hl7 data format. This is enabled by default.		Boolean
camel.dataformat.hl7.validate	Whether to validate the HL7 message Is by default true.	true	Boolean
camel.language.hl7terser.enabled	Whether to enable auto configuration of the hl7terser language. This is enabled by default.		Boolean
camel.language.hl7terser.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

CHAPTER 77. JACKSONXML

Jackson XML is a Data Format which uses the Jackson library with the XMLMapper extension to unmarshal an XML payload into Java objects or to marshal Java objects into an XML payload. NOTE: If you are familiar with Jackson, this XML data format behaves in the same way as its JSON counterpart, and thus can be used with classes annotated for JSON serialization/deserialization.

This extension also mimics JAXB's "Code first" approach.

This data format relies on Woodstox (especially for features like pretty printing), a fast and efficient XML processor.

```
from("activemq:My.Queue").
  unmarshal().jacksonxml().
  to("mqseries:Another.Queue");
```

77.1. JACKSONXML OPTIONS

The JacksonXML dataformat supports 15 options, which are listed below.

Name	Default	Java Type	Description
xmlMapper		**String**	Lookup and use the existing XmlMapper with the given id.
prettyPrint	false	**Boolean**	To enable pretty printing output nicely formatted. Is by default false.
unmarshalType		**String**	Class name of the java type to use when unmarshalling.
jsonView		**String**	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.
include		**String**	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.
allowJmsType		**Boolean**	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.
collectionType		**String**	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.
useList		**Boolean**	To unmarshal to a List of Map or a List of Pojo.

Name	Default	Java Type	Description
enableJaxbAnnotationModule		**Boolean**	Whether to enable the JAXB annotations module when using jackson. When enabled then JAXB annotations can be used by Jackson.
moduleClassNames		**String**	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.
moduleRefs		**String**	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.
enableFeatures		**String**	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
disableFeatures		**String**	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
allowUnmarshallType		**Boolean**	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.
contentTypeHeader		**Boolean**	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.

77.1.1. Using Jackson XML in Spring DSL

When using Data Format in Spring DSL you need to declare the data formats first. This is done in the **DataFormats** XML tag.

```
    <dataFormats>
      <!-- here we define a Xml data format with the id jack and that it should use the TestPojo as the class type when
           doing unmarshal. The unmarshalType is optional, if not provided Camel will use a Map as the type -->
      <jacksonxml id="jack" unmarshalType="org.apache.camel.component.jacksonxml.TestPojo"/>
    </dataFormats>
```

And then you can refer to this id in the route:

```
<route>
    <from uri="direct:back"/>
    <unmarshal><custom ref="jack"/></unmarshal>
    <to uri="mock:reverse"/>
 </route>
```

77.1.2. Excluding POJO fields from marshalling

When marshalling a POJO to XML you might want to exclude certain fields from the XML output. With Jackson you can use JSON views to accomplish this. First create one or more marker classes.

Use the marker classes with the **@JsonView** annotation to include/exclude certain fields. The annotation also works on getters.

Finally use the Camel **JacksonXMLDataFormat** to marshall the above POJO to XML.

Note that the weight field is missing in the resulting XML:

```
<pojo age="30" weight="70"/>
```

77.2. INCLUDE/EXCLUDE FIELDS USING THE JSONVIEW ATTRIBUTE WITH `JACKSONXML`DATAFORMAT

As an example of using this attribute you can instead of:

```
JacksonXMLDataFormat ageViewFormat = new JacksonXMLDataFormat(TestPojoView.class,
Views.Age.class);
from("direct:inPojoAgeView").
  marshal(ageViewFormat);
```

Directly specify your JSON view inside the Java DSL as:

```
from("direct:inPojoAgeView").
  marshal().jacksonxml(TestPojoView.class, Views.Age.class);
```

And the same in XML DSL:

```
<from uri="direct:inPojoAgeView"/>
  <marshal>
    <jacksonxml unmarshalType="org.apache.camel.component.jacksonxml.TestPojoView"
jsonView="org.apache.camel.component.jacksonxml.Views$Age"/>
  </marshal>
```

77.3. SETTING SERIALIZATION INCLUDE OPTION

If you want to marshal a pojo to XML, and the pojo has some fields with null values. And you want to skip these null values, then you need to set either an annotation on the pojo,

```
@JsonInclude(Include.NON_NULL)
public class MyPojo {
   ...
}
```

But this requires you to include that annotation in your pojo source code. You can also configure the Camel JacksonXMLDataFormat to set the include option, as shown below:

```
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.setInclude("NON_NULL");
```

Or from XML DSL you configure this as

```
<dataFormats>
  <jacksonxml id="jacksonxml" include="NON_NULL"/>
</dataFormats>
```

77.4. UNMARSHALLING FROM XML TO POJO WITH DYNAMIC CLASS NAME

If you use jackson to unmarshal XML to POJO, then you can now specify a header in the message that indicate which class name to unmarshal to.
The header has key **CamelJacksonUnmarshalType** if that header is present in the message, then Jackson will use that as FQN for the POJO class to unmarshal the XML payload as.

```
 For JMS end users there is the JMSType header from the JMS spec that
indicates that also. To enable support for JMSType you would need to
turn that on, on the jackson data format as shown:
```

```
JacksonDataFormat format = new JacksonDataFormat();
format.setAllowJmsType(true);
```

Or from XML DSL you configure this as

```
<dataFormats>
  <jacksonxml id="jacksonxml" allowJmsType="true"/>
</dataFormats>
```

77.5. UNMARSHALLING FROM XML TO LIST<MAP> OR LIST<POJO>

If you are using Jackson to unmarshal XML to a list of map/pojo, you can now specify this by setting **useList="true"** or use the **org.apache.camel.component.jacksonxml.ListJacksonXMLDataFormat**.
For example with Java you can do as shown below:

```
JacksonXMLDataFormat format = new ListJacksonXMLDataFormat();
// or
JacksonXMLDataFormat format = new JacksonXMLDataFormat();
format.useList();
// and you can specify the pojo class type also
format.setUnmarshalType(MyPojo.class);
```

And if you use XML DSL then you configure to use list using **useList** attribute as shown below:

```
<dataFormats>
  <jacksonxml id="jack" useList="true"/>
</dataFormats>
```

And you can specify the pojo type also

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalType="com.foo.MyPojo"/>
</dataFormats>
```

77.6. USING CUSTOM JACKSON MODULES

You can use custom Jackson modules by specifying the class names of those using the moduleClassNames option as shown below.

```
<dataFormats>
  <jacksonxml id="jack" useList="true" unmarshalType="com.foo.MyPojo"
moduleClassNames="com.foo.MyModule,com.foo.MyOtherModule"/>
</dataFormats>
```

When using moduleClassNames then the custom jackson modules are not configured, by created using default constructor and used as-is. If a custom module needs any custom configuration, then an instance of the module can be created and configured, and then use modulesRefs to refer to the module as shown below:

```
<bean id="myJacksonModule" class="com.foo.MyModule">
  ... // configure the module as you want
</bean>

<dataFormats>
  <jacksonxml id="jacksonxml" useList="true" unmarshalType="com.foo.MyPojo"
moduleRefs="myJacksonModule"/>
</dataFormats>
```

```
Multiple modules can be specified separated by comma, such as
moduleRefs="myJacksonModule,myOtherModule"
```

77.7. ENABLING OR DISABLE FEATURES USING JACKSON

Jackson has a number of features you can enable or disable, which its ObjectMapper uses. For example to disable failing on unknown properties when marshalling, you can configure this using the disableFeatures:

```
<dataFormats>
  <jacksonxml id="jacksonxml" unmarshalType="com.foo.MyPojo"
disableFeatures="FAIL_ON_UNKNOWN_PROPERTIES"/>
</dataFormats>
```

You can disable multiple features by separating the values using comma. The values for the features must be the name of the enums from Jackson from the following enum classes

- com.fasterxml.jackson.databind.SerializationFeature
- com.fasterxml.jackson.databind.DeserializationFeature
- com.fasterxml.jackson.databind.MapperFeature

To enable a feature use the enableFeatures options instead.

From Java code you can use the type safe methods from camel-jackson module:

```
JacksonDataFormat df = new JacksonDataFormat(MyPojo.class);
df.disableFeature(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
df.disableFeature(DeserializationFeature.FAIL_ON_NULL_FOR_PRIMITIVES);
```

77.8. CONVERTING MAPS TO POJO USING JACKSON

Jackson **ObjectMapper** can be used to convert maps to POJO objects. Jackson component comes with the data converter that can be used to convert **java.util.Map** instance to non-String, non-primitive and non-Number objects.

```
Map<String, Object> invoiceData = new HashMap<String, Object>();
invoiceData.put("netValue", 500);
producerTemplate.sendBody("direct:mapToInvoice", invoiceData);
...
// Later in the processor
Invoice invoice = exchange.getIn().getBody(Invoice.class);
```

If there is a single **ObjectMapper** instance available in the Camel registry, it will used by the converter to perform the conversion. Otherwise the default mapper will be used.

77.9. FORMATTED XML MARSHALLING (PRETTY-PRINTING)

Using the **prettyPrint** option one can output a well formatted XML while marshalling:

```
<dataFormats>
    <jacksonxml id="jack" prettyPrint="true"/>
</dataFormats>
```

And in Java DSL:

```
from("direct:inPretty").marshal().jacksonxml(true);
```

Please note that there are 5 different overloaded **jacksonxml()** DSL methods which support the **prettyPrint** option in combination with other settings for **unmarshalType**, **jsonView** etc.

77.10. DEPENDENCIES

To use Jackson XML in your camel routes you need to add the dependency on **camel-jacksonxml** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jacksonxml</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

77.11. SPRING BOOT AUTO-CONFIGURATION

When using jacksonxml with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jacksonxml-starter</artifactId>
</dependency>
```

The component supports 16 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.jacksonxml.allow-jms-type	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.	false	Boolean
camel.dataformat.jacksonxml.allow-unmarshall-type	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.	false	Boolean
camel.dataformat.jacksonxml.collection-type	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.		String
camel.dataformat.jacksonxml.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.jacksonxml.disable-features	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String

Name	Description	Default	Type
camel.dataformat.jacksonxml.enable-features	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.jacksonxml.enable-jaxb-annotation-module	Whether to enable the JAXB annotations module when using jackson. When enabled then JAXB annotations can be used by Jackson.	false	Boolean
camel.dataformat.jacksonxml.enabled	Whether to enable auto configuration of the jacksonxml data format. This is enabled by default.		Boolean
camel.dataformat.jacksonxml.include	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.		String
camel.dataformat.jacksonxml.json-view	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.		String
camel.dataformat.jacksonxml.module-class-names	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.		String
camel.dataformat.jacksonxml.module-refs	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.		String
camel.dataformat.jacksonxml.pretty-print	To enable pretty printing output nicely formatted. Is by default false.	false	Boolean
camel.dataformat.jacksonxml.unmarshal-type	Class name of the java type to use when unmarshalling.		String

Name	Description	Default	Type
camel.dataformat.jacksonxml.use-list	To unmarshal to a List of Map or a List of Pojo.	false	Boolean
camel.dataformat.jacksonxml.xml-mapper	Lookup and use the existing XmlMapper with the given id.		String

CHAPTER 78. JAXB

JAXB is a Data Format which uses the JAXB2 XML marshalling standard which is included in Java 6 to unmarshal an XML payload into Java objects or to marshal Java objects into an XML payload.

78.1. OPTIONS

The JAXB dataformat supports 19 options, which are listed below.

Name	Default	Java Type	Description
contextPath		**String**	**Required** Package name where your JAXB classes are located.
contextPathIsClassName		**Boolean**	This can be set to true to mark that the contextPath is referring to a classname and not a package name.
schema		**String**	To validate against an existing schema. Your can use the prefix classpath:, file: or http: to specify how the resource should by resolved. You can separate multiple schema files by using the ',' character.
schemaSeverityLevel		**Enum**	Sets the schema severity level to use when validating against a schema. This level determines the minimum severity error that triggers JAXB to stop continue parsing. The default value of 0 (warning) means that any error (warning, error or fatal error) will trigger JAXB to stop. There are the following three levels: 0=warning, 1=error, 2=fatal error. Enum values: • 0 • 1 • 2
prettyPrint		**Boolean**	To enable pretty printing output nicely formatted. Is by default false.
objectFactory		**Boolean**	Whether to allow using ObjectFactory classes to create the POJO classes during marshalling. This only applies to POJO classes that has not been annotated with JAXB and providing jaxb.index descriptor files.
ignoreJAXBElement		**Boolean**	Whether to ignore JAXBElement elements - only needed to be set to false in very special use-cases.

Name	Default	Java Type	Description
mustBeJAXBElement		**Boolean**	Whether marhsalling must be java objects with JAXB annotations. And if not then it fails. This option can be set to false to relax that, such as when the data is already in XML format.
filterNonXmlChars		**Boolean**	To ignore non xml characheters and replace them with an empty space.
encoding		**String**	To overrule and use a specific encoding.
fragment		**Boolean**	To turn on marshalling XML fragment trees. By default JAXB looks for XmlRootElement annotation on given class to operate on whole XML tree. This is useful but not always - sometimes generated code does not have XmlRootElement annotation, sometimes you need unmarshall only part of tree. In that case you can use partial unmarshalling. To enable this behaviours you need set property partClass. Camel will pass this class to JAXB's unmarshaler.
partClass		**String**	Name of class used for fragment parsing. See more details at the fragment option.
partNamespace		**String**	XML namespace to use for fragment parsing. See more details at the fragment option.
namespacePrefixRef		**String**	When marshalling using JAXB or SOAP then the JAXB implementation will automatic assign namespace prefixes, such as ns2, ns3, ns4 etc. To control this mapping, Camel allows you to refer to a map which contains the desired mapping.
xmlStreamWriterWrapper		**String**	To use a custom xml stream writer.
schemaLocation		**String**	To define the location of the schema.
noNamespaceSchemaLocation		**String**	To define the location of the namespaceless schema.
jaxbProviderProperties		**String**	Refers to a custom java.util.Map to lookup in the registry containing custom JAXB provider properties to be used with the JAXB marshaller.
contentTypeHeader		**Boolean**	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.

78.2. USING THE JAVA DSL

For example the following uses a named DataFormat of **jaxb** which is configured with a number of Java package names to initialize the JAXBContext.

```
DataFormat jaxb = new JaxbDataFormat("com.acme.model");

from("activemq:My.Queue").
  unmarshal(jaxb).
  to("mqseries:Another.Queue");
```

You can if you prefer use a named reference to a data format which can then be defined in your Registry such as via your Spring XML file. e.g.

```
from("activemq:My.Queue").
  unmarshal("myJaxbDataType").
  to("mqseries:Another.Queue");
```

78.3. USING SPRING XML

The following example shows how to configure the **JaxbDataFormat** and use it in multiple routes.

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

  <bean id="myJaxb" class="org.apache.camel.converter.jaxb.JaxbDataFormat">
    <property name="contextPath" value="org.apache.camel.example"/>
  </bean>

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="direct:start"/>
      <marshal><custom ref="myJaxb"/></marshal>
      <to uri="direct:marshalled"/>
    </route>
    <route>
      <from uri="direct:marshalled"/>
      <unmarshal><custom ref="myJaxb"/></unmarshal>
      <to uri="mock:result"/>
    </route>
  </camelContext>

</beans>
```

Multiple context paths

It is possible to use this data format with more than one context path. You can specify context path using **:** as separator, for example **com.mycompany:com.mycompany2**. Note that this is handled by JAXB implementation and might change if you use different vendor than RI.

78.4. PARTIAL MARSHALLING/UNMARSHALLING

JAXB 2 supports marshalling and unmarshalling XML tree fragments. By default JAXB looks for **@XmlRootElement** annotation on given class to operate on whole XML tree. This is useful but not always - sometimes generated code does not have @XmlRootElement annotation, sometimes you need unmarshall only part of tree.

In that case you can use partial unmarshalling. To enable this behaviours you need set property **partClass**. Camel will pass this class to JAXB's unmarshaler. If **JaxbConstants.JAXB_PART_CLASS** is set as one of headers, (even if partClass property is set on DataFormat), the property on DataFormat is surpassed and the one set in the headers is used.

For marshalling you have to add **partNamespace** attribute with QName of destination namespace. Example of Spring DSL you can find above.

If **JaxbConstants.JAXB_PART_NAMESPACE** is set as one of headers, (even if partNamespace property is set on DataFormat), the property on DataFormat is surpassed and the one set in the headers is used. While setting **partNamespace** through **JaxbConstants.JAXB_PART_NAMESPACE**, please note that you need to specify its value \{[namespaceUri]}[localPart]

```
...
.setHeader(JaxbConstants.JAXB_PART_NAMESPACE, simple("
{http://www.camel.apache.org/jaxb/example/address/1}address"));
...
```

78.5. FRAGMENT

JaxbDataFormat has new property fragment which can set the the **Marshaller.JAXB_FRAGMENT** encoding property on the JAXB Marshaller. If you don't want the JAXB Marshaller to generate the XML declaration, you can set this option to be true. The default value of this property is false.

78.6. IGNORING THE NONXML CHARACTER

JaxbDataFormat supports to ignore the NonXML Character, you just need to set the filterNonXmlChars property to be true, JaxbDataFormat will replace the NonXML character with " " when it is marshaling or unmarshaling the message. You can also do it by setting the Exchange property **Exchange.FILTER_NON_XML_CHARS**.

	JDK 1.5	JDK 1.6+
Filtering in use	StAX API and implementation	No
Filtering not in use	StAX API only	No

This feature has been tested with Woodstox 3.2.9 and Sun JDK 1.6 StAX implementation.

JaxbDataFormat now allows you to customize the XMLStreamWriter used to marshal the stream to XML. Using this configuration, you can add your own stream writer to completely remove, escape, or replace non-xml characters.

```
JaxbDataFormat customWriterFormat = new JaxbDataFormat("org.apache.camel.foo.bar");
customWriterFormat.setXmlStreamWriterWrapper(new TestXmlStreamWriter());
```

The following example shows using the Spring DSL and also enabling Camel's NonXML filtering:

```
<bean id="testXmlStreamWriterWrapper" class="org.apache.camel.jaxb.TestXmlStreamWriter"/>
<jaxb filterNonXmlChars="true"  contextPath="org.apache.camel.foo.bar"
xmlStreamWriterWrapper="#testXmlStreamWriterWrapper" />
```

78.7. WORKING WITH THE OBJECTFACTORY

If you use XJC to create the java class from the schema, you will get an ObjectFactory for you JAXB context. Since the ObjectFactory uses JAXBElement to hold the reference of the schema and element instance value, jaxbDataformat will ignore the JAXBElement by default and you will get the element instance value instead of the JAXBElement object form the unmarshaled message body.

If you want to get the JAXBElement object form the unmarshaled message body, you need to set the JaxbDataFormat object's ignoreJAXBElement property to be false.

78.8. SETTING ENCODING

You can set the **encoding** option to use when marshalling. Its the **Marshaller.JAXB_ENCODING** encoding property on the JAXB Marshaller.

You can setup which encoding to use when you declare the JAXB data format. You can also provide the encoding in the Exchange property **Exchange.CHARSET_NAME**. This property will overrule the encoding set on the JAXB data format.

In this Spring DSL we have defined to use **iso-8859-1** as the encoding.

78.9. CONTROLLING NAMESPACE PREFIX MAPPING

When marshalling using JAXB or SOAP then the JAXB implementation will automatic assign namespace prefixes, such as ns2, ns3, ns4 etc. To control this mapping, Camel allows you to refer to a map which contains the desired mapping.

Notice this requires having JAXB-RI 2.1 or better (from SUN) on the classpath, as the mapping functionality is dependent on the implementation of JAXB, whether its supported.

For example in Spring XML we can define a Map with the mapping. In the mapping file below, we map SOAP to use soap as prefix. While our custom namespace "http://www.mycompany.com/foo/2" is not using any prefix.

```
<util:map id="myMap">
  <entry key="http://www.w3.org/2003/05/soap-envelope" value="soap"/>
  <!-- we dont want any prefix for our namespace -->
  <entry key="http://www.mycompany.com/foo/2" value=""/>
</util:map>
```

To use this in JAXB or SOAP you refer to this map, using the **namespacePrefixRef** attribute as shown below. Then Camel will lookup in the Registry a **java.util.Map** with the id "myMap", which was what we defined above.

```
<marshal>
  <soapjaxb version="1.2" contextPath="com.mycompany.foo" namespacePrefixRef="myMap"/>
</marshal>
```

78.10. SCHEMA VALIDATION

The JAXB Data Format supports validation by marshalling and unmarshalling from/to XML. Your can use the prefix **classpath:**, **file:** or **http:** to specify how the resource should by resolved. You can separate multiple schema files by using the **','** character.

Using the Java DSL, you can configure it in the following way:

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchema("classpath:person.xsd,classpath:address.xsd");
```

You can do the same using the XML DSL:

```
<marshal>
   <jaxb id="jaxb" schema="classpath:person.xsd,classpath:address.xsd"/>
</marshal>
```

Camel will create and pool the underling **SchemaFactory** instances on the fly, because the **SchemaFactory** shipped with the JDK is not thread safe.
However, if you have a **SchemaFactory** implementation which is thread safe, you can configure the JAXB data format to use this one:

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setSchemaFactory(thradSafeSchemaFactory);
```

78.11. SCHEMA LOCATION

The JAXB Data Format supports to specify the SchemaLocation when marshaling the XML.

Using the Java DSL, you can configure it in the following way:

```
JaxbDataFormat jaxbDataFormat = new JaxbDataFormat();
jaxbDataFormat.setContextPath(Person.class.getPackage().getName());
jaxbDataFormat.setSchemaLocation("schema/person.xsd");
```

You can do the same using the XML DSL:

```
<marshal>
   <jaxb id="jaxb" schemaLocation="schema/person.xsd"/>
</marshal>
```

78.12. MARSHAL DATA THAT IS ALREADY XML

The JAXB marshaller requires that the message body is JAXB compatible, eg its a JAXBElement, eg a java instance that has JAXB annotations, or extend JAXBElement. There can be situations where the message body is already in XML, eg from a String type.

There is a new option **mustBeJAXBElement** you can set to false, to relax this check, so the JAXB marshaller only attempts to marshal JAXBElements (javax.xml.bind.JAXBIntrospector#isElement returns true). And in those situations the marshaller fallbacks to marshal the message body as-is.

78.13. DEPENDENCIES

To use JAXB in your camel routes you need to add the a dependency on **camel-jaxb** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jaxb</artifactId>
  <version>{CamelSBVersion}</version>
</dependency>
```

78.14. SPRING BOOT AUTO-CONFIGURATION

When using jaxb with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jaxb-starter</artifactId>
</dependency>
```

The component supports 20 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.jaxb.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.jaxb.context-path	Package name where your JAXB classes are located.		String
camel.dataformat.jaxb.context-path-is-class-name	This can be set to true to mark that the contextPath is referring to a classname and not a package name.	false	Boolean

Name	Description	Default	Type
camel.dataformat.jaxb.enabled	Whether to enable auto configuration of the jaxb data format. This is enabled by default.		Boolean
camel.dataformat.jaxb.encoding	To overrule and use a specific encoding.		String
camel.dataformat.jaxb.filter-non-xml-chars	To ignore non xml characheters and replace them with an empty space.	false	Boolean
camel.dataformat.jaxb.fragment	To turn on marshalling XML fragment trees. By default JAXB looks for XmlRootElement annotation on given class to operate on whole XML tree. This is useful but not always - sometimes generated code does not have XmlRootElement annotation, sometimes you need unmarshall only part of tree. In that case you can use partial unmarshalling. To enable this behaviours you need set property partClass. Camel will pass this class to JAXB's unmarshaler.	false	Boolean
camel.dataformat.jaxb.ignore-j-a-x-b-element	Whether to ignore JAXBElement elements - only needed to be set to false in very special use-cases.	false	Boolean
camel.dataformat.jaxb.jaxb-provider-properties	Refers to a custom java.util.Map to lookup in the registry containing custom JAXB provider properties to be used with the JAXB marshaller.		String
camel.dataformat.jaxb.must-be-j-a-x-b-element	Whether marhsalling must be java objects with JAXB annotations. And if not then it fails. This option can be set to false to relax that, such as when the data is already in XML format.	false	Boolean
camel.dataformat.jaxb.namespace-prefix-ref	When marshalling using JAXB or SOAP then the JAXB implementation will automatic assign namespace prefixes, such as ns2, ns3, ns4 etc. To control this mapping, Camel allows you to refer to a map which contains the desired mapping.		String
camel.dataformat.jaxb.no-namespace-schema-location	To define the location of the namespaceless schema.		String

Name	Description	Default	Type
camel.dataformat.jaxb.object-factory	Whether to allow using ObjectFactory classes to create the POJO classes during marshalling. This only applies to POJO classes that has not been annotated with JAXB and providing jaxb.index descriptor files.	false	Boolean
camel.dataformat.jaxb.part-class	Name of class used for fragment parsing. See more details at the fragment option.		String
camel.dataformat.jaxb.part-namespace	XML namespace to use for fragment parsing. See more details at the fragment option.		String
camel.dataformat.jaxb.pretty-print	To enable pretty printing output nicely formatted. Is by default false.	false	Boolean
camel.dataformat.jaxb.schema	To validate against an existing schema. Your can use the prefix classpath:, file: or http: to specify how the resource should by resolved. You can separate multiple schema files by using the ',' character.		String
camel.dataformat.jaxb.schema-location	To define the location of the schema.		String
camel.dataformat.jaxb.schema-severity-level	Sets the schema severity level to use when validating against a schema. This level determines the minimum severity error that triggers JAXB to stop continue parsing. The default value of 0 (warning) means that any error (warning, error or fatal error) will trigger JAXB to stop. There are the following three levels: 0=warning, 1=error, 2=fatal error.	0	Integer
camel.dataformat.jaxb.xml-stream-writer-wrapper	To use a custom xml stream writer.		String

CHAPTER 79. JSON GSON

Gson is a Data Format which uses the Gson Library.

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Gson).
  to("mqseries:Another.Queue");
```

79.1. GSON OPTIONS

The JSON Gson dataformat supports 3 options, which are listed below.

Name	Default	Java Type	Description
prettyPrint		**Boolean**	To enable pretty printing output nicely formatted. Is by default false.
unmarshalType		**String**	Class name of the java type to use when unmarshalling.
contentTypeHeader		**Boolean**	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.

79.2. DEPENDENCIES

To use Gson in your camel routes you need to add the dependency on **camel-gson** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-gson</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

79.3. SPRING BOOT AUTO-CONFIGURATION

When using json-gson with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-gson-starter</artifactId>
</dependency>
```

The component supports 4 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.json-gson.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.json-gson.enabled	Whether to enable auto configuration of the json-gson data format. This is enabled by default.		Boolean
camel.dataformat.json-gson.pretty-print	To enable pretty printing output nicely formatted. Is by default false.	false	Boolean
camel.dataformat.json-gson.unmarshal-type	Class name of the java type to use when unmarshalling.		String

CHAPTER 80. JSON JACKSON

Jackson is a Data Format which uses the Jackson Library

```
from("activemq:My.Queue").
  marshal().json(JsonLibrary.Jackson).
  to("mqseries:Another.Queue");
```

80.1. JACKSON OPTIONS

The JSON Jackson dataformat supports 20 options, which are listed below.

Name	Default	Java Type	Description
objectMapper		**String**	Lookup and use the existing ObjectMapper with the given id when using Jackson.
useDefaultObjectMapper		**Boolean**	Whether to lookup and use default Jackson ObjectMapper from the registry.
prettyPrint		**Boolean**	To enable pretty printing output nicely formatted. Is by default false.
unmarshalType		**String**	Class name of the java type to use when unmarshalling.

Name	Default	Java Type	Description
jsonView		String	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.
include		String	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.
allowJmsType		Boolean	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.
collectionType		String	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.

Name	Default	Java Type	Description
useList		Boolean	To unmarshal to a List of Map or a List of Pojo.
moduleClassNames		String	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.
moduleRefs		String	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.
enableFeatures		String	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.

Name	Default	Java Type	Description
disableFeatures		String	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
allowUnmarshallType		Boolean	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.
timezone		String	If set then Jackson will use the Timezone when marshalling/unmarshalling. This option will have no effect on the others Json DataFormat, like gson, fastjson and xstream.

Name	Default	Java Type	Description
autoDiscoverObjectMapper		Boolean	If set to true then Jackson will lookup for an objectMapper into the registry.
contentTypeHeader		Boolean	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.
schemaResolver		String	Optional schema resolver used to lookup schemas for the data in transit.
autoDiscoverSchemaResolver		Boolean	When not disabled, the SchemaResolver will be looked up into the registry.

Name	Default	Java Type	Description
namingStrategy		**String**	If set then Jackson will use the the defined Property Naming Strategy.Possible values are: LOWER_CAMEL_CASE, LOWER_DOT_CASE, LOWER_CASE, KEBAB_CASE, SNAKE_CASE and UPPER_CAMEL_CASE.

80.2. USING CUSTOM OBJECTMAPPER

You can configure **JacksonDataFormat** to use a custom **ObjectMapper** in case you need more control of the mapping configuration.

If you setup a single **ObjectMapper** in the registry, then Camel will automatic lookup and use this **ObjectMapper**. For example if you use Spring Boot, then Spring Boot can provide a default **ObjectMapper** for you if you have Spring MVC enabled. And this would allow Camel to detect that there is one bean of **ObjectMapper** class type in the Spring Boot bean registry and then use it. When this happens you should set a **INFO** logging from Camel.

80.3. USING JACKSON FOR AUTOMATIC TYPE CONVERSION

The **camel-jackson** module allows integrating Jackson as a Type Converter. This works in a similar way to JAXB that integrates with Camel's type converter.

To use this **camel-jackson** must be enabled, which is done by setting the following options on the **CamelContext** global options, as shown:

```
@Bean
CamelContextConfiguration contextConfiguration() {
    return new CamelContextConfiguration() {
        @Override
        public void beforeApplicationStart(CamelContext context) {
        // Enable Jackson JSON type converter.
            context.getGlobalOptions().put(JacksonConstants.ENABLE_TYPE_CONVERTER, "true");
        // Allow Jackson JSON to convert to pojo types also
        // (by default Jackson only converts to String and other simple types)
            getContext().getGlobalOptions().put(JacksonConstants.TYPE_CONVERTER_TO_POJO,
"true");
        }
```

```
        @Override
        public void afterApplicationStart(CamelContext camelContext) {

        }
    };
}
```

The **camel-jackson** type converter integrates with JAXB which means you can annotate POJO class with **JAXB** annotations that Jackson can use. You can also use Jackson's own annotations on your POJO classes.

80.4. DEPENDENCIES

To use Jackson in your camel routes you need to add the dependency on **camel-jackson** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

80.5. SPRING BOOT AUTO-CONFIGURATION

When using json-jackson with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jackson-starter</artifactId>
</dependency>
```

The component supports 21 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.json-jackson.allow-jms-type	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.	false	Boolean
camel.dataformat.json-jackson.allow-unmarshall-type	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.	false	Boolean

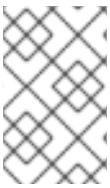
Name	Description	Default	Type
camel.dataformat.json-jackson.auto-discover-object-mapper	If set to true then Jackson will lookup for an objectMapper into the registry.	false	Boolean
camel.dataformat.json-jackson.auto-discover-schema-resolver	When not disabled, the SchemaResolver will be looked up into the registry.	true	Boolean
camel.dataformat.json-jackson.collection-type	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.		String
camel.dataformat.json-jackson.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean
camel.dataformat.json-jackson.disable-features	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.json-jackson.enable-features	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.json-jackson.enabled	Whether to enable auto configuration of the json-jackson data format. This is enabled by default.		Boolean
camel.dataformat.json-jackson.include	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.		String

Name	Description	Default	Type
camel.dataformat.json-jackson.json-view	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.		String
camel.dataformat.json-jackson.module-class-names	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.		String
camel.dataformat.json-jackson.module-refs	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.		String
camel.dataformat.json-jackson.naming-strategy	If set then Jackson will use the the defined Property Naming Strategy.Possible values are: LOWER_CAMEL_CASE, LOWER_DOT_CASE, LOWER_CASE, KEBAB_CASE, SNAKE_CASE and UPPER_CAMEL_CASE.		String
camel.dataformat.json-jackson.object-mapper	Lookup and use the existing ObjectMapper with the given id when using Jackson.		String
camel.dataformat.json-jackson.pretty-print	To enable pretty printing output nicely formatted. Is by default false.	false	Boolean
camel.dataformat.json-jackson.schema-resolver	Optional schema resolver used to lookup schemas for the data in transit.		String
camel.dataformat.json-jackson.timezone	If set then Jackson will use the Timezone when marshalling/unmarshalling. This option will have no effect on the others Json DataFormat, like gson, fastjson and xstream.		String
camel.dataformat.json-jackson.unmarshal-type	Class name of the java type to use when unmarshalling.		String

Name	Description	Default	Type
camel.dataformat.json-jackson.use-default-object-mapper	Whether to lookup and use default Jackson ObjectMapper from the registry.	true	Boolean
camel.dataformat.json-jackson.use-list	To unmarshal to a List of Map or a List of Pojo.	false	Boolean

CHAPTER 81. PROTOBUF JACKSON

Jackson Protobuf is a Data Format which uses the Jackson library with the Protobuf extension to unmarshal a Protobuf payload into Java objects or to marshal Java objects into a Protobuf payload.



NOTE

If you are familiar with Jackson, this Protobuf data format behaves in the same way as its JSON counterpart, and thus can be used with classes annotated for JSON serialization/deserialization.

```
from("kafka:topic").
  unmarshal().protobuf(ProtobufLibrary.Jackson, JsonNode.class).
  to("log:info");
```

81.1. CONFIGURING THE SCHEMARESOLVER

Since Protobuf serialization is schema-based, this data format requires that you provide a SchemaResolver object that is able to lookup the schema for each exchange that is going to be marshalled/unmarshalled.

You can add a single SchemaResolver to the registry and it will be looked up automatically. Or you can explicitly specify the reference to a custom SchemaResolver.

81.2. PROTOBUF JACKSON OPTIONS

The Protobuf Jackson dataformat supports 18 options, which are listed below.

Name	Default	Java Type	Description
contentTypeHeader		**Boolean**	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.
objectMapper		**String**	Lookup and use the existing ObjectMapper with the given id when using Jackson.
useDefaultObjectMapper		**Boolean**	Whether to lookup and use default Jackson ObjectMapper from the registry.
unmarshalType		**String**	Class name of the java type to use when unmarshalling.

Name	Default	Java Type	Description
jsonView		String	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.
include		String	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.
allowJmsType		Boolean	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.
collectionType		String	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.
useList		Boolean	To unmarshal to a List of Map or a List of Pojo.
moduleClassNames		String	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.
moduleRefs		String	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.
enableFeatures		String	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.

Name	Default	Java Type	Description
disableFeatures		String	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.
allowUnmarshallType		Boolean	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.
timezone		String	If set then Jackson will use the Timezone when marshalling/unmarshalling.
autoDiscoverObjectMapper		Boolean	If set to true then Jackson will lookup for an objectMapper into the registry.
schemaResolver		String	Optional schema resolver used to lookup schemas for the data in transit.
autoDiscoverSchemaResolver		Boolean	When not disabled, the SchemaResolver will be looked up into the registry.

81.3. USING CUSTOM PROTOBUFMAPPER

You can configure **JacksonProtobufDataFormat** to use a custom **ProtobufMapper** in case you need more control of the mapping configuration.

If you setup a single **ProtobufMapper** in the registry, then Camel will automatic lookup and use this **ProtobufMapper**.

81.4. DEPENDENCIES

To use Protobuf Jackson in your camel routes you need to add the dependency on **camel-jackson-protobuf** which implements this data format.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jackson-protobuf</artifactId>
```

```
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

81.5. SPRING BOOT AUTO-CONFIGURATION

When using protobuf-jackson with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jackson-protobuf-starter</artifactId>
</dependency>
```

The component supports 19 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.protobuf-jackson.allow-jms-type	Used for JMS users to allow the JMSType header from the JMS spec to specify a FQN classname to use to unmarshal to.	false	Boolean
camel.dataformat.protobuf-jackson.allow-unmarshall-type	If enabled then Jackson is allowed to attempt to use the CamelJacksonUnmarshalType header during the unmarshalling. This should only be enabled when desired to be used.	false	Boolean
camel.dataformat.protobuf-jackson.auto-discover-object-mapper	If set to true then Jackson will lookup for an objectMapper into the registry.	false	Boolean
camel.dataformat.protobuf-jackson.auto-discover-schema-resolver	When not disabled, the SchemaResolver will be looked up into the registry.	true	Boolean
camel.dataformat.protobuf-jackson.collection-type	Refers to a custom collection type to lookup in the registry to use. This option should rarely be used, but allows to use different collection types than java.util.Collection based as default.		String
camel.dataformat.protobuf-jackson.content-type-header	Whether the data format should set the Content-Type header with the type from the data format. For example application/xml for data formats marshalling to XML, or application/json for data formats marshalling to JSON.	true	Boolean

Name	Description	Default	Type
camel.dataformat.protobuf-jackson.disable-features	Set of features to disable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.protobuf-jackson.enable-features	Set of features to enable on the Jackson com.fasterxml.jackson.databind.ObjectMapper. The features should be a name that matches a enum from com.fasterxml.jackson.databind.SerializationFeature, com.fasterxml.jackson.databind.DeserializationFeature, or com.fasterxml.jackson.databind.MapperFeature Multiple features can be separated by comma.		String
camel.dataformat.protobuf-jackson.enabled	Whether to enable auto configuration of the protobuf-jackson data format. This is enabled by default.		Boolean
camel.dataformat.protobuf-jackson.include	If you want to marshal a pojo to JSON, and the pojo has some fields with null values. And you want to skip these null values, you can set this option to NON_NULL.		String
camel.dataformat.protobuf-jackson.json-view	When marshalling a POJO to JSON you might want to exclude certain fields from the JSON output. With Jackson you can use JSON views to accomplish this. This option is to refer to the class which has JsonView annotations.		String
camel.dataformat.protobuf-jackson.module-class-names	To use custom Jackson modules com.fasterxml.jackson.databind.Module specified as a String with FQN class names. Multiple classes can be separated by comma.		String
camel.dataformat.protobuf-jackson.module-refs	To use custom Jackson modules referred from the Camel registry. Multiple modules can be separated by comma.		String
camel.dataformat.protobuf-jackson.object-mapper	Lookup and use the existing ObjectMapper with the given id when using Jackson.		String

Name	Description	Default	Type
camel.dataformat.protobuf-jackson.schema-resolver	Optional schema resolver used to lookup schemas for the data in transit.		String
camel.dataformat.protobuf-jackson.timezone	If set then Jackson will use the Timezone when marshalling/unmarshalling.		String
camel.dataformat.protobuf-jackson.unmarshal-type	Class name of the java type to use when unmarshalling.		String
camel.dataformat.protobuf-jackson.use-default-object-mapper	Whether to lookup and use default Jackson ObjectMapper from the registry.	true	Boolean
camel.dataformat.protobuf-jackson.use-list	To unmarshal to a List of Map or a List of Pojo.	false	Boolean

CHAPTER 82. SOAP

SOAP is a Data Format which uses JAXB2 and JAX-WS annotations to marshal and unmarshal SOAP payloads. It provides the basic features of Apache CXF without need for the CXF Stack.

Namespace prefix mapping

See JAXB for details how you can control namespace prefix mappings when marshalling using SOAP data format.

82.1. SOAP OPTIONS

The SOAP dataformat supports 6 options, which are listed below.

Name	Default	Java Type	Description
contextPath		**String**	**Required** Package name where your JAXB classes are located.
encoding		**String**	To overrule and use a specific encoding.
elementNameStrategyRef		**String**	Refers to an element strategy to lookup from the registry. An element name strategy is used for two purposes. The first is to find a xml element name for a given object and soap action when marshaling the object into a SOAP message. The second is to find an Exception class for a given soap fault name. The following three element strategy class name is provided out of the box. QNameStrategy - Uses a fixed qName that is configured on instantiation. Exception lookup is not supported TypeNameStrategy - Uses the name and namespace from the XMLType annotation of the given type. If no namespace is set then package-info is used. Exception lookup is not supported ServiceInterfaceStrategy - Uses information from a webservice interface to determine the type name and to find the exception class for a SOAP fault All three classes is located in the package name org.apache.camel.dataformat.soap.name If you have generated the web service stub code with cxf-codegen or a similar tool then you probably will want to use the ServiceInterfaceStrategy. In the case you have no annotated service interface you should use QNameStrategy or TypeNameStrategy.
version		**String**	SOAP version should either be 1.1 or 1.2. Is by default 1.1.
namespacePrefixRef		**String**	When marshalling using JAXB or SOAP then the JAXB implementation will automatic assign namespace prefixes, such as ns2, ns3, ns4 etc. To control this mapping, Camel allows you to refer to a map which contains the desired mapping.

Name	Default	Java Type	Description
schema		**String**	To validate against an existing schema. Your can use the prefix classpath:, file: or http: to specify how the resource should by resolved. You can separate multiple schema files by using the ',' character.

82.2. ELEMENTNAMESTRATEGY

An element name strategy is used for two purposes. The first is to find a xml element name for a given object and soap action when marshaling the object into a SOAP message. The second is to find an Exception class for a given soap fault name.

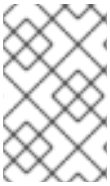
Strategy	Usage
QNameStrategy	Uses a fixed qName that is configured on instantiation. Exception lookup is not supported
TypeNameStrategy	Uses the name and namespace from the @XMLType annotation of the given type. If no namespace is set then package-info is used. Exception lookup is not supported
ServiceInterfaceStrategy	Uses information from a webservice interface to determine the type name and to find the exception class for a SOAP fault

If you have generated the web service stub code with cxf-codegen or a similar tool then you probably will want to use the ServiceInterfaceStrategy. In the case you have no annotated service interface you should use QNameStrategy or TypeNameStrategy.

82.3. USING THE JAVA DSL

The following example uses a named DataFormat of *soap* which is configured with the package com.example.customerservice to initialize the JAXBContext. The second parameter is the ElementNameStrategy. The route is able to marshal normal objects as well as exceptions. (Note the below just sends a SOAP Envelope to a queue. A web service provider would actually need to be listening to the queue for a SOAP call to actually occur, in which case it would be a one way SOAP request. If you need request reply then you should look at the next example.)

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```



NOTE

See also
As the SOAP dataformat inherits from the JAXB dataformat most settings apply here as well.

82.3.1. Using SOAP 1.2

Since Camel 2.11

```
SoapJaxbDataFormat soap = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
soap.setVersion("1.2");
from("direct:start")
  .marshal(soap)
  .to("jms:myQueue");
```

When using XML DSL there is a version attribute you can set on the <soapjaxb> element.

```
    <!-- Defining a ServiceInterfaceStrategy for retrieving the element name when marshalling -->
    <bean id="myNameStrategy"
class="org.apache.camel.dataformat.soap.name.ServiceInterfaceStrategy">
        <constructor-arg value="com.example.customerservice.CustomerService"/>
    <constructor-arg value="true"/>
    </bean>
```

And in the Camel route

```
<route>
  <from uri="direct:start"/>
  <marshal>
    <soapjaxb contentPath="com.example.customerservice" version="1.2"
elementNameStrategyRef="myNameStrategy"/>
  </marshal>
  <to uri="jms:myQueue"/>
</route>
```

82.4. MULTI-PART MESSAGES

Multi-part SOAP messages are supported by the ServiceInterfaceStrategy. The ServiceInterfaceStrategy must be initialized with a service interface definition that is annotated in accordance with JAX-WS 2.2 and meets the requirements of the Document Bare style. The target method must meet the following criteria, as per the JAX-WS specification: 1) it must have at most one **in** or **in**/**out** non-header parameter, 2) if it has a return type other than **void** it must have no **in**/**out** or **out** non-header parameters, 3) if it it has a return type of **void** it must have at most one **in**/**out** or **out** non-header parameter.

The ServiceInterfaceStrategy should be initialized with a boolean parameter that indicates whether the mapping strategy applies to the request parameters or response parameters.

```
ServiceInterfaceStrategy strat =  new
ServiceInterfaceStrategy(com.example.customerservice.multipart.MultiPartCustomerService.class,
true);
```

```
SoapJaxbDataFormat soapDataFormat = new
SoapJaxbDataFormat("com.example.customerservice.multipart", strat);
```

82.4.1. Holder Object mapping

JAX-WS specifies the use of a type-parameterized **javax.xml.ws.Holder** object for **In**/**Out** and **Out** parameters. You may use an instance of the parameterized-type directly. The camel-soap DataFormat marshals Holder values in accordance with the JAXB mapping for the class of the **Holder's value. No mapping is provided for \'Holder** objects in an unmarshalled response.

82.5. EXAMPLES

82.5.1. Webservice client

The following route supports marshalling the request and unmarshalling a response or fault.

```
String WS_URI = "cxf://http://myserver/customerservice?
serviceClass=com.example.customerservice&dataFormat=RAW";
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
from("direct:customerServiceClient")
  .onException(Exception.class)
    .handled(true)
    .unmarshal(soapDF)
  .end()
  .marshal(soapDF)
  .to(WS_URI)
  .unmarshal(soapDF);
```

The below snippet creates a proxy for the service interface and makes a SOAP call to the above route.

```
import org.apache.camel.Endpoint;
import org.apache.camel.component.bean.ProxyHelper;
...

Endpoint startEndpoint = context.getEndpoint("direct:customerServiceClient");
ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
// CustomerService below is the service endpoint interface, *not* the javax.xml.ws.Service subclass
CustomerService proxy = ProxyHelper.createProxy(startEndpoint, classLoader,
CustomerService.class);
GetCustomersByNameResponse response = proxy.getCustomersByName(new
GetCustomersByName());
```

82.5.2. Webservice Server

Using the following route sets up a webservice server that listens on jms queue customerServiceQueue and processes requests using the class CustomerServiceImpl. The customerServiceImpl of course should implement the interface CustomerService. Instead of directly instantiating the server class it could be defined in a spring context as a regular bean.

```
SoapJaxbDataFormat soapDF = new SoapJaxbDataFormat("com.example.customerservice", new
ServiceInterfaceStrategy(CustomerService.class));
CustomerService serverBean = new CustomerServiceImpl();
```

```
from("jms://queue:customerServiceQueue")
  .onException(Exception.class)
    .handled(true)
    .marshal(soapDF)
  .end()
  .unmarshal(soapDF)
  .bean(serverBean)
  .marshal(soapDF);
```

82.6. DEPENDENCIES

To use the SOAP dataformat in your camel routes you need to add the following dependency to your pom.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-soap</artifactId>
  <version>{CamelSBVersion}</version>
</dependency>
```

82.7. SPRING BOOT AUTO-CONFIGURATION

When using soapjaxb with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-soap-starter</artifactId>
</dependency>
```

The component supports 7 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.soapjaxb.context-path	Package name where your JAXB classes are located.		String

Name	Description	Default	Type
camel.dataformat.soapjaxb.element-name-strategy-ref	Refers to an element strategy to lookup from the registry. An element name strategy is used for two purposes. The first is to find a xml element name for a given object and soap action when marshaling the object into a SOAP message. The second is to find an Exception class for a given soap fault name. The following three element strategy class name is provided out of the box. QNameStrategy - Uses a fixed qName that is configured on instantiation. Exception lookup is not supported TypeNameStrategy - Uses the name and namespace from the XMLType annotation of the given type. If no namespace is set then package-info is used. Exception lookup is not supported ServiceInterfaceStrategy - Uses information from a webservice interface to determine the type name and to find the exception class for a SOAP fault All three classes is located in the package name org.apache.camel.dataformat.soap.name If you have generated the web service stub code with cxf-codegen or a similar tool then you probably will want to use the ServiceInterfaceStrategy. In the case you have no annotated service interface you should use QNameStrategy or TypeNameStrategy.		String
camel.dataformat.soapjaxb.enabled	Whether to enable auto configuration of the soapjaxb data format. This is enabled by default.		Boolean
camel.dataformat.soapjaxb.encoding	To overrule and use a specific encoding.		String
camel.dataformat.soapjaxb.namespace-prefix-ref	When marshalling using JAXB or SOAP then the JAXB implementation will automatic assign namespace prefixes, such as ns2, ns3, ns4 etc. To control this mapping, Camel allows you to refer to a map which contains the desired mapping.		String
camel.dataformat.soapjaxb.schema	To validate against an existing schema. Your can use the prefix classpath:, file: or http: to specify how the resource should by resolved. You can separate multiple schema files by using the ',' character.		String
camel.dataformat.soapjaxb.version	SOAP version should either be 1.1 or 1.2. Is by default 1.1.	1.1	String

CHAPTER 83. ZIP FILE

The Zip File Data Format is a message compression and de-compression format. Messages can be marshalled (compressed) to Zip files containing a single entry, and Zip files containing a single entry can be unmarshalled (decompressed) to the original file contents. This data format supports ZIP64, as long as Java 7 or later is being used].

83.1. ZIPFILE OPTIONS

The Zip File dataformat supports 4 options, which are listed below.

Name	Default	Java Type	Description
usingIterator		**Boolean**	If the zip file has more then one entry, the setting this option to true, allows to work with the splitter EIP, to split the data using an iterator in a streaming mode.
allowEmptyDirectory		**Boolean**	If the zip file has more then one entry, setting this option to true, allows to get the iterator even if the directory is empty.
preservePathElements		**Boolean**	If the file name contains path elements, setting this option to true, allows the path to be maintained in the zip file.
maxDecompressedSize		**Integer**	Set the maximum decompressed size of a zip file (in bytes). The default value if not specified corresponds to 1 gigabyte. An IOException will be thrown if the decompressed size exceeds this amount. Set to -1 to disable setting a maximum decompressed size.

83.2. MARSHAL

In this example we marshal a regular text/XML payload to a compressed payload using Zip file compression, and send it to an ActiveMQ queue called MY_QUEUE.

```
from("direct:start")
    .marshal().zipFile()
    .to("activemq:queue:MY_QUEUE");
```

The name of the Zip entry inside the created Zip file is based on the incoming **CamelFileName** message header, which is the standard message header used by the file component. Additionally, the outgoing **CamelFileName** message header is automatically set to the value of the incoming **CamelFileName** message header, with the ".zip" suffix. So for example, if the following route finds a file named "test.txt" in the input directory, the output will be a Zip file named "test.txt.zip" containing a single Zip entry named "test.txt":

```
from("file:input/directory?antInclude=*/.txt")
    .marshal().zipFile()
    .to("file:output/directory");
```

If there is no incoming **CamelFileName** message header (for example, if the file component is not the

consumer), then the message ID is used by default, and since the message ID is normally a unique generated ID, you will end up with filenames like **ID-MACHINENAME-2443-1211718892437-1-0.zip**. If you want to override this behavior, then you can set the value of the **CamelFileName** header explicitly in your route:

```
from("direct:start")
    .setHeader(Exchange.FILE_NAME, constant("report.txt"))
    .marshal().zipFile()
    .to("file:output/directory");
```

This route would result in a Zip file named "report.txt.zip" in the output directory, containing a single Zip entry named "report.txt".

83.3. UNMARSHAL

In this example we unmarshal a Zip file payload from an ActiveMQ queue called MY_QUEUE to its original format, and forward it for processing to the **UnZippedMessageProcessor**.

```
from("activemq:queue:MY_QUEUE")
    .unmarshal().zipFile()
    .process(new UnZippedMessageProcessor());
```

If the zip file has more then one entry, the usingIterator option of ZipFileDataFormat to be true, and you can use splitter to do the further work.

```
ZipFileDataFormat zipFile = new ZipFileDataFormat();
zipFile.setUsingIterator(true);

from("file:src/test/resources/org/apache/camel/dataformat/zipfile/?delay=1000&noop=true")
    .unmarshal(zipFile)
    .split(body(Iterator.class)).streaming()
        .process(new UnZippedMessageProcessor())
    .end();
```

Or you can use the ZipSplitter as an expression for splitter directly like this

```
from("file:src/test/resources/org/apache/camel/dataformat/zipfile?delay=1000&noop=true")
    .split(new ZipSplitter()).streaming()
        .process(new UnZippedMessageProcessor())
    .end();
```

83.3.1. Aggregate

NOTE

This aggregation strategy requires eager completion check to work properly.

In this example we aggregate all text files found in the input directory into a single Zip file that is stored in the output directory.

```
from("file:input/directory?antInclude=*/.txt")
    .aggregate(constant(true), new ZipAggregationStrategy())
```

```
        .completionFromBatchConsumer().eagerCheckCompletion()
        .to("file:output/directory");
```

The outgoing **CamelFileName** message header is created using java.io.File.createTempFile, with the ".zip" suffix. If you want to override this behavior, then you can set the value of the **CamelFileName** header explicitly in your route:

```
from("file:input/directory?antInclude=*/.txt")
    .aggregate(constant(true), new ZipAggregationStrategy())
        .completionFromBatchConsumer().eagerCheckCompletion()
        .setHeader(Exchange.FILE_NAME, constant("reports.zip"))
        .to("file:output/directory");
```

83.4. DEPENDENCIES

To use Zip files in your camel routes you need to add a dependency on **camel-zipfile** which implements this data format.

If you use Maven you can just add the following to your **pom.xml**, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-zipfile</artifactId>
  <version>{CamelSBVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

83.5. SPRING BOOT AUTO-CONFIGURATION

When using zipfile with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-zipfile-starter</artifactId>
</dependency>
```

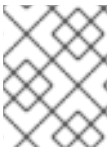
The component supports 5 options, which are listed below.

Name	Description	Default	Type
camel.dataformat.zipfile.allow-empty-directory	If the zip file has more then one entry, setting this option to true, allows to get the iterator even if the directory is empty.	false	Boolean
camel.dataformat.zipfile.enabled	Whether to enable auto configuration of the zipfile data format. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.dataformat.zipfile.max-decompressed-size	Set the maximum decompressed size of a zip file (in bytes). The default value if not specified corresponds to 1 gigabyte. An IOException will be thrown if the decompressed size exceeds this amount. Set to -1 to disable setting a maximum decompressed size.	1073741824	Long
camel.dataformat.zipfile.preserve-path-elements	If the file name contains path elements, setting this option to true, allows the path to be maintained in the zip file.	false	Boolean
camel.dataformat.zipfile.using-iterator	If the zip file has more then one entry, the setting this option to true, allows to work with the splitter EIP, to split the data using an iterator in a streaming mode.	false	Boolean

CHAPTER 84. CONSTANT

The Constant Expression Language is really just a way to use a constant value or object.



NOTE

This is a fixed constant value (or object) that is only set once during starting up the route, do not use this if you want dynamic values during routing.

84.1. CONSTANT OPTIONS

The Constant language supports 2 options, which are listed below.

Name	Default	Java Type	Description
resultType		**String**	Sets the class name of the constant type.
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

84.2. EXAMPLE

The **setHeader** EIP can utilize a constant expression like:

```
<route>
  <from uri="seda:a"/>
  <setHeader name="theHeader">
    <constant>the value</constant>
  </setHeader>
  <to uri="mock:b"/>
</route>
```

in this case, the message coming from the seda:a endpoint will have the header with key **theHeader** set its value as **the value** (string type).

And the same example using Java DSL:

```
from("seda:a")
  .setHeader("theHeader", constant("the value"))
  .to("mock:b");
```

84.2.1. Specifying type of value

The option **resultType** can be used to specify the type of the value, when the value is given as a **String** value, which happens when using XML or YAML DSL:

For example to set a header with **int** type you can do:

```
<route>
  <from uri="seda:a"/>
```

```
  <setHeader name="zipCode">
    <constant resultType="int">90210</constant>
  </setHeader>
  <to uri="mock:b"/>
</route>
```

84.3. LOADING CONSTANT FROM EXTERNAL RESOURCE

You can externalize the constant and have Camel load it from a resource such as **"classpath:"**, **"file:"**, or **"http:"**.
This is done using the following syntax: **"resource:scheme:location"**, eg to refer to a file on the classpath you can do:

```
.setHeader("myHeader").constant("resource:classpath:constant.txt")
```

84.4. DEPENDENCIES

The Constant language is part of **camel-core**.

84.5. SPRING BOOT AUTO-CONFIGURATION

When using constant with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map

Name	Description	Default	Type
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map

Name	Description	Default	Type
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String

Name	Description	Default	Type
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean

Name	Description	Default	Type
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer

Name	Description	Default	Type
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer

Name	Description	Default	Type
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer

Name	Description	Default	Type
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String

Name	Description	Default	Type
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer

Name	Description	Default	Type
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String

Name	Description	Default	Type
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map

Name	Description	Default	Type
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String

Name	Description	Default	Type
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String

Name	Description	Default	Type
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 85. CSIMPLE

The CSimple language is **compiled** Simple language.

85.1. DIFFERENT BETWEEN CSIMPLE AND SIMPLE

The simple language is a dynamic expression language which is runtime parsed into a set of Camel Expressions or Predicates.

The csimple language is parsed into regular Java source code and compiled together with all the other source code, or compiled once during bootstrap via the **camel-csimple-joor** module.

The simple language is generally very lightweight and fast, however for some use-cases with dynamic method calls via OGNL paths, then the simple language does runtime introspection and reflection calls. This has an overhead on performance, and was one of the reasons why csimple was created.

The csimple language requires to be typesafe and method calls via OGNL paths requires to know the type during parsing. This means for csimple languages expressions you would need to provide the class type in the script, whereas simple introspects this at runtime.

In other words the simple language is using *duck typing* (if it looks like a duck, and quacks like a duck, then it is a duck) and csimple is using Java type (typesafety). If there is a type error then simple will report this at runtime, and with csimple there will be a Java compilation error.

85.1.1. Additional CSimple functions

The csimple language includes some additional functions to support common use-cases working with **Collection**, **Map** or array types. The following functions *bodyAsIndex*, *headerAsIndex*, and *exchangePropertyAsIndex* is used for these use-cases as they are typed.

Function	Type	Description
bodyAsIndex(*type*, *index*)	Type	To be used for collecting the body from an existing **Collection**, **Map** or array (lookup by the index) and then converting the body to the given type determined by its classname. The converted body can be null.
mandatoryBodyAsIndex(*type*, *index*)	Type	To be used for collecting the body from an existing **Collection**, **Map** or array (lookup by the index) and then converting the body to the given type determined by its classname. Expects the body to be not null.
headerAsIndex(*key*, *type*, *index*)	Type	To be used for collecting a header from an existing **Collection**, **Map** or array (lookup by the index) and then converting the header value to the given type determined by its classname. The converted header can be null.
mandatoryHeaderAsIndex(*key*, *type*, *index*)	Type	To be used for collecting a header from an existing **Collection**, **Map** or array (lookup by the index) and then converting the header value to the given type determined by its classname. Expects the header to be not null.

Function	Type	Description
exchangePropertyAsIndex(*key*, *type*, *index*)	Type	To be used for collecting an exchange property from an existing **Collection**, **Map** or array (lookup by the index) and then converting the exchange property to the given type determined by its classname. The converted exchange property can be null.
mandatoryExchangePropertyAsIndex(*key*, *type*, *index*)	Type	To be used for collecting an exchange property from an existing **Collection**, **Map** or array (lookup by the index) and then converting the exchange property to the given type determined by its classname. Expects the exchange property to be not null.

For example given the following simple expression:

```
Hello $\{body[0].name}
```

This script has no type information, and the simple language will resolve this at runtime, by introspecting the message body and if it's a collection based then lookup the first element, and then invoke a method named **getName** via reflection.

In csimple (compiled) we want to pre compile this and therefore the end user must provide type information with the *bodyAsIndex* function:

```
Hello $\{bodyAsIndex(com.foo.MyUser, 0).name}
```

85.2. COMPILATION

The csimple language is parsed into regular Java source code and compiled together with all the other source code, or it can be compiled once during bootstrap via the **camel-csimple-joor** module.

There are two ways to compile csimple

- using the **camel-csimple-maven-plugin** generating source code at built time.
- using **camel-csimple-joor** which does runtime in-memory compilation during bootstrap of Camel.

85.2.1. Using camel-csimple-maven-plugin

The **camel-csimple-maven-plugin** Maven plugin is used for discovering all the csimple scripts from the source code, and then automatic generate source code in the **src/generated/java** folder, which then gets compiled together with all the other sources.

The maven plugin will do source code scanning of **.java** and **.xml** files (Java and XML DSL). The scanner limits to detect certain code patterns, and it may miss discovering some csimple scripts if they are being used in unusual/rare ways.

The runtime compilation using **camel-csimple-joor** does not have this limitation.

The benefit is all the csimple scripts will be compiled using the regular Java compiler and therefore everything is included out of the box as **.class** files in the application JAR file, and no additional dependencies is required at runtime.

To use **camel-csimple-maven-plugin** you need to add it to your **pom.xml** file as shown:

```
<plugins>
  <!-- generate source code for csimple languages -->
  <plugin>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-csimple-maven-plugin</artifactId>
    <version>${camel.version}</version>
    <executions>
      <execution>
        <id>generate</id>
        <goals>
          <goal>generate</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  <!-- include source code generated to maven sources paths -->
  <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>3.1.0</version>
    <executions>
      <execution>
        <phase>generate-sources</phase>
        <goals>
          <goal>add-source</goal>
          <goal>add-resource</goal>
        </goals>
        <configuration>
          <sources>
            <source>src/generated/java</source>
          </sources>
          <resources>
            <resource>
              <directory>src/generated/resources</directory>
            </resource>
          </resources>
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
```

And then you must also add the **build-helper-maven-plugin** Maven plugin to include **src/generated** to the list of source folders for the Java compiler, to ensure the generated source code is compiled and included in the application JAR file.

See the **camel-example-csimple** example at Camel Examples which uses the maven plugin.

85.2.2. Using camel-csimple-joor

The jOOR library integrates with the Java compiler and performs runtime compilation of Java code.

The supported runtime when using **camel-simple-joor** is intended for Java standalone, Spring Boot, Camel Quarkus and other microservices runtimes. It is not supported in OSGi, Camel Karaf or any kind of Java Application Server runtime.

jOOR does not support runtime compilation with Spring Boot using *fat jar* packaging (https://github.com/jOOQ/jOOR/issues/69), it works with exploded classpath.

To use **camel-simple-joor** you simply just add it as dependency to the classpath:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-csimple-joor</artifactId>
  <version>{CamelSBProjectVersion}</version>
</dependency>
```

There is no need for adding Maven plugins to the **pom.xml** file.

See the **camel-example-csimple-joor** example at Camel Examples which uses the jOOR compiler.

85.3. CSIMPLE LANGUAGE OPTIONS

The CSimple language supports 2 options, which are listed below.

Name	Default	Java Type	Description
resultType		**String**	Sets the class name of the result type (type from output).
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

85.4. LIMITATIONS

Currently, the csimple language does **not** support:

- nested functions (aka functions inside functions)
- the *null safe* operator (**?**).

For example the following scripts cannot compile:

```
Hello ${bean:greeter(${body}, ${header.counter})}
```

```
${bodyAs(MyUser)?.address?.zip} > 10000
```

85.5. AUTO IMPORTS

The csimple language will automatically import from:

```
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.*;
import org.apache.camel.*;
import org.apache.camel.util.*;
```

85.6. CONFIGURATION FILE

You can configure the csimple language in the **camel-csimple.properties** file which is loaded from the root classpath.

For example you can add additional imports in the **camel-csimple.properties** file by adding:

```
import com.foo.MyUser;
import com.bar.*;
import static com.foo.MyHelper.*;
```

You can also add aliases (key=value) where an alias will be used as a shorthand replacement in the code.

```
echo()=${bodyAs(String)} ${bodyAs(String)}
```

Which allows to use *echo()* in the csimple language script such as:

```
from("direct:hello")
    .transform(csimple("Hello echo()"))
    .log("You said ${body}");
```

The *echo()* alias will be replaced with its value resulting in a script as:

```
    .transform(csimple("Hello ${bodyAs(String)} ${bodyAs(String)}"))
```

85.7. SEE ALSO

See the Simple language as **csimple** has the same set of functions as simple language.

85.8. SPRING BOOT AUTO-CONFIGURATION

When using csimple with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long

Name	Description	Default	Type
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String

Name	Description	Default	Type
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map

Name	Description	Default	Type
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer

Name	Description	Default	Type
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer

Name	Description	Default	Type
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String

Name	Description	Default	Type
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer

Name	Description	Default	Type
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float

Name	Description	Default	Type
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean

Name	Description	Default	Type
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map

Name	Description	Default	Type
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String

Name	Description	Default	Type
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 86. EXCHANGEPROPERTY

The ExchangeProperty Expression Language allows you to extract values of named exchange properties.

86.1. EXCHANGE PROPERTY OPTIONS

The ExchangeProperty language supports 1 options, which are listed below.

Name	Default	Java Type	Description
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

86.2. EXAMPLE

The **recipientList** EIP can utilize a exchangeProperty like:

```
<route>
  <from uri="direct:a" />
  <recipientList>
    <exchangeProperty>myProperty</exchangeProperty>
  </recipientList>
</route>
```

In this case, the list of recipients are contained in the property 'myProperty'.

And the same example in Java DSL:

```
from("direct:a").recipientList(exchangeProperty("myProperty"));
```

86.3. DEPENDENCIES

The ExchangeProperty language is part of **camel-core**.

86.4. SPRING BOOT AUTO-CONFIGURATION

When using exchangeProperty with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String

Name	Description	Default	Type
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String

Name	Description	Default	Type
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map

Name	Description	Default	Type
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer

Name	Description	Default	Type
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer

Name	Description	Default	Type
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer

Name	Description	Default	Type
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float

Name	Description	Default	Type
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean

Name	Description	Default	Type
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map

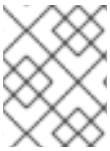
Name	Description	Default	Type
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String

Name	Description	Default	Type
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 87. FILE

The File Expression Language is an extension to the language, adding file related capabilities. These capabilities are related to common use cases working with file path and names. The goal is to allow expressions to be used with the

components for setting dynamic file patterns for both consumer and producer.



NOTE

The file language is merged with language which means you can use all the file syntax directly within the simple language.

87.1. FILE LANGUAGE OPTIONS

The File language supports 2 options, which are listed below.

Name	Default	Java Type	Description
resultType		**String**	Sets the class name of the result type (type from output).
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

87.2. SYNTAX

This language is an **extension** to the language so the syntax applies also. So the table below only lists the additional file related functions.

All the file tokens use the same expression name as the method on the **java.io.File** object, for instance **file:absolute** refers to the **java.io.File.getAbsolute()** method. Notice that not all expressions are supported by the current Exchange. For instance the component supports some options, whereas the File component supports all of them.

Expression	Type	File Consumer	File Producer	FTP Consumer	FTP Producer	Description
file:name	String	yes	no	yes	no	refers to the file name (is relative to the starting directory, see note below)
file:name.ext	String	yes	no	yes	no	refers to the file extension only

Expression	Type	File Consumer	File Producer	FTP Consumer	FTP Producer	Description
file:name.ext.single	String	yes	no	yes	no	refers to the file extension. If the file extension has multiple dots, then this expression strips and only returns the last part.
file:name.noext	String	yes	no	yes	no	refers to the file name with no extension (is relative to the starting directory, see note below)
file:name.noext.single	String	yes	no	yes	no	refers to the file name with no extension (is relative to the starting directory, see note below). If the file extension has multiple dots, then this expression strips only the last part, and keep the others.
file:onlyname	String	yes	no	yes	no	refers to the file name only with no leading paths.
file:onlyname.noext	String	yes	no	yes	no	refers to the file name only with no extension and with no leading paths.
file:onlyname.noext.single	String	yes	no	yes	no	refers to the file name only with no extension and with no leading paths. If the file extension has multiple dots, then this expression strips only the last part, and keep the others.
file:ext	String	yes	no	yes	no	refers to the file extension only

Expression	Type	File Consumer	File Producer	FTP Consumer	FTP Producer	Description
file:parent	String	yes	no	yes	no	refers to the file parent
file:path	String	yes	no	yes	no	refers to the file path
file:absolute	Boolean	yes	no	no	no	refers to whether the file is regarded as absolute or relative
file:absolute.path	String	yes	no	no	no	refers to the absolute file path
file:length	Long	yes	no	yes	no	refers to the file length returned as a Long type
file:size	Long	yes	no	yes	no	refers to the file length returned as a Long type
file:modified	Date	yes	no	yes	no	Refers to the file last modified returned as a Date type
date:_command:pattern_	String	yes	yes	yes	yes	for date formatting using the **java.text.SimpleDateFormat** patterns. Is an **extension** to the language. Additional command is: **file** (consumers only) for the last modified timestamp of the file. Notice: all the commands from the language can also be used.

87.3. FILE TOKEN EXAMPLE

87.3.1. Relative paths

We have a **java.io.File** handle for the file **hello.txt** in the following **relative** directory: **.\filelanguage\test**. And we configure our endpoint to use this starting directory **.\filelanguage**. The file tokens will return as:

Expression	Returns
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt
file:parent	filelanguage\test
file:path	filelanguage\test\hello.txt
file:absolute	false
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

87.3.2. Absolute paths

We have a **java.io.File** handle for the file **hello.txt** in the following **absolute** directory: **\workspace\camel\camel-core\target\filelanguage\test**. And we configure out endpoint to use the absolute starting directory **\workspace\camel\camel-core\target\filelanguage**. The file tokens will return as:

Expression	Returns
file:name	test\hello.txt
file:name.ext	txt
file:name.noext	test\hello
file:onlyname	hello.txt
file:onlyname.noext	hello
file:ext	txt
file:parent	\workspace\camel\camel-core\target\filelanguage\test

Expression	Returns
file:path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt
file:absolute	true
file:absolute.path	\workspace\camel\camel-core\target\filelanguage\test\hello.txt

87.4. SAMPLES

You can enter a fixed file name such as **myfile.txt**:

```
fileName="myfile.txt"
```

Let's assume we use the file consumer to read files and want to move the read files to back up folder with the current date as a sub folder. This can be done using an expression like:

```
fileName="backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

relative folder names are also supported so suppose the backup folder should be a sibling folder then you can append **..** as shown:

```
fileName="../backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

As this is an extension to the language we have access to all the goodies from this language also, so in this use case we want to use the in.header.type as a parameter in the dynamic expression:

```
fileName="../backup/${date:now:yyyyMMdd}/type-${in.header.type}/backup-of-${file:name.noext}.bak"
```

If you have a custom date you want to use in the expression then Camel supports retrieving dates from the message header:

```
fileName="orders/order-${in.header.customerId}-${date:in.header.orderDate:yyyyMMdd}.xml"
```

And finally we can also use a bean expression to invoke a POJO class that generates some String output (or convertible to String) to be used:

```
fileName="uniquefile-${bean:myguidgenerator.generateid}.txt"
```

Of course all this can be combined in one expression where you can use the and the language in one combined expression. This is pretty powerful for those common file path patterns.

87.5. DEPENDENCIES

The File language is part of **camel-core**.

87.6. SPRING BOOT AUTO-CONFIGURATION

When using file with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String

Name	Description	Default	Type
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.username	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String

Name	Description	Default	Type
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean

Name	Description	Default	Type
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean

Name	Description	Default	Type
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer

Name	Description	Default	Type
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String

Name	Description	Default	Type
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String

Name	Description	Default	Type
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String

Name	Description	Default	Type
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map

Name	Description	Default	Type
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String

Name	Description	Default	Type
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 88. HEADER

The Header Expression Language allows you to extract values of named headers.

88.1. HEADER OPTIONS

The Header language supports 1 options, which are listed below.

Name	Default	Java Type	Description
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

88.2. EXAMPLE USAGE

The **recipientList** EIP can utilize a header:

```
<route>
  <from uri="direct:a" />
  <recipientList>
    <header>myHeader</header>
  </recipientList>
</route>
```

In this case, the list of recipients are contained in the header 'myHeader'.

And the same example in Java DSL:

```
from("direct:a").recipientList(header("myHeader"));
```

88.3. DEPENDENCIES

The Header language is part of **camel-core**.

88.4. SPRING BOOT AUTO-CONFIGURATION

When using header with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String

Name	Description	Default	Type
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String

Name	Description	Default	Type
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer

Name	Description	Default	Type
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean

Name	Description	Default	Type
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean

Name	Description	Default	Type
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer

Name	Description	Default	Type
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float

Name	Description	Default	Type
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer

Name	Description	Default	Type
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String

Name	Description	Default	Type
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map

Name	Description	Default	Type
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String

Name	Description	Default	Type
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 89. JSONPATH

Camel supports JSONPath to allow using Expression or Predicate on JSON messages.

89.1. JSONPATH OPTIONS

The JSONPath language supports 8 options, which are listed below.

Name	Default	Java Type	Description
resultType		**String**	Sets the class name of the result type (type from output).
suppressExceptions		**Boolean**	Whether to suppress exceptions such as PathNotFoundException.
allowSimple		**Boolean**	Whether to allow in inlined Simple exceptions in the JSONPath expression.
allowEasyPredicate		**Boolean**	Whether to allow using the easy predicate parser to pre-parse predicates.
writeAsString		**Boolean**	Whether to write the output of each row/element as a JSON String value instead of a Map/POJO value.
headerName		**String**	Name of header to use as input, instead of the message body.
option		**Enum**	To configure additional options on JSONPath. Multiple values can be separated by comma. Enum values: • DEFAULT_PATH_LEAF_TO_NULL • ALWAYS_RETURN_LIST • AS_PATH_LIST • SUPPRESS_EXCEPTIONS • REQUIRE_PROPERTIES
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

89.2. EXAMPLES

For example, you can use JSONPath in a Predicate with the Content Based Router EIP .

```
from("queue:books.new")
  .choice()
```

```
    .when().jsonpath("$.store.book[?(@.price < 10)]")
      .to("jms:queue:book.cheap")
    .when().jsonpath("$.store.book[?(@.price < 30)]")
      .to("jms:queue:book.average")
    .otherwise()
      .to("jms:queue:book.expensive");
```

And in XML DSL:

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath>$.store.book[?(@.price &lt; 10)]</jsonpath>
      <to uri="mock:cheap"/>
    </when>
    <when>
      <jsonpath>$.store.book[?(@.price &lt; 30)]</jsonpath>
      <to uri="mock:average"/>
    </when>
    <otherwise>
      <to uri="mock:expensive"/>
    </otherwise>
  </choice>
</route>
```

89.3. JSONPATH SYNTAX

Using the JSONPath syntax takes some time to learn, even for basic predicates. So for example to find out all the cheap books you have to do:

```
$.store.book[?(@.price < 20)]
```

89.3.1. Easy JSONPath Syntax

However, what if you could just write it as:

```
store.book.price < 20
```

And you can omit the path if you just want to look at nodes with a price key:

```
price < 20
```

To support this there is a **EasyPredicateParser** which kicks-in if you have defined the predicate using a basic style. That means the predicate must not start with the **$** sign, and only include one operator.

The easy syntax is:

```
left OP right
```

You can use Camel simple language in the right operator, eg:

```
store.book.price < ${header.limit}
```

See the JSONPath project page for more syntax examples.

89.4. SUPPORTED MESSAGE BODY TYPES

Camel JSonPath supports message body using the following types:

Type	Comment
File	Reading from files
String	Plain strings
Map	Message bodies as **java.util.Map** types
List	Message bodies as **java.util.List** types
POJO	**Optional** If Jackson is on the classpath, then camel-jsonpath is able to use Jackson to read the message body as POJO and convert to **java.util.Map** which is supported by JSonPath. For example, you can add **camel-jackson** as dependency to include Jackson.
InputStream	If none of the above types matches, then Camel will attempt to read the message body as a **java.io.InputStream**.

If a message body is of unsupported type then an exception is thrown by default, however you can configure JSonPath to suppress exceptions (see below)

89.5. SUPPRESSING EXCEPTIONS

By default, jsonpath will throw an exception if the json payload does not have a valid path accordingly to the configured jsonpath expression. In some use-cases you may want to ignore this in case the json payload contains optional data. Therefore, you can set the option **suppressExceptions** to **true** to ignore this as shown:

```
from("direct:start")
    .choice()
        // use true to suppress exceptions
        .when().jsonpath("person.middlename", true)
            .to("mock:middle")
        .otherwise()
            .to("mock:other");
```

And in XML DSL:

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
```

```
      <jsonpath suppressExceptions="true">person.middlename</jsonpath>
      <to uri="mock:middle"/>
    </when>
    <otherwise>
      <to uri="mock:other"/>
    </otherwise>
  </choice>
</route>
```

This option is also available on the **@JsonPath** annotation.

89.6. INLINE SIMPLE EXPRESSIONS

It's possible to inlined Simple language in the JSONPath expression using the simple syntax **${xxx}**.

An example is shown below:

```
from("direct:start")
  .choice()
    .when().jsonpath("$.store.book[?(@.price < ${header.cheap})]")
      .to("mock:cheap")
    .when().jsonpath("$.store.book[?(@.price < ${header.average})]")
      .to("mock:average")
    .otherwise()
      .to("mock:expensive");
```

And in XML DSL:

```
<route>
  <from uri="direct:start"/>
  <choice>
    <when>
      <jsonpath>$.store.book[?(@.price &lt; ${header.cheap})]</jsonpath>
      <to uri="mock:cheap"/>
    </when>
    <when>
      <jsonpath>$.store.book[?(@.price &lt; ${header.average})]</jsonpath>
      <to uri="mock:average"/>
    </when>
    <otherwise>
      <to uri="mock:expensive"/>
    </otherwise>
  </choice>
</route>
```

You can turn off support for inlined Simple expression by setting the option **allowSimple** to **false** as shown:

```
.when().jsonpath("$.store.book[?(@.price < 10)]", false, false)
```

And in XML DSL:

```
<jsonpath allowSimple="false">$.store.book[?(@.price &lt; 10)]</jsonpath>
```

89.7. JSONPATH INJECTION

You can use Bean Integration to invoke a method on a bean and use various languages such as JSONPath (via the **@JsonPath** annotation) to extract a value from the message and bind it to a method parameter, as shown below:

```
public class Foo {

    @Consume("activemq:queue:books.new")
    public void doSomething(@JsonPath("$.store.book[*].author") String author, @Body String json) {
      // process the inbound message here
    }
}
```

89.8. ENCODING DETECTION

The encoding of the JSON document is detected automatically, if the document is encoded in unicode (UTF-8, UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE) as specified in RFC-4627. If the encoding is a non-unicode encoding, you can either make sure that you enter the document in String format to JSONPath, or you can specify the encoding in the header **CamelJsonPathJsonEncoding** which is defined as a constant in: **JsonpathConstants.HEADER_JSON_ENCODING**.

89.9. SPLIT JSON DATA INTO SUB ROWS AS JSON

You can use JSONPath to split a JSON document, such as:

```
from("direct:start")
    .split().jsonpath("$.store.book[*]")
    .to("log:book");
```

Then each book is logged, however the message body is a **Map** instance. Sometimes you may want to output this as plain String JSON value instead, which can be done with the **writeAsString** option as shown:

```
from("direct:start")
    .split().jsonpathWriteAsString("$.store.book[*]")
    .to("log:book");
```

Then each book is logged as a String JSON value.

89.10. USING HEADER AS INPUT

By default, JSONPath uses the message body as the input source. However, you can also use a header as input by specifying the **headerName** option.

For example to count the number of books from a JSON document that was stored in a header named **books** you can do:

```
from("direct:start")
    .setHeader("numberOfBooks")
        .jsonpath("$..store.book.length()", false, int.class, "books")
    .to("mock:result");
```

In the **jsonpath** expression above we specify the name of the header as **books**, and we also told that we wanted the result to be converted as an integer by **int.class**.

The same example in XML DSL would be:

```
<route>
  <from uri="direct:start"/>
  <setHeader name="numberOfBooks">
    <jsonpath headerName="books" resultType="int">$..store.book.length()</jsonpath>
  </setHeader>
  <to uri="mock:result"/>
</route>
```

89.11. SPRING BOOT AUTO-CONFIGURATION

When using jsonpath with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-jsonpath-starter</artifactId>
</dependency>
```

The component supports 8 options, which are listed below.

Name	Description	Default	Type
camel.language.jsonpath.allow-easy-predicate	Whether to allow using the easy predicate parser to pre-parse predicates.	true	Boolean
camel.language.jsonpath.allow-simple	Whether to allow in inlined Simple exceptions in the JSONPath expression.	true	Boolean
camel.language.jsonpath.enabled	Whether to enable auto configuration of the jsonpath language. This is enabled by default.		Boolean
camel.language.jsonpath.header-name	Name of header to use as input, instead of the message body.		String
camel.language.jsonpath.option	To configure additional options on JSONPath. Multiple values can be separated by comma.		String
camel.language.jsonpath.suppress-exceptions	Whether to suppress exceptions such as PathNotFoundException.	false	Boolean

Name	Description	Default	Type
camel.language.jsonpath.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.jsonpath.write-as-string	Whether to write the output of each row/element as a JSON String value instead of a Map/POJO value.	false	Boolean

CHAPTER 90. REF

The Ref Expression Language is really just a way to lookup a custom **Expression** or **Predicate** from the Registry.

This is particular useable in XML DSLs.

90.1. REF LANGUAGE OPTIONS

The Ref language supports 1 options, which are listed below.

Name	Default	Java Type	Description
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

90.2. EXAMPLE USAGE

The Splitter EIP in XML DSL can utilize a custom expression using **<ref>** like:

```
<bean id="myExpression" class="com.mycompany.MyCustomExpression"/>

<route>
  <from uri="seda:a"/>
  <split>
    <ref>myExpression</ref>
    <to uri="mock:b"/>
  </split>
</route>
```

in this case, the message coming from the seda:a endpoint will be splitted using a custom **Expression** which has the id **myExpression** in the Registry.

And the same example using Java DSL:

```
from("seda:a").split().ref("myExpression").to("seda:b");
```

90.3. DEPENDENCIES

The Ref language is part of **camel-core**.

90.4. SPRING BOOT AUTO-CONFIGURATION

When using ref with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long

Name	Description	Default	Type
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String

Name	Description	Default	Type
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer

Name	Description	Default	Type
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean

Name	Description	Default	Type
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean

Name	Description	Default	Type
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer

Name	Description	Default	Type
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean

Name	Description	Default	Type
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float

Name	Description	Default	Type
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer

Name	Description	Default	Type
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String

Name	Description	Default	Type
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map

Name	Description	Default	Type
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String

Name	Description	Default	Type
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 91. XQUERY

Camel supports XQuery to allow an Expression or Predicate to be used in the DSL.

For example, you could use XQuery to create a predicate in a Message Filter or as an expression for a Recipient List.

91.1. XQUERY LANGUAGE OPTIONS

The XQuery language supports 4 options, which are listed below.

Name	Default	Java Type	Description
type		**String**	Sets the class name of the result type (type from output) The default result type is NodeSet.
headerName		**String**	Name of header to use as input, instead of the message body.
configurationRef		**String**	Reference to a saxon configuration instance in the registry to use for xquery (requires camel-saxon). This may be needed to add custom functions to a saxon configuration, so these custom functions can be used in xquery expressions.
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

91.2. VARIABLES

The message body will be set as the **contextItem**. And the following variables are available as well:

Variable	Type	Description
exchange	Exchange	The current Exchange
in.body	Object	The message body
out.body	Object	**deprecated** The OUT message body (if any)
in.headers.*	Object	You can access the value of exchange.in.headers with key **foo** by using the variable which name is in.headers.foo
out.headers.*	Object	**deprecated** You can access the value of exchange.out.headers with key**foo** by using the variable which name is out.headers.foo variable

Variable	Type	Description
key name	Object	Any exchange.properties and exchange.in.headers and any additional parameters set using **setParameters(Map)**. These parameters are added with they own key name, for instance if there is an IN header with the key name **foo** then its added as **foo**.

91.3. EXAMPLE

```
from("queue:foo")
  .filter().xquery("//foo")
  .to("queue:bar")
```

You can also use functions inside your query, in which case you need an explicit type conversion, or you will get an **org.w3c.dom.DOMException: HIERARCHY_REQUEST_ERR**). You need to pass in the expected output type of the function. For example the concat function returns a **String** which is done as shown:

```
from("direct:start")
  .recipientList().xquery("concat('mock:foo.', /person/@city)", String.class);
```

And in XML DSL:

```
<route>
  <from uri="direct:start"/>
  <recipientList>
    <xquery type="java.lang.String">concat('mock:foo.', /person/@city</xquery>
  </recipientList>
</route>
```

91.3.1. Using namespaces

If you have a standard set of namespaces you wish to work with and wish to share them across many XQuery expressions you can use the **org.apache.camel.support.builder.Namespaces** when using Java DSL as shown:

```
Namespaces ns = new Namespaces("c", "http://acme.com/cheese");

from("direct:start")
  .filter().xquery("/c:person[@name='James']", ns)
  .to("mock:result");
```

Notice how the namespaces are provided to **xquery** with the **ns** variable that are passed in as the 2nd parameter.

Each namespace is a key=value pair, where the prefix is the key. In the XQuery expression then the namespace is used by its prefix, eg:

```
/c:person[@name='James']
```

The namespace builder supports adding multiple namespaces as shown:

```
Namespaces ns = new Namespaces("c", "http://acme.com/cheese")
                    .add("w", "http://acme.com/wine")
                    .add("b", "http://acme.com/beer");
```

When using namespaces in XML DSL then its different, as you setup the namespaces in the XML root tag (or one of the **camelContext**, **routes**, **route** tags).

In the XML example below we use Spring XML where the namespace is declared in the root tag **beans**, in the line with **xmlns:foo="http://example.com/person"**:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:foo="http://example.com/person"
       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">

  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
      <from uri="activemq:MyQueue"/>
      <filter>
        <xquery>/foo:person[@name='James']</xquery>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

This namespace uses **foo** as prefix, so the **<xquery>** expression uses **/foo:** to use this namespace.

91.4. USING XQUERY AS TRANSFORMATION

We can do a message translation using transform or setBody in the route, as shown below:

```
from("direct:start").
    transform().xquery("/people/person");
```

Notice that xquery will use DOMResult by default, so if we want to grab the value of the person node, using **text()** we need to tell XQuery to use String as result type, as shown:

```
from("direct:start").
    transform().xquery("/people/person/text()", String.class);
```

If you want to use Camel variables like headers, you have to explicitly declare them in the XQuery expression.

```
<transform>
    <xquery>
        declare variable $in.headers.foo external;
```

```
        element item {$in.headers.foo}
    </xquery>
</transform>
```

91.5. LOADING SCRIPT FROM EXTERNAL RESOURCE

You can externalize the script and have Camel load it from a resource such as **"classpath:"**, **"file:"**, or **"http:"**. This is done using the following syntax: **"resource:scheme:location"**, e.g. to refer to a file on the classpath you can do:

```
.setHeader("myHeader").xquery("resource:classpath:myxquery.txt", String.class)
```

91.6. LEARNING XQUERY

XQuery is a very powerful language for querying, searching, sorting and returning XML. For help learning XQuery try these tutorials

- Mike Kay's XQuery Primer
- The W3Schools XQuery Tutorial

91.7. DEPENDENCIES

To use XQuery in your camel routes you need to add the a dependency on **camel-saxon** which implements the XQuery language.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-saxon</artifactId>
  <version>{CamelSBVersion}</version>
</dependency>
```

91.8. SPRING BOOT AUTO-CONFIGURATION

When using xquery with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-saxon-starter</artifactId>
</dependency>
```

The component supports 11 options, which are listed below.

Name	Description	Default	Type
camel.component.xquery.autowired-enabled	Whether autowiring is enabled. This is used for automatic autowiring options (the option must be marked as autowired) by looking up in the registry to find if there is a single instance of matching type, which then gets configured on the component. This can be used for automatic configuring JDBC data sources, JMS connection factories, AWS Clients, etc.	true	Boolean
camel.component.xquery.bridge-error-handler	Allows for bridging the consumer to the Camel routing Error Handler, which mean any exceptions occurred while the consumer is trying to pickup incoming messages, or the likes, will now be processed as a message and handled by the routing Error Handler. By default the consumer will use the org.apache.camel.spi.ExceptionHandler to deal with exceptions, that will be logged at WARN or ERROR level and ignored.	false	Boolean
camel.component.xquery.configuration	To use a custom Saxon configuration. The option is a net.sf.saxon.Configuration type.		Configuration
camel.component.xquery.configuration-properties	To set custom Saxon configuration properties.		Map
camel.component.xquery.enabled	Whether to enable auto configuration of the xquery component. This is enabled by default.		Boolean
camel.component.xquery.lazy-start-producer	Whether the producer should be started lazy (on the first message). By starting lazy you can use this to allow CamelContext and routes to startup in situations where a producer may otherwise fail during starting and cause the route to fail being started. By deferring this startup to be lazy then the startup failure can be handled during routing messages via Camel's routing error handlers. Beware that when the first message is processed then creating and starting the producer may take a little time and prolong the total processing time of the processing.	false	Boolean
camel.component.xquery.module-u-r-i-resolver	To use the custom ModuleURIResolver. The option is a net.sf.saxon.lib.ModuleURIResolver type.		ModuleURIResolver

Name	Description	Default	Type
camel.language.xquery.configuration-ref	Reference to a saxon configuration instance in the registry to use for xquery (requires camel-saxon). This may be needed to add custom functions to a saxon configuration, so these custom functions can be used in xquery expressions.		String
camel.language.xquery.enabled	Whether to enable auto configuration of the xquery language. This is enabled by default.		Boolean
camel.language.xquery.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.xquery.type	Sets the class name of the result type (type from output) The default result type is NodeSet.		String

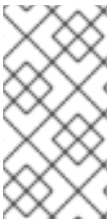
CHAPTER 92. SIMPLE

The Simple Expression Language was a really simple language when it was created, but has since grown more powerful. It is primarily intended for being a very small and simple language for evaluating **Expression** or **Predicate** without requiring any new dependencies or knowledge of other scripting languages such as Groovy.

The simple language is designed with intend to cover almost all the common use cases when little need for scripting in your Camel routes.

However, for much more complex use cases then a more powerful language is recommended such as:

- Groovy
- MVEL
- OGNL



NOTE

The simple language requires **camel-bean** JAR as classpath dependency if the simple language uses OGNL expressions, such as calling a method named **myMethod** on the message body: **${body.myMethod()}**. At runtime the simple language will then us its built-in OGNL support which requires the **camel-bean** component.

The simple language uses **${body}** placeholders for complex expressions or functions.

NOTE

See also the CSimple language which is compiled.

NOTE

Alternative syntax
You can also use the alternative syntax which uses **$simple{ }** as placeholders. This can be used in situations to avoid clashes when using for example Spring property placeholder together with Camel.

92.1. SIMPLE LANGUAGE OPTIONS

The Simple language supports 2 options, which are listed below.

Name	Default	Java Type	Description
resultType		**String**	Sets the class name of the result type (type from output).
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

92.2. VARIABLES

Variable	Type	Description
camelId	String	the CamelContext name
camelContext.**OGNL**	Object	the CamelContext invoked using a Camel OGNL expression.
exchange	Exchange	the Exchange
exchange.**OGNL**	Object	the Exchange invoked using a Camel OGNL expression.
exchangeId	String	the exchange id
id	String	the message id
messageTimestamp	String	the message timestamp (millis since epoc) that this message originates from. Some systems like JMS, Kafka, AWS have a timestamp on the event/message, that Camel received. This method returns the timestamp, if a timestamp exists. The message timestamp and exchange created are not the same. An exchange always have a created timestamp which is the local timestamp when Camel created the exchange. The message timestamp is only available in some Camel components when the consumer is able to extract the timestamp from the source event. If the message has no timestamp then 0 is returned.
body	Object	the body
body.**OGNL**	Object	the body invoked using a Camel OGNL expression.
bodyAs(*type*)	Type	Converts the body to the given type determined by its classname. The converted body can be null.
bodyAs(*type*).**OGNL**	Object	Converts the body to the given type determined by its classname and then invoke methods using a Camel OGNL expression. The converted body can be null.
bodyOneLine	String	Converts the body to a String and removes all line-breaks so the string is in one line.
mandatoryBodyAs(*type*)	Type	Converts the body to the given type determined by its classname, and expects the body to be not null.
mandatoryBodyAs(*type*).**OGNL**	Object	Converts the body to the given type determined by its classname and then invoke methods using a Camel OGNL expression.
header.foo	Object	refer to the foo header

Variable	Type	Description
header[foo]	Object	refer to the foo header
headers.foo	Object	refer to the foo header
headers:foo	Object	refer to the foo header
headers[foo]	Object	refer to the foo header
header.foo[bar]	Object	regard foo header as a map and perform lookup on the map with bar as key
header.foo.**OGNL**	Object	refer to the foo header and invoke its value using a Camel OGNL expression.
headerAs(*key*,*type*)	Type	converts the header to the given type determined by its classname
headers	Map	refer to the headers
exchangeProperty.foo	Object	refer to the foo property on the exchange
exchangeProperty[foo]	Object	refer to the foo property on the exchange
exchangeProperty.foo.**OGNL**	Object	refer to the foo property on the exchange and invoke its value using a Camel OGNL expression.
sys.foo	String	refer to the JVM system property
sysenv.foo	String	refer to the system environment variable
env.foo	String	refer to the system environment variable
exception	Object	refer to the exception object on the exchange, is **null** if no exception set on exchange. Will fallback and grab caught exceptions (**Exchange.EXCEPTION_CAUGHT**) if the Exchange has any.
exception.**OGNL**	Object	refer to the exchange exception invoked using a Camel OGNL expression object
exception.message	String	refer to the exception.message on the exchange, is **null** if no exception set on exchange. Will fallback and grab caught exceptions (**Exchange.EXCEPTION_CAUGHT**) if the Exchange has any.

Variable	Type	Description
exception.stacktrace	String	refer to the exception.stracktrace on the exchange, is **null** if no exception set on exchange. Will fallback and grab caught exceptions (**Exchange.EXCEPTION_CAUGHT**) if the Exchange has any.
date:_command_	Date	evaluates to a Date object. Supported commands are: **now** for current timestamp, **exchangeCreated** for the timestamp when the current exchange was created, **header.xxx** to use the Long/Date object header with the key xxx. **exchangeProperty.xxx** to use the Long/Date object in the exchange property with the key xxx. **file** for the last modified timestamp of the file (available with a File consumer). Command accepts offsets such as: **now-24h** or **header.xxx+1h** or even **now+1h30m-100**.
date:_command:pattern_	String	Date formatting using **java.text.SimpleDateFormat** patterns.
date-with-timezone:_command:timezone:pattern_	String	Date formatting using **java.text.SimpleDateFormat** timezones and patterns.
bean:_bean expression_	Object	Invoking a bean expression using the language. Specifying a method name you must use dot as separator. We also support the ?method=methodname syntax that is used by the component. Camel will by default lookup a bean by the given name. However if you need to refer to a bean class (such as calling a static method) then you can prefix with type, such as **bean:type:fqnClassName**.
properties:key:default	String	Lookup a property with the given key. If the key does not exists or has no value, then an optional default value can be specified.
routeId	String	Returns the id of the current route the Exchange is being routed.
stepId	String	Returns the id of the current step the Exchange is being routed.
threadName	String	Returns the name of the current thread. Can be used for logging purpose.
hostname	String	Returns the local hostname (may be empty if not possible to resolve).
ref:xxx	Object	To lookup a bean from the Registry with the given id.

Variable	Type	Description
type:name.field	Object	To refer to a type or field by its FQN name. To refer to a field you can append .FIELD_NAME. For example, you can refer to the constant field from Exchange as: **org.apache.camel.Exchange.FILE_NAME**
null	null	represents a **null**
random(value)	Integer	returns a random Integer between 0 (included) and *value* (excluded)
random(min,max)	Integer	returns a random Integer between *min* (included) and *max* (excluded)
collate(group)	List	The collate function iterates the message body and groups the data into sub lists of specified size. This can be used with the Splitter EIP to split a message body and group/batch the splitted sub message into a group of N sub lists. This method works similar to the collate method in Groovy.
skip(number)	Iterator	The skip function iterates the message body and skips the first number of items. This can be used with the Splitter EIP to split a message body and skip the first N number of items.
messageHistory	String	The message history of the current exchange how it has been routed. This is similar to the route stack-trace message history the error handler logs in case of an unhandled exception.
messageHistory(false)	String	As messageHistory but without the exchange details (only includes the route stack-trace). This can be used if you do not want to log sensitive data from the message itself.

92.3. OGNL EXPRESSION SUPPORT

When using **OGNL** then **camel-bean** JAR is required to be on the classpath.

Camel's OGNL support is for invoking methods only. You cannot access fields. Camel support accessing the length field of Java arrays.

The Simple and Bean language now supports a Camel OGNL notation for invoking beans in a chain like fashion. Suppose the Message IN body contains a POJO which has a **getAddress()** method.

Then you can use Camel OGNL notation to access the address object:

```
simple("${body.address}")
simple("${body.address.street}")
simple("${body.address.zip}")
```

Camel understands the shorthand names for getters, but you can invoke any method or use the real name such as:

```
simple("${body.address}")
simple("${body.getAddress.getStreet}")
simple("${body.address.getZip}")
simple("${body.doSomething}")
```

You can also use the null safe operator (**?.**) to avoid NPE if for example the body does NOT have an address

```
simple("${body?.address?.street}")
```

It is also possible to index in **Map** or **List** types, so you can do:

```
simple("${body[foo].name}")
```

To assume the body is **Map** based and lookup the value with **foo** as key, and invoke the **getName** method on that value.

If the key has space, then you **must** enclose the key with quotes, for example 'foo bar':

```
simple("${body['foo bar'].name}")
```

You can access the **Map** or **List** objects directly using their key name (with or without dots) :

```
simple("${body[foo]}")
simple("${body[this.is.foo]}")
```

Suppose there was no value with the key **foo** then you can use the null safe operator to avoid the NPE as shown:

```
simple("${body[foo]?.name}")
```

You can also access **List** types, for example to get lines from the address you can do:

```
simple("${body.address.lines[0]}")
simple("${body.address.lines[1]}")
simple("${body.address.lines[2]}")
```

There is a special **last** keyword which can be used to get the last value from a list.

```
simple("${body.address.lines[last]}")
```

And to get the 2nd last you can subtract a number, so we can use **last-1** to indicate this:

```
simple("${body.address.lines[last-1]}")
```

And the 3rd last is of course:

```
simple("${body.address.lines[last-2]}")
```

And you can call the size method on the list with

```
simple("${body.address.lines.size}")
```

Camel supports the length field for Java arrays as well, eg:

```
String[] lines = new String[]{"foo", "bar", "cat"};
exchange.getIn().setBody(lines);

simple("There are ${body.length} lines")
```

And yes you can combine this with the operator support as shown below:

```
simple("${body.address.zip} > 1000")
```

92.4. OPERATOR SUPPORT

The parser is limited to only support a single operator.

To enable it the left value must be enclosed in $\\{ }. The syntax is:

```
${leftValue} OP rightValue
```

Where the **rightValue** can be a String literal enclosed in **' '**, **null**, a constant value or another expression enclosed in **$\{ }**.

NOTE

There **must** be spaces around the operator.

Camel will automatically type convert the rightValue type to the leftValue type, so it is able to eg. convert a string into a numeric, so you can use **>** comparison for numeric values.

The following operators are supported:

Operator	Description
==	equals
=~	equals ignore case (will ignore case when comparing String values)
>	greater than
>=	greater than or equals
<	less than
<=	less than or equals

Operator	Description
!=	not equals
!=~	not equals ignore case (will ignore case when comparing String values)
contains	For testing if contains in a string based value
!contains	For testing if not contains in a string based value
~~	For testing if contains by ignoring case sensitivity in a string based value
!~~	For testing if not contains by ignoring case sensitivity in a string based value
regex	For matching against a given regular expression pattern defined as a String value
!regex	For not matching against a given regular expression pattern defined as a String value
in	For matching if in a set of values, each element must be separated by comma. If you want to include an empty value, then it must be defined using double comma, eg ',,bronze,silver,gold', which is a set of four values with an empty value and then the three medals.
!in	For matching if not in a set of values, each element must be separated by comma. If you want to include an empty value, then it must be defined using double comma, eg ',,bronze,silver,gold', which is a set of four values with an empty value and then the three medals.
is	For matching if the left hand side type is an instance of the value.
!is	For matching if the left hand side type is not an instance of the value.
range	For matching if the left hand side is within a range of values defined as numbers: **from..to**..
!range	For matching if the left hand side is not within a range of values defined as numbers: **from..to**. .
startsWith	For testing if the left hand side string starts with the right hand string.
starts with	Same as the startsWith operator.
endsWith	For testing if the left hand side string ends with the right hand string.
ends with	Same as the endsWith operator.

And the following unary operators can be used:

Operator	Description
++	To increment a number by one. The left hand side must be a function, otherwise parsed as literal.
–	To decrement a number by one. The left hand side must be a function, otherwise parsed as literal.
\n	To use newline character.
\t	To use tab character.
\r	To use carriage return character.
\}	To use the } character as text. This may be needed when building a JSon structure with the simple language.

And the following logical operators can be used to group expressions:

Operator	Description
&&	The logical and operator is used to group two expressions.
	The logical or operator is used to group two expressions.

The syntax for AND is:

```
${leftValue} OP rightValue && ${leftValue} OP rightValue
```

And the syntax for OR is:

```
${leftValue} OP rightValue || ${leftValue} OP rightValue
```

Some examples:

```
// exact equals match
simple("${header.foo} == 'foo'")

// ignore case when comparing, so if the header has value FOO this will match
simple("${header.foo} =~ 'foo'")

// here Camel will type convert '100' into the type of header.bar and if it is an Integer '100' will also be converter to an Integer
simple("${header.bar} == '100'")
```

```
simple("${header.bar} == 100")

// 100 will be converter to the type of header.bar so we can do > comparison
simple("${header.bar} > 100")
```

92.4.1. Comparing with different types

When you compare with different types such as String and int, then you have to take a bit care. Camel will use the type from the left hand side as 1st priority. And fallback to the right hand side type if both values couldn't be compared based on that type.
This means you can flip the values to enforce a specific type. Suppose the bar value above is a String. Then you can flip the equation:

```
simple("100 < ${header.bar}")
```

which then ensures the int type is used as 1st priority.

This may change in the future if the Camel team improves the binary comparison operations to prefer numeric types to String based. It's most often the String type which causes problem when comparing with numbers.

```
// testing for null
simple("${header.baz} == null")

// testing for not null
simple("${header.baz} != null")
```

And a bit more advanced example where the right value is another expression

```
simple("${header.date} == ${date:now:yyyyMMdd}")

simple("${header.type} == ${bean:orderService?method=getOrderType}")
```

And an example with contains, testing if the title contains the word Camel

```
simple("${header.title} contains 'Camel'")
```

And an example with regex, testing if the number header is a 4 digit value:

```
simple("${header.number} regex '\\d{4}'")
```

And finally an example if the header equals any of the values in the list. Each element must be separated by comma, and no space around.
This also works for numbers etc, as Camel will convert each element into the type of the left hand side.

```
simple("${header.type} in 'gold,silver'")
```

And for all the last 3 we also support the negate test using not:

```
simple("${header.type} !in 'gold,silver'")
```

And you can test if the type is a certain instance, eg for instance a String

-

```
simple("${header.type} is 'java.lang.String'")
```

We have added a shorthand for all **java.lang** types so you can write it as:

```
simple("${header.type} is 'String'")
```

Ranges are also supported. The range interval requires numbers and both from and end are inclusive. For instance to test whether a value is between 100 and 199:

```
simple("${header.number} range 100..199")
```

Notice we use **..** in the range without spaces. It is based on the same syntax as Groovy.

```
simple("${header.number} range '100..199'")
```

As the XML DSL does not have all the power as the Java DSL with all its various builder methods, you have to resort to use some other languages for testing with simple operators. Now you can do this with the simple language. In the sample below we want to test if the header is a widget order:

```
<from uri="seda:orders">
  <filter>
    <simple>${header.type} == 'widget'</simple>
    <to uri="bean:orderService?method=handleWidget"/>
  </filter>
</from>
```

92.4.2. Using and / or

If you have two expressions you can combine them with the **&&** or || operator.

For instance:

```
simple("${header.title} contains 'Camel' && ${header.type'} == 'gold'")
```

And of course the || is also supported. The sample would be:

```
simple("${header.title} contains 'Camel' || ${header.type'} == 'gold'")
```

92.5. EXAMPLES

In the XML DSL sample below we filter based on a header value:

```
<from uri="seda:orders">
  <filter>
    <simple>${header.foo}</simple>
    <to uri="mock:fooOrders"/>
  </filter>
</from>
```

The Simple language can be used for the predicate test above in the Message Filter pattern, where we test if the in message has a **foo** header (a header with the key **foo** exists). If the expression evaluates to **true** then the message is routed to the **mock:fooOrders** endpoint, otherwise the message is dropped.

The same example in Java DSL:

```
from("seda:orders")
    .filter().simple("${header.foo}")
        .to("seda:fooOrders");
```

You can also use the simple language for simple text concatenations such as:

```
from("direct:hello")
    .transform().simple("Hello ${header.user} how are you?")
    .to("mock:reply");
```

Notice that we must use $\\{ } placeholders in the expression now to allow Camel to parse it correctly.

And this sample uses the date command to output current date.

```
from("direct:hello")
    .transform().simple("The today is ${date:now:yyyyMMdd} and it is a great day.")
    .to("mock:reply");
```

And in the sample below we invoke the bean language to invoke a method on a bean to be included in the returned string:

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator}")
    .to("mock:reply");
```

Where **orderIdGenerator** is the id of the bean registered in the Registry. If using Spring then it is the Spring bean id.

If we want to declare which method to invoke on the order id generator bean we must prepend **.method name** such as below where we invoke the **generateId** method.

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator.generateId}")
    .to("mock:reply");
```

We can use the **?method=methodname** option that we are familiar with the Bean component itself:

```
from("direct:order")
    .transform().simple("OrderId: ${bean:orderIdGenerator?method=generateId}")
    .to("mock:reply");
```

You can also convert the body to a given type, for example to ensure that it is a String you can do:

```
<transform>
  <simple>Hello ${bodyAs(String)} how are you?</simple>
</transform>
```

There are a few types which have a shorthand notation, so we can use **String** instead of **java.lang.String**. These are: **byte[], String, Integer, Long**. All other types must use their FQN name, e.g. **org.w3c.dom.Document**.

It is also possible to lookup a value from a header **Map**:

```
<transform>
  <simple>The gold value is ${header.type[gold]}</simple>
</transform>
```

In the code above we lookup the header with name **type** and regard it as a **java.util.Map** and we then lookup with the key **gold** and return the value. If the header is not convertible to Map an exception is thrown. If the header with name **type** does not exist **null** is returned.

You can nest functions, such as shown below:

```
<setHeader name="myHeader">
  <simple>${properties:${header.someKey}}</simple>
</setHeader>
```

92.6. SETTING RESULT TYPE

You can now provide a result type to the Simple expression, which means the result of the evaluation will be converted to the desired type. This is most usable to define types such as booleans, integers, etc.

For example to set a header as a boolean type you can do:

```
.setHeader("cool", simple("true", Boolean.class))
```

And in XML DSL

```
<setHeader name="cool">
  <!-- use resultType to indicate that the type should be a java.lang.Boolean -->
  <simple resultType="java.lang.Boolean">true</simple>
</setHeader>
```

92.7. USING NEW LINES OR TABS IN XML DSLS

It is easier to specify new lines or tabs in XML DSLs as you can escape the value now

```
<transform>
  <simple>The following text\nis on a new line</simple>
</transform>
```

92.8. LEADING AND TRAILING WHITESPACE HANDLING

The trim attribute of the expression can be used to control whether the leading and trailing whitespace characters are removed or preserved. The default value is true, which removes the whitespace characters.

```
<setBody>
  <simple trim="false">You get some trailing whitespace characters.     </simple>
</setBody>
```

92.9. LOADING SCRIPT FROM EXTERNAL RESOURCE

You can externalize the script and have Camel load it from a resource such as **"classpath:"**, **"file:"**, or **"http:"**. This is done using the following syntax: **"resource:scheme:location"**, e.g. to refer to a file on the classpath you can do:

```
.setHeader("myHeader").simple("resource:classpath:mysimple.txt")
```

92.10. SPRING BOOT AUTO-CONFIGURATION

When using simple with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean

Name	Description	Default	Type
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String

Name	Description	Default	Type
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean

Name	Description	Default	Type
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean

Name	Description	Default	Type
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer

Name	Description	Default	Type
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer

Name	Description	Default	Type
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String

Name	Description	Default	Type
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer

Name	Description	Default	Type
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String

Name	Description	Default	Type
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map

Name	Description	Default	Type
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String

Name	Description	Default	Type
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String

Name	Description	Default	Type
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 93. TOKENIZE

The tokenizer language is a built-in language in **camel-core**, which is most often used with the Split EIP to split a message using a token-based strategy.

The tokenizer language is intended to tokenize text documents using a specified delimiter pattern. It can also be used to tokenize XML documents with some limited capability. For a truly XML-aware tokenization, the use of the XML Tokenize language is recommended as it offers a faster, more efficient tokenization specifically for XML documents.

93.1. TOKENIZE OPTIONS

The Tokenize language supports 11 options, which are listed below.

Name	Default	Java Type	Description
token		**String**	**Required** The (start) token to use as tokenizer, for example you can use the new line token. You can use simple language as the token to support dynamic tokens.
endToken		**String**	The end token to use as tokenizer if using start/end token pairs. You can use simple language as the token to support dynamic tokens.
inheritNamespaceTagName		**String**	To inherit namespaces from a root/parent tag name when using XML You can use simple language as the tag name to support dynamic names.
headerName		**String**	Name of header to tokenize instead of using the message body.
regex		**Boolean**	If the token is a regular expression pattern. The default value is false.
xml		**Boolean**	Whether the input is XML messages. This option must be set to true if working with XML payloads.
includeTokens		**Boolean**	Whether to include the tokens in the parts when using pairs The default value is false.
group		**String**	To group N parts together, for example to split big files into chunks of 1000 lines. You can use simple language as the group to support dynamic group sizes.
groupDelimiter		**String**	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.
skipFirst		**Boolean**	To skip the very first element.

Name	Default	Java Type	Description
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

93.2. EXAMPLE

The following example shows how to take a request from the direct:a endpoint then split it into pieces using an Expression, then forward each piece to direct:b:

```
<route>
  <from uri="direct:a"/>
  <split>
    <tokenize token="\n"/>
    <to uri="direct:b"/>
  </split>
</route>
```

And in Java DSL:

```
from("direct:a")
    .split(body().tokenize("\n"))
        .to("direct:b");
```

93.3. SEE ALSO

For more examples see Split EIP.

93.4. SPRING BOOT AUTO-CONFIGURATION

When using tokenize with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-core-starter</artifactId>
</dependency>
```

The component supports 147 options, which are listed below.

Name	Description	Default	Type
camel.cloud.consul.service-discovery.acl-token	Sets the ACL token to be used with Consul.		String

Name	Description	Default	Type
camel.cloud.consul.service-discovery.block-seconds	The seconds to wait for a watch event, default 10 seconds.	10	Integer
camel.cloud.consul.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.consul.service-discovery.connect-timeout-millis	Connect timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.datacenter	The data center.		String
camel.cloud.consul.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.consul.service-discovery.password	Sets the password to be used for basic authentication.		String
camel.cloud.consul.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.consul.service-discovery.read-timeout-millis	Read timeout for OkHttpClient.		Long
camel.cloud.consul.service-discovery.url	The Consul agent URL.		String
camel.cloud.consul.service-discovery.user-name	Sets the username to be used for basic authentication.		String

Name	Description	Default	Type
camel.cloud.consul.service-discovery.write-timeout-millis	Write timeout for OkHttpClient.		Long
camel.cloud.dns.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.dns.service-discovery.domain	The domain name;.		String
camel.cloud.dns.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.dns.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.dns.service-discovery.proto	The transport protocol of the desired service.	_tcp	String
camel.cloud.etcd.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.etcd.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.etcd.service-discovery.password	The password to use for basic authentication.		String
camel.cloud.etcd.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map

Name	Description	Default	Type
camel.cloud.etcd.service-discovery.service-path	The path to look for for service discovery.	/services/	String
camel.cloud.etcd.service-discovery.timeout	To set the maximum time an action could take to complete.		Long
camel.cloud.etcd.service-discovery.type	To set the discovery type, valid values are on-demand and watch.	on-demand	String
camel.cloud.etcd.service-discovery.uris	The URIs the client can connect to.		String
camel.cloud.etcd.service-discovery.user-name	The user name to use for basic authentication.		String
camel.cloud.kubernetes.service-discovery.api-version	Sets the API version when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-data	Sets the Certificate Authority data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.ca-cert-file	Sets the Certificate Authority data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-data	Sets the Client Certificate data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-cert-file	Sets the Client Certificate data that are loaded from the file when using client lookup.		String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.client-key-algo	Sets the Client Keystore algorithm, such as RSA when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-data	Sets the Client Keystore data when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-file	Sets the Client Keystore data that are loaded from the file when using client lookup.		String
camel.cloud.kubernetes.service-discovery.client-key-passphrase	Sets the Client Keystore passphrase when using client lookup.		String
camel.cloud.kubernetes.service-discovery.configurations	Define additional configuration definitions.		Map
camel.cloud.kubernetes.service-discovery.dns-domain	Sets the DNS domain to use for DNS lookup.		String
camel.cloud.kubernetes.service-discovery.enabled	Enable the component.	true	Boolean
camel.cloud.kubernetes.service-discovery.lookup	How to perform service lookup. Possible values: client, dns, environment. When using client, then the client queries the kubernetes master to obtain a list of active pods that provides the service, and then random (or round robin) select a pod. When using dns the service name is resolved as name.namespace.svc.dnsDomain. When using dnssrv the service name is resolved with SRV query for .... svc... When using environment then environment variables are used to lookup the service. By default environment is used.	environment	String

Name	Description	Default	Type
camel.cloud.kubernetes.service-discovery.master-url	Sets the URL to the master when using client lookup.		String
camel.cloud.kubernetes.service-discovery.namespace	Sets the namespace to use. Will by default use namespace from the ENV variable KUBERNETES_MASTER.		String
camel.cloud.kubernetes.service-discovery.oauth-token	Sets the OAUTH token for authentication (instead of username/password) when using client lookup.		String
camel.cloud.kubernetes.service-discovery.password	Sets the password for authentication when using client lookup.		String
camel.cloud.kubernetes.service-discovery.port-name	Sets the Port Name to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.port-protocol	Sets the Port Protocol to use for DNS/DNSSRV lookup.		String
camel.cloud.kubernetes.service-discovery.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.kubernetes.service-discovery.trust-certs	Sets whether to turn on trust certificate check when using client lookup.	false	Boolean
camel.cloud.kubernetes.service-discovery.username	Sets the username for authentication when using client lookup.		String

Name	Description	Default	Type
camel.cloud.ribbon.load-balancer.client-name	Sets the Ribbon client name.		String
camel.cloud.ribbon.load-balancer.configurations	Define additional configuration definitions.		Map
camel.cloud.ribbon.load-balancer.enabled	Enable the component.	true	Boolean
camel.cloud.ribbon.load-balancer.namespace	The namespace.		String
camel.cloud.ribbon.load-balancer.password	The password.		String
camel.cloud.ribbon.load-balancer.properties	Set client properties to use. These properties are specific to what service call implementation are in use. For example if using ribbon, then the client properties are define in com.netflix.client.config.CommonClientConfigKey.		Map
camel.cloud.ribbon.load-balancer.username	The username.		String
camel.hystrix.allow-maximum-size-to-diverge-from-core-size	Allows the configuration for maximumSize to take effect. That value can then be equal to, or higher, than coreSize.	false	Boolean
camel.hystrix.circuit-breaker-enabled	Whether to use a HystrixCircuitBreaker or not. If false no circuit-breaker logic will be used and all requests permitted. This is similar in effect to circuitBreakerForceClosed() except that continues tracking metrics and knowing whether it should be open/closed, this property results in not even instantiating a circuit-breaker.	true	Boolean

Name	Description	Default	Type
camel.hystrix.circuit-breaker-error-threshold-percentage	Error percentage threshold (as whole number such as 50) at which point the circuit breaker will trip open and reject requests. It will stay tripped for the duration defined in circuitBreakerSleepWindowInMilliseconds; The error percentage this is compared against comes from HystrixCommandMetrics.getHealthCounts().	50	Integer
camel.hystrix.circuit-breaker-force-closed	If true the HystrixCircuitBreaker#allowRequest() will always return true to allow requests regardless of the error percentage from HystrixCommandMetrics.getHealthCounts(). The circuitBreakerForceOpen() property takes precedence so if it set to true this property does nothing.	false	Boolean
camel.hystrix.circuit-breaker-force-open	If true the HystrixCircuitBreaker.allowRequest() will always return false, causing the circuit to be open (tripped) and reject all requests. This property takes precedence over circuitBreakerForceClosed();.	false	Boolean
camel.hystrix.circuit-breaker-request-volume-threshold	Minimum number of requests in the metricsRollingStatisticalWindowInMilliseconds() that must exist before the HystrixCircuitBreaker will trip. If below this number the circuit will not trip regardless of error percentage.	20	Integer
camel.hystrix.circuit-breaker-sleep-window-in-milliseconds	The time in milliseconds after a HystrixCircuitBreaker trips open that it should wait before trying requests again.	5000	Integer
camel.hystrix.configurations	Define additional configuration definitions.		Map
camel.hystrix.core-pool-size	Core thread-pool size that gets passed to java.util.concurrent.ThreadPoolExecutor#setCorePoolSize(int).	10	Integer
camel.hystrix.enabled	Enable the component.	true	Boolean
camel.hystrix.execution-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.run(). Requests beyond the concurrent limit will be rejected. Applicable only when executionIsolationStrategy == SEMAPHORE.	20	Integer

Name	Description	Default	Type
camel.hystrix.execution-isolation-strategy	What isolation strategy HystrixCommand.run() will be executed with. If THREAD then it will be executed on a separate thread and concurrent requests limited by the number of threads in the thread-pool. If SEMAPHORE then it will be executed on the calling thread and concurrent requests limited by the semaphore count.	THREAD	String
camel.hystrix.execution-isolation-thread-interrupt-on-timeout	Whether the execution thread should attempt an interrupt (using Future#cancel) when a thread times out. Applicable only when executionIsolationStrategy() == THREAD.	true	Boolean
camel.hystrix.execution-timeout-enabled	Whether the timeout mechanism is enabled for this command.	true	Boolean
camel.hystrix.execution-timeout-in-milliseconds	Time in milliseconds at which point the command will timeout and halt execution. If executionIsolationThreadInterruptOnTimeout == true and the command is thread-isolated, the executing thread will be interrupted. If the command is semaphore-isolated and a HystrixObservableCommand, that command will get unsubscribed.	1000	Integer
camel.hystrix.fallback-enabled	Whether HystrixCommand.getFallback() should be attempted when failure occurs.	true	Boolean
camel.hystrix.fallback-isolation-semaphore-max-concurrent-requests	Number of concurrent requests permitted to HystrixCommand.getFallback(). Requests beyond the concurrent limit will fail-fast and not attempt retrieving a fallback.	10	Integer
camel.hystrix.group-key	Sets the group key to use. The default value is CamelHystrix.	Camel Hystrix	String
camel.hystrix.keep-alive-time	Keep-alive time in minutes that gets passed to ThreadPoolExecutor#setKeepAliveTime(long,TimeUnit).	1	Integer

Name	Description	Default	Type
camel.hystrix.max-queue-size	Max queue size that gets passed to BlockingQueue in HystrixConcurrencyStrategy.getBlockingQueue(int) This should only affect the instantiation of a threadpool - it is not eliglible to change a queue size on the fly. For that, use queueSizeRejectionThreshold().	-1	Integer
camel.hystrix.maximum-size	Maximum thread-pool size that gets passed to ThreadPoolExecutor#setMaximumPoolSize(int) . This is the maximum amount of concurrency that can be supported without starting to reject HystrixCommands. Please note that this setting only takes effect if you also set allowMaximumSizeToDivergeFromCoreSize.	10	Integer
camel.hystrix.metrics-health-snapshot-interval-in-milliseconds	Time in milliseconds to wait between allowing health snapshots to be taken that calculate success and error percentages and affect HystrixCircuitBreaker.isOpen() status. On high-volume circuits the continual calculation of error percentage can become CPU intensive thus this controls how often it is calculated.	500	Integer
camel.hystrix.metrics-rolling-percentile-bucket-size	Maximum number of values stored in each bucket of the rolling percentile. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10	Integer
camel.hystrix.metrics-rolling-percentile-enabled	Whether percentile metrics should be captured using HystrixRollingPercentile inside HystrixCommandMetrics.	true	Boolean
camel.hystrix.metrics-rolling-percentile-window-buckets	Number of buckets the rolling percentile window is broken into. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	6	Integer
camel.hystrix.metrics-rolling-percentile-window-in-milliseconds	Duration of percentile rolling window in milliseconds. This is passed into HystrixRollingPercentile inside HystrixCommandMetrics.	10000	Integer
camel.hystrix.metrics-rolling-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside HystrixCommandMetrics.	10	Integer

Name	Description	Default	Type
camel.hystrix.metrics-rolling-statistical-window-in-milliseconds	This property sets the duration of the statistical rolling window, in milliseconds. This is how long metrics are kept for the thread pool. The window is divided into buckets and rolls by those increments.	10000	Integer
camel.hystrix.queue-size-rejection-threshold	Queue size rejection threshold is an artificial max size at which rejections will occur even if maxQueueSize has not been reached. This is done because the maxQueueSize of a BlockingQueue can not be dynamically changed and we want to support dynamically changing the queue size that affects rejections. This is used by HystrixCommand when queuing a thread for execution.	5	Integer
camel.hystrix.request-log-enabled	Whether HystrixCommand execution and events should be logged to HystrixRequestLog.	true	Boolean
camel.hystrix.thread-pool-key	Sets the thread pool key to use. Will by default use the same value as groupKey has been configured to use.	Camel Hystrix	String
camel.hystrix.thread-pool-rolling-number-statistical-window-buckets	Number of buckets the rolling statistical window is broken into. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10	Integer
camel.hystrix.thread-pool-rolling-number-statistical-window-in-milliseconds	Duration of statistical rolling window in milliseconds. This is passed into HystrixRollingNumber inside each HystrixThreadPoolMetrics instance.	10000	Integer
camel.language.constant.enabled	Whether to enable auto configuration of the constant language. This is enabled by default.		Boolean
camel.language.constant.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.csimple.enabled	Whether to enable auto configuration of the csimple language. This is enabled by default.		Boolean
camel.language.csimple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

Name	Description	Default	Type
camel.language.exchangeproperty.enabled	Whether to enable auto configuration of the exchangeProperty language. This is enabled by default.		Boolean
camel.language.exchangeproperty.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.file.enabled	Whether to enable auto configuration of the file language. This is enabled by default.		Boolean
camel.language.file.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.header.enabled	Whether to enable auto configuration of the header language. This is enabled by default.		Boolean
camel.language.header.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.ref.enabled	Whether to enable auto configuration of the ref language. This is enabled by default.		Boolean
camel.language.ref.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.simple.enabled	Whether to enable auto configuration of the simple language. This is enabled by default.		Boolean
camel.language.simple.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.language.tokenize.enabled	Whether to enable auto configuration of the tokenize language. This is enabled by default.		Boolean
camel.language.tokenize.group-delimiter	Sets the delimiter to use when grouping. If this has not been set then token will be used as the delimiter.		String
camel.language.tokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean
camel.resilience4j.automatic-transition-from-open-to-half-open-enabled	Enables automatic transition from OPEN to HALF_OPEN state once the waitDurationInOpenState has passed.	false	Boolean

Name	Description	Default	Type
camel.resilience4j.circuit-breaker-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreaker instance to lookup and use from the registry. When using this, then any other circuit breaker options are not in use.		String
camel.resilience4j.config-ref	Refers to an existing io.github.resilience4j.circuitbreaker.CircuitBreakerConfig instance to lookup and use from the registry.		String
camel.resilience4j.configurations	Define additional configuration definitions.		Map
camel.resilience4j.enabled	Enable the component.	true	Boolean
camel.resilience4j.failure-rate-threshold	Configures the failure rate threshold in percentage. If the failure rate is equal or greater than the threshold the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 50 percentage.		Float
camel.resilience4j.minimum-number-of-calls	Configures the minimum number of calls which are required (per sliding window period) before the CircuitBreaker can calculate the error rate. For example, if minimumNumberOfCalls is 10, then at least 10 calls must be recorded, before the failure rate can be calculated. If only 9 calls have been recorded the CircuitBreaker will not transition to open even if all 9 calls have failed. Default minimumNumberOfCalls is 100.	100	Integer
camel.resilience4j.permitted-number-of-calls-in-half-open-state	Configures the number of permitted calls when the CircuitBreaker is half open. The size must be greater than 0. Default size is 10.	10	Integer

Name	Description	Default	Type
camel.resilience4j.sliding-window-size	Configures the size of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. slidingWindowSize configures the size of the sliding window. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. The slidingWindowSize must be greater than 0. The minimumNumberOfCalls must be greater than 0. If the slidingWindowType is COUNT_BASED, the minimumNumberOfCalls cannot be greater than slidingWindowSize . If the slidingWindowType is TIME_BASED, you can pick whatever you want. Default slidingWindowSize is 100.	100	Integer
camel.resilience4j.sliding-window-type	Configures the type of the sliding window which is used to record the outcome of calls when the CircuitBreaker is closed. Sliding window can either be count-based or time-based. If slidingWindowType is COUNT_BASED, the last slidingWindowSize calls are recorded and aggregated. If slidingWindowType is TIME_BASED, the calls of the last slidingWindowSize seconds are recorded and aggregated. Default slidingWindowType is COUNT_BASED.	COUNT_BASED	String
camel.resilience4j.slow-call-duration-threshold	Configures the duration threshold (seconds) above which calls are considered as slow and increase the slow calls percentage. Default value is 60 seconds.	60	Integer
camel.resilience4j.slow-call-rate-threshold	Configures a threshold in percentage. The CircuitBreaker considers a call as slow when the call duration is greater than slowCallDurationThreshold Duration. When the percentage of slow calls is equal or greater the threshold, the CircuitBreaker transitions to open and starts short-circuiting calls. The threshold must be greater than 0 and not greater than 100. Default value is 100 percentage which means that all recorded calls must be slower than slowCallDurationThreshold.		Float
camel.resilience4j.wait-duration-in-open-state	Configures the wait duration (in seconds) which specifies how long the CircuitBreaker should stay open, before it switches to half open. Default value is 60 seconds.	60	Integer

Name	Description	Default	Type
camel.resilience4j.writable-stack-trace-enabled	Enables writable stack traces. When set to false, Exception.getStackTrace returns a zero length array. This may be used to reduce log spam when the circuit breaker is open as the cause of the exceptions is already known (the circuit breaker is short-circuiting calls).	true	Boolean
camel.rest.api-component	The name of the Camel component to use as the REST API (such as swagger) If no API Component has been explicit configured, then Camel will lookup if there is a Camel component responsible for servicing and generating the REST API documentation, or if a org.apache.camel.spi.RestApiProcessorFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.api-context-path	Sets a leading API context-path the REST API services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path.		String
camel.rest.api-context-route-id	Sets the route id to use for the route that services the REST API. The route will by default use an auto assigned route id.		String
camel.rest.api-host	To use an specific hostname for the API documentation (eg swagger) This can be used to override the generated host with this configured hostname.		String
camel.rest.api-property	Allows to configure as many additional properties for the api documentation (swagger). For example set property api.title to my cool stuff.		Map
camel.rest.api-vendor-extension	Whether vendor extension is enabled in the Rest APIs. If enabled then Camel will include additional information as vendor extension (eg keys starting with x-) such as route ids, class names etc. Not all 3rd party API gateways and tools supports vendor-extensions when importing your API docs.	false	Boolean
camel.rest.binding-mode	Sets the binding mode to use. The default value is off.		RestBindingMode

Name	Description	Default	Type
camel.rest.client-request-validation	Whether to enable validation of the client request to check whether the Content-Type and Accept headers from the client is supported by the Rest-DSL configuration of its consumes/produces settings. This can be turned on, to enable this check. In case of validation error, then HTTP Status codes 415 or 406 is returned. The default value is false.	false	Boolean
camel.rest.component	The Camel Rest component to use for the REST transport (consumer), such as netty-http, jetty, servlet, undertow. If no component has been explicit configured, then Camel will lookup if there is a Camel component that integrates with the Rest DSL, or if a org.apache.camel.spi.RestConsumerFactory is registered in the registry. If either one is found, then that is being used.		String
camel.rest.component-property	Allows to configure as many additional properties for the rest component in use.		Map
camel.rest.consumer-property	Allows to configure as many additional properties for the rest consumer in use.		Map
camel.rest.context-path	Sets a leading context-path the REST services will be using. This can be used when using components such as camel-servlet where the deployed web application is deployed using a context-path. Or for components such as camel-jetty or camel-netty-http that includes a HTTP server.		String
camel.rest.cors-headers	Allows to configure custom CORS headers.		Map
camel.rest.data-format-property	Allows to configure as many additional properties for the data formats in use. For example set property prettyPrint to true to have json outputted in pretty mode. The properties can be prefixed to denote the option is only for either JSON or XML and for either the IN or the OUT. The prefixes are: json.in. json.out. xml.in. xml.out. For example a key with value xml.out.mustBeJAXBElement is only for the XML data format for the outgoing. A key without a prefix is a common key for all situations.		Map
camel.rest.enable-cors	Whether to enable CORS headers in the HTTP response. The default value is false.	false	Boolean

Name	Description	Default	Type
camel.rest.endpoint-property	Allows to configure as many additional properties for the rest endpoint in use.		Map
camel.rest.host	The hostname to use for exposing the REST service.		String
camel.rest.host-name-resolver	If no hostname has been explicit configured, then this resolver is used to compute the hostname the REST service will be using.		RestHostNameResolver
camel.rest.json-data-format	Name of specific json data format to use. By default json-jackson will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.port	The port number to use for exposing the REST service. Notice if you use servlet component then the port number configured here does not apply, as the port number in use is the actual port number the servlet component is using. eg if using Apache Tomcat its the tomcat http port, if using Apache Karaf its the HTTP service in Karaf that uses port 8181 by default etc. Though in those situations setting the port number here, allows tooling and JMX to know the port number, so its recommended to set the port number to the number that the servlet engine uses.		String
camel.rest.producer-api-doc	Sets the location of the api document (swagger api) the REST producer will use to validate the REST uri and query parameters are valid accordingly to the api document. This requires adding camel-swagger-java to the classpath, and any miss configuration will let Camel fail on startup and report the error(s). The location of the api document is loaded from classpath by default, but you can use file: or http: to refer to resources to load from file or http url.		String
camel.rest.producer-component	Sets the name of the Camel component to use as the REST producer.		String
camel.rest.scheme	The scheme to use for exposing the REST service. Usually http or https is supported. The default value is http.		String
camel.rest.skip-binding-on-error-code	Whether to skip binding on output if there is a custom HTTP error code header. This allows to build custom error messages that do not bind to json / xml etc, as success messages otherwise will do.	false	Boolean

Name	Description	Default	Type
camel.rest.use-x-forward-headers	Whether to use X-Forward headers for Host and related setting. The default value is true.	true	Boolean
camel.rest.xml-data-format	Name of specific XML data format to use. By default jaxb will be used. Important: This option is only for setting a custom name of the data format, not to refer to an existing data format instance.		String
camel.rest.api-context-id-pattern	**Deprecated** Sets an CamelContext id pattern to only allow Rest APIs from rest services within CamelContext's which name matches the pattern. The pattern name refers to the CamelContext name, to match on the current CamelContext only. For any other value, the pattern uses the rules from PatternHelper#matchPattern(String,String).		String
camel.rest.api-context-listing	**Deprecated** Sets whether listing of all available CamelContext's with REST services in the JVM is enabled. If enabled it allows to discover these contexts, if false then only the current CamelContext is in use.	false	Boolean

CHAPTER 94. XML TOKENIZE

The XML Tokenize language is a built-in language in **camel-xml-jaxp**, which is a truly XML-aware tokenizer that can be used with the Split EIP as the conventional Tokenize to efficiently and effectively tokenize XML documents..

XML Tokenize is capable of not only recognizing XML namespaces and hierarchical structures of the document but also more efficiently tokenizing XML documents than the conventional Tokenize language.

Additional dependency

In order to use this component, an additional dependency is required as follows:

```
<dependency>
  <groupId>org.codehaus.woodstox</groupId>
  <artifactId>woodstox-core-asl</artifactId>
  <version>4.4.1</version>
</dependency>
```

or

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-stax-starter</artifactId>
</dependency>
```

94.1. XML TOKENIZER OPTIONS

The XML Tokenize language supports 4 options, which are listed below.

Name	Default	Java Type	Description
headerName		**String**	Name of header to tokenize instead of using the message body.
mode		**Enum**	The extraction mode. The available extraction modes are: i - injecting the contextual namespace bindings into the extracted token (default) w - wrapping the extracted token in its ancestor context u - unwrapping the extracted token to its child content t - extracting the text content of the specified element. Enum values: • i • w • u • t

Name	Default	Java Type	Description
group		**Integer**	To group N parts together.
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

94.2. EXAMPLE

See Split EIP which has examples using the XML Tokenize language.

94.3. SPRING BOOT AUTO-CONFIGURATION

When using xtokenize with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-xml-jaxp-starter</artifactId>
</dependency>
```

The component supports 3 options, which are listed below.

Name	Description	Default	Type
camel.language.xtokenize.enabled	Whether to enable auto configuration of the xtokenize language. This is enabled by default.		Boolean
camel.language.xtokenize.mode	The extraction mode. The available extraction modes are: i - injecting the contextual namespace bindings into the extracted token (default) w - wrapping the extracted token in its ancestor context u - unwrapping the extracted token to its child content t - extracting the text content of the specified element.		String
camel.language.xtokenize.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

CHAPTER 95. XPATH

Camel supports XPath to allow an Expression or Predicate to be used in the DSL.

For example, you could use XPath to create a predicate in a Message Filter or as an expression for a Recipient List.

95.1. XPATH LANGUAGE OPTIONS

The XPath language supports 10 options, which are listed below.

Name	Default	Java Type	Description
documentType		**String**	Name of class for document type The default value is org.w3c.dom.Document.
resultType		**Enum**	Sets the class name of the result type (type from output) The default result type is NodeSet. Enum values: • NUMBER • STRING • BOOLEAN • NODESET • NODE
saxon		**Boolean**	Whether to use Saxon.
factoryRef		**String**	References to a custom XPathFactory to lookup in the registry.
objectModel		**String**	The XPath object model to use.
logNamespaces		**Boolean**	Whether to log namespaces which can assist during troubleshooting.
headerName		**String**	Name of header to use as input, instead of the message body.

Name	Default	Java Type	Description
threadSafety		**Boolean**	Whether to enable thread-safety for the returned result of the xpath expression. This applies to when using NODESET as the result type, and the returned set has multiple elements. In this situation there can be thread-safety issues if you process the NODESET concurrently such as from a Camel Splitter EIP in parallel processing mode. This option prevents concurrency issues by doing defensive copies of the nodes. It is recommended to turn this option on if you are using camel-saxon or Saxon in your application. Saxon has thread-safety issues which can be prevented by turning this option on.
preCompile		**Boolean**	Whether to enable pre-compiling the xpath expression during initialization phase. pre-compile is enabled by default. This can be used to turn off, for example in cases the compilation phase is desired at the starting phase, such as if the application is ahead of time compiled (for example with camel-quarkus) which would then load the xpath factory of the built operating system, and not a JVM runtime.
trim		**Boolean**	Whether to trim the value to remove leading and trailing whitespaces and line breaks.

95.2. NAMESPACES

You can easily use namespaces with XPath expressions using the **Namespaces** helper class.

95.3. VARIABLES

Variables in XPath is defined in different namespaces. The default namespace is **http://camel.apache.org/schema/spring**.

Namespace URI	Local part	Type	Description
http://camel.apache.org/xml/in/	in	Message	the message
http://camel.apache.org/xml/out/	out	Message	**deprecated** the output message (do not use)
http://camel.apache.org/xml/function/	functions	Object	Additional functions

Namespace URI	Local part	Type	Description
http://camel.apache.org/xml/variables/environment-variables	env	Object	OS environment variables
http://camel.apache.org/xml/variables/system-properties	system	Object	Java System properties
http://camel.apache.org/xml/variables/exchange-property		Object	the exchange property

Camel will resolve variables according to either:

- namespace given
- no namespace given

95.3.1. Namespace given

If the namespace is given then Camel is instructed exactly what to return. However, when resolving Camel will try to resolve a header with the given local part first, and return it. If the local part has the value **body** then the body is returned instead.

95.3.2. No namespace given

If there is no namespace given then Camel resolves only based on the local part. Camel will try to resolve a variable in the following steps:

- from **variables** that has been set using the **variable(name, value)** fluent builder
- from **message.in.header** if there is a header with the given key
- from **exchange.properties** if there is a property with the given key

95.4. FUNCTIONS

Camel adds the following XPath functions that can be used to access the exchange:

Function	Argument	Type	Description
in:body	none	Object	Will return the message body.
in:header	the header name	Object	Will return the message header.
out:body	none	Object	**deprecated** Will return the out message body.
out:header	the header name	Object	**deprecated** Will return the out message header.

Function	Argument	Type	Description
function:properties	key for property	String	To use a .
function:simple	simple expression	Object	To evaluate a language.

NOTE

function:properties and **function:simple** is not supported when the return type is a **NodeSet**, such as when using with a Split EIP.

Here's an example showing some of these functions in use.

95.4.1. Functions example

If you prefer to configure your routes in your Spring XML file then you can use XPath expressions as follows

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

  <camelContext id="camel" xmlns="http://activemq.apache.org/camel/schema/spring"
                xmlns:foo="http://example.com/person">
    <route>
      <from uri="activemq:MyQueue"/>
      <filter>
        <xpath>/foo:person[@name='James']</xpath>
        <to uri="mqseries:SomeOtherQueue"/>
      </filter>
    </route>
  </camelContext>
</beans>
```

Notice how we can reuse the namespace prefixes, **foo** in this case, in the XPath expression for easier namespace based XPath expressions.

95.5. STREAM BASED MESSAGE BODIES

If the message body is stream based, which means the input it receives is submitted to Camel as a stream. That means you will only be able to read the content of the stream **once**. So often when you use XPath as Message Filter or Content Based Router then you need to access the data multiple times, and you should use Stream Caching or convert the message body to a **String** prior which is safe to be re-read multiple times.

```
from("queue:foo").
  filter().xpath("//foo")).
  to("queue:bar")
```

```
from("queue:foo").
  choice().xpath("//foo")).to("queue:bar").
  otherwise().to("queue:others");
```

95.6. SETTING RESULT TYPE

The XPath expression will return a result type using native XML objects such as **org.w3c.dom.NodeList**. However, many times you want a result type to be a **String**. To do this you have to instruct the XPath which result type to use.

In Java DSL:

```
xpath("/foo:person/@id", String.class)
```

In XML DSL you use the **resultType** attribute to provide the fully qualified classname.

```
<xpath resultType="java.lang.String">/foo:person/@id</xpath>
```

NOTE

Classes from **java.lang** can omit the FQN name, so you can use **resultType="String"**

Using **@XPath** annotation:

```
@XPath(value = "concat('foo-',//order/name/)", resultType = String.class) String name)
```

Where we use the xpath function concat to prefix the order name with **foo-**. In this case we have to specify that we want a **String** as result type, so the concat function works.

95.7. USING XPATH ON HEADERS

Some users may have XML stored in a header. To apply an XPath to a header's value you can do this by defining the 'headerName' attribute.

```
<xpath headerName="invoiceDetails">/invoice/@orderType = 'premium'</xpath>
```

And in Java DSL you specify the headerName as the 2nd parameter as shown:

```
xpath("/invoice/@orderType = 'premium'", "invoiceDetails")
```

95.8. EXAMPLE

Here is a simple example using an XPath expression as a predicate in a Message Filter:

```
from("direct:start")
    .filter().xpath("/person[@name='James']")
        .to("mock:result");
```

And in XML

```
<route>
  <from uri="direct:start"/>
  <filter>
    <xpath>/person[@name='James']</xpath>
    <to uri="mock:result"/>
  </filter>
</route>
```

95.9. USING NAMESPACES

If you have a standard set of namespaces you wish to work with and wish to share them across many XPath expressions you can use the **org.apache.camel.support.builder.Namespaces** when using Java DSL as shown:

```
Namespaces ns = new Namespaces("c", "http://acme.com/cheese");

from("direct:start")
    .filter(xpath("/c:person[@name='James']", ns))
        .to("mock:result");
```

Notice how the namespaces are provided to **xpath** with the **ns** variable that are passed in as the 2nd parameter.

Each namespace is a key=value pair, where the prefix is the key. In the XPath expression then the namespace is used by its prefix, eg:

```
/c:person[@name='James']
```

The namespace builder supports adding multiple namespaces as shown:

```
Namespaces ns = new Namespaces("c", "http://acme.com/cheese")
                    .add("w", "http://acme.com/wine")
                    .add("b", "http://acme.com/beer");
```

When using namespaces in XML DSL then its different, as you setup the namespaces in the XML root tag (or one of the **camelContext**, **routes**, **route** tags).

In the XML example below we use Spring XML where the namespace is declared in the root tag **beans**, in the line with **xmlns:foo="http://example.com/person"**:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:foo="http://example.com/person"
       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
    ">

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="direct:start"/>
      <filter>
```

```
      <xpath logNamespaces="true">/foo:person[@name='James']</xpath>
      <to uri="mock:result"/>
    </filter>
  </route>
</camelContext>

</beans>
```

This namespace uses **foo** as prefix, so the **<xpath>** expression uses **/foo:** to use this namespace.

95.10. USING @XPATH ANNOTATION FOR BEAN INTEGRATION

You can use Bean Integration to invoke a method on a bean and use various languages such as **@XPath** to extract a value from the message and bind it to a method parameter.

NOTE

The default **@XPath** annotation has SOAP and XML namespaces available.

```
public class Foo {

    @Consume(uri = "activemq:my.queue")
    public void doSomething(@XPath("/person/@name") String name, String xml) {
        // process the inbound message here
    }
}
```

95.11. USING XPATHBUILDER WITHOUT AN EXCHANGE

You can now use the **org.apache.camel.language.xpath.XPathBuilder** without the need for an **Exchange**. This comes handy if you want to use it as a helper to do custom XPath evaluations.

It requires that you pass in a **CamelContext** since a lot of the moving parts inside the **XPathBuilder** requires access to the Camel Type Converter and hence why **CamelContext** is needed.

For example, you can do something like this:

```
boolean matches = XPathBuilder.xpath("/foo/bar/@xyz").matches(context, "<foo><bar xyz='cheese'/>
</foo>"));
```

This will match the given predicate.

You can also evaluate as shown in the following three examples:

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar></foo>",
String.class);
Integer number = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>123</bar></foo>",
Integer.class);
Boolean bool = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>true</bar></foo>",
Boolean.class);
```

Evaluating with a **String** result is a common requirement and make this simpler:

```
String name = XPathBuilder.xpath("foo/bar").evaluate(context, "<foo><bar>cheese</bar></foo>");
```

95.12. USING SAXON WITH XPATHBUILDER

You need to add **camel-saxon** as dependency to your project.

It's now easier to use Saxon with the XPathBuilder which can be done in several ways as shown below

- Using a custom XPathFactory
- Using ObjectModel

95.12.1. Setting a custom XPathFactory using System Property

Camel now supports reading the JVM system property **javax.xml.xpath.XPathFactory** that can be used to set a custom XPathFactory to use.

This unit test shows how this can be done to use Saxon instead:

Camel will log at **INFO** level if it uses a non default XPathFactory such as:

```
XPathBuilder  INFO  Using system property
javax.xml.xpath.XPathFactory:http://saxon.sf.net/jaxp/xpath/om with value:
                net.sf.saxon.xpath.XPathFactoryImpl when creating XPathFactory
```

To use Apache Xerces you can configure the system property

```
-Djavax.xml.xpath.XPathFactory=org.apache.xpath.jaxp.XPathFactoryImpl
```

95.12.2. Enabling Saxon from XML DSL

Similarly to Java DSL, to enable Saxon from XML DSL you have three options:

Referring to a custom factory:

```
<xpath factoryRef="saxonFactory" resultType="java.lang.String">current-dateTime()</xpath>
```

And declare a bean with the factory:

```
<bean id="saxonFactory" class="net.sf.saxon.xpath.XPathFactoryImpl"/>
```

Specifying the object model:

```
<xpath objectModel="http://saxon.sf.net/jaxp/xpath/om" resultType="java.lang.String">current-
dateTime()</xpath>
```

And the recommended approach is to set **saxon=true** as shown:

```
<xpath saxon="true" resultType="java.lang.String">current-dateTime()</xpath>
```

95.13. NAMESPACE AUDITING TO AID DEBUGGING

Many XPath-related issues that users frequently face are linked to the usage of namespaces. You may have some misalignment between the namespaces present in your message, and those that your XPath expression is aware of or referencing. XPath predicates or expressions that are unable to locate the XML elements and attributes due to namespaces issues may simply look like *they are not working*, when in reality all there is to it is a lack of namespace definition.

Namespaces in XML are completely necessary, and while we would love to simplify their usage by implementing some magic or voodoo to wire namespaces automatically, truth is that any action down this path would disagree with the standards and would greatly hinder interoperability.

Therefore, the utmost we can do is assist you in debugging such issues by adding two new features to the XPath Expression Language and are thus accessible from both predicates and expressions.

95.13.1. Logging the Namespace Context of your XPath expression/predicate

Every time a new XPath expression is created in the internal pool, Camel will log the namespace context of the expression under the **org.apache.camel.language.xpath.XPathBuilder** logger. Since Camel represents Namespace Contexts in a hierarchical fashion (parent-child relationships), the entire tree is output in a recursive manner with the following format:

```
[me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}, {prefix -> namespace}], [parent: [me: {prefix -> namespace}]]]
```

Any of these options can be used to activate this logging:

- Enable TRACE logging on the **org.apache.camel.language.xpath.XPathBuilder** logger, or some parent logger such as **org.apache.camel** or the root logger
- Enable the **logNamespaces** option as indicated in the following section, in which case the logging will occur on the INFO level

95.13.2. Auditing namespaces

Camel is able to discover and dump all namespaces present on every incoming message before evaluating an XPath expression, providing all the richness of information you need to help you analyse and pinpoint possible namespace issues.

To achieve this, it in turn internally uses another specially tailored XPath expression to extract all namespace mappings that appear in the message, displaying the prefix and the full namespace URI(s) for each individual mapping.

Some points to take into account:

- The implicit XML namespace (**xmlns:xml="http://www.w3.org/XML/1998/namespace"**) is suppressed from the output because it adds no value
- Default namespaces are listed under the **DEFAULT** keyword in the output
- Keep in mind that namespaces can be remapped under different scopes. Think of a top-level 'a' prefix which in inner elements can be assigned a different namespace, or the default namespace changing in inner scopes. For each discovered prefix, all associated URIs are listed.

You can enable this option in Java DSL and XML DSL:

Java DSL:

```
XPathBuilder.xpath("/foo:person/@id", String.class).logNamespaces()
```

XML DSL:

```
<xpath logNamespaces="true" resultType="String">/foo:person/@id</xpath>
```

The result of the auditing will be appeared at the INFO level under the **org.apache.camel.language.xpath.XPathBuilder** logger and will look like the following:

```
2012-01-16 13:23:45,878 [stSaxonWithFlag] INFO  XPathBuilder  - Namespaces discovered in
message:
{xmlns:a=[http://apache.org/camel], DEFAULT=[http://apache.org/default],
xmlns:b=[http://apache.org/camelA, http://apache.org/camelB]}
```

95.14. LOADING SCRIPT FROM EXTERNAL RESOURCE

You can externalize the script and have Camel load it from a resource such as **"classpath:"**, **"file:"**, or **"http:"**. This is done using the following syntax: **"resource:scheme:location"**, eg to refer to a file on the classpath you can do:

```
.setHeader("myHeader").xpath("resource:classpath:myxpath.txt", String.class)
```

95.15. DEPENDENCIES

To use XPath in your camel routes you need to add the dependency on **camel-xpath** which implements the XPath language.

If you use maven you could just add the following to your pom.xml, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-xpath</artifactId>
  <version>{CamelSBProjectVersion}</version>
</dependency>
```

95.16. SPRING BOOT AUTO-CONFIGURATION

When using xpath with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-xpath-starter</artifactId>
</dependency>
```

The component supports 9 options, which are listed below.

Name	Description	Default	Type
camel.language.xpath.document-type	Name of class for document type The default value is org.w3c.dom.Document.		String
camel.language.xpath.enabled	Whether to enable auto configuration of the xpath language. This is enabled by default.		Boolean
camel.language.xpath.factory-ref	References to a custom XPathFactory to lookup in the registry.		String
camel.language.xpath.log-namespaces	Whether to log namespaces which can assist during troubleshooting.	false	Boolean
camel.language.xpath.object-model	The XPath object model to use.		String
camel.language.xpath.pre-compile	Whether to enable pre-compiling the xpath expression during initialization phase. pre-compile is enabled by default. This can be used to turn off, for example in cases the compilation phase is desired at the starting phase, such as if the application is ahead of time compiled (for example with camel-quarkus) which would then load the xpath factory of the built operating system, and not a JVM runtime.	true	Boolean
camel.language.xpath.saxon	Whether to use Saxon.	false	Boolean
camel.language.xpath.thread-safety	Whether to enable thread-safety for the returned result of the xpath expression. This applies to when using NODESET as the result type, and the returned set has multiple elements. In this situation there can be thread-safety issues if you process the NODESET concurrently such as from a Camel Splitter EIP in parallel processing mode. This option prevents concurrency issues by doing defensive copies of the nodes. It is recommended to turn this option on if you are using camel-saxon or Saxon in your application. Saxon has thread-safety issues which can be prevented by turning this option on.	false	Boolean
camel.language.xpath.trim	Whether to trim the value to remove leading and trailing whitespaces and line breaks.	true	Boolean

CHAPTER 96. KAMELET MAIN

Since Camel 3.11

A **main** class that is opinionated to boostrap and run Camel standalone with Kamelets (or plain YAML routes) for development and demo purposes.

96.1. INITIAL CONFIGURATION

The **KameletMain** is pre-configured with the following properties:

```
camel.component.kamelet.location = classpath:/kamelets,github:apache:camel-kamelets/kamelets
camel.component.rest.consumerComponentName = platform-http
camel.component.rest.producerComponentName = vertx-http
```

You can override these settings by updating the configuration in **application.properties**.

96.2. AUTOMATIC DEPENDENCIES DOWNLOADING

The Kamelet Main can automatically download Kamelet YAML files from a remote location over http/https, and from github as well.

The official Kamelets from the Apache Camel Kamelet Catalog is stored on github and they can be used out of the box as-is.

For example a Camel route can be **coded** in YAML which uses the Earthquake Kamelet from the catalog, as shown below:

```
- route:
    from: "kamelet:earthquake-source"
    steps:
      - unmarshal:
          json: {}
      - log: "Earthquake with magnitude ${body[properties][mag]} at ${body[properties][place]}"
```

In the above example, the earthquake kamelet will be downloaded from github, and as well its required dependencies.

For more information, see Kamelet Main example

CHAPTER 97. OPENAPI JAVA

The Rest DSL can be integrated with the **camel-openapi-java** module which is used for exposing the REST services and their APIs using OpenApi.

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

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-openapi-java</artifactId>
  <version>{CamelSBProjectVersion}</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

The camel-openapi-java module can be used from the REST components (without the need for servlet)

97.1. USING OPENAPI IN REST-DSL

You can enable the OpenApi api from the rest-dsl by configuring the **apiContextPath** dsl as shown below:

```
public class UserRouteBuilder extends RouteBuilder {
  @Override
  public void configure() throws Exception {
    // configure we want to use servlet as the component for the rest DSL
    // and we enable json binding mode
    restConfiguration().component("netty-http").bindingMode(RestBindingMode.json)
      // and output using pretty print
      .dataFormatProperty("prettyPrint", "true")
      // setup context path and port number that netty will use
      .contextPath("/").port(8080)
      // add OpenApi api-doc out of the box
      .apiContextPath("/api-doc")
        .apiProperty("api.title", "User API").apiProperty("api.version", "1.2.3")
        // and enable CORS
        .apiProperty("cors", "true");

    // this user REST service is json only
    rest("/user").description("User rest service")
      .consumes("application/json").produces("application/json")
      .get("/{id}").description("Find user by id").outType(User.class)
        .param().name("id").type(path).description("The id of the user to
get").dataType("int").endParam()
        .to("bean:userService?method=getUser(${header.id})")
      .put().description("Updates or create a user").type(User.class)
        .param().name("body").type(body).description("The user to update or create").endParam()
        .to("bean:userService?method=updateUser")
      .get("/findAll").description("Find all users").outType(User[].class)
        .to("bean:userService?method=listUsers");
  }
}
```

97.2. OPTIONS

The OpenApi module can be configured using the following options. To configure using a servlet you use the init-param as shown above. When configuring directly in the rest-dsl, you use the appropriate method, such as **enableCORS**, **host,contextPath**, dsl. The options with **api.xxx** is configured using **apiProperty** dsl.

Option	Type	Description
cors	Boolean	Whether to enable CORS. Notice this only enables CORS for the api browser, and not the actual access to the REST services. Is default false.
openapi.version	String	OpenApi spec version. Is default 3.0.
host	String	To setup the hostname. If not configured camel-openapi-java will calculate the name as localhost based.
schemes	String	The protocol schemes to use. Multiple values can be separated by comma such as "http,https". The default value is "http".
base.path	String	**Required**: To setup the base path where the REST services is available. The path is relative (eg do not start with http/https) and camel-openapi-java will calculate the absolute base path at runtime, which will be **protocol://host:port/context-path/base.path**
api.path	String	To setup the path where the API is available (eg /api-docs). The path is relative (eg do not start with http/https) and camel-openapi-java will calculate the absolute base path at runtime, which will be **protocol://host:port/context-path/api.path** So using relative paths is much easier. See above for an example.
api.version	String	The version of the api. Is default 0.0.0.
api.title	String	The title of the application.
api.description	String	A short description of the application.
api.termsOfService	String	A URL to the Terms of Service of the API.
api.contact.name	String	Name of person or organization to contact
api.contact.email	String	An email to be used for API-related correspondence.
api.contact.url	String	A URL to a website for more contact information.
api.license.name	String	The license name used for the API.
api.license.url	String	A URL to the license used for the API.

97.3. ADDING SECURITY DEFINITIONS IN API DOC

The Rest DSL now supports declaring OpenApi **securityDefinitions** in the generated API document. For example as shown below:

```
rest("/user").tag("dude").description("User rest service")
    // setup security definitions
    .securityDefinitions()
        .oauth2("petstore_auth").authorizationUrl("http://petstore.swagger.io/oauth/dialog").end()
        .apiKey("api_key").withHeader("myHeader").end()
    .end()
    .consumes("application/json").produces("application/json")
```

Here we have setup two security definitions

- OAuth2 - with implicit authorization with the provided url
- Api Key - using an api key that comes from HTTP header named *myHeader*

Then you need to specify on the rest operations which security to use by referring to their key (petstore_auth or api_key).

```
.get("/{id}/{date}").description("Find user by id and date").outType(User.class)
    .security("api_key")

...

.put().description("Updates or create a user").type(User.class)
    .security("petstore_auth", "write:pets,read:pets")
```

Here the get operation is using the Api Key security and the put operation is using OAuth security with permitted scopes of read and write pets.

97.4. JSON OR YAML

The camel-openapi-java module supports both JSon and Yaml out of the box. You can specify in the request url what you want returned by using /openapi.json or /openapi.yaml for either one. If none is specified then the HTTP Accept header is used to detect if json or yaml can be accepted. If either both is accepted or none was set as accepted then json is returned as the default format.

97.5. USEXFORWARDHEADERS AND API URL RESOLUTION

The OpenApi specification allows you to specify the host, port & path that is serving the API. In OpenApi V2 this is done via the **host** field and in OpenAPI V3 it is part of the **servers** field.

By default, the value for these fields is determined by **X-Forwarded** headers, **X-Forwarded-Host** & **X-Forwarded-Proto**.

This can be overridden by disabling the lookup of **X-Forwarded** headers and by specifying your own host, port & scheme on the REST configuration.

```
restConfiguration().component("netty-http")
    .useXForwardHeaders(false)
    .apiProperty("schemes", "https");
    .host("localhost")
    .port(8080);
```

97.6. EXAMPLES

In the Apache Camel distribution we ship the **camel-example-openapi-cdi** and **camel-example-spring-boot-rest-openapi-simple** which demonstrates using this OpenApi component.

97.7. SPRING BOOT AUTO-CONFIGURATION

When using openapi-java with Spring Boot make sure to use the following Maven dependency to have support for auto configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-openapi-java-starter</artifactId>
</dependency>
```

The component supports 1 options, which are listed below.

Name	Description	Default	Type
camel.openapi.enabled	Enables Camel Rest DSL to automatic register its OpenAPI (eg swagger doc) in Spring Boot which allows tooling such as SpringDoc to integrate with Camel.	true	Boolean

CHAPTER 98. OPENTELEMETRY

Since Camel 3.5

The OpenTelemetry component is used for tracing and timing the incoming and outgoing Camel messages using OpenTelemetry.

Events (spans) are captured for incoming and outgoing messages that are sent to/from Camel.

98.1. CONFIGURATION

The configuration properties for the OpenTelemetry tracer are:

Option	Default	Description
excludePatterns		Sets exclude pattern(s) that will disable tracing for Camel messages that matches the pattern. The content is a Set<String> where the key is a pattern. The pattern uses the rules from Intercept.
encoding	false	Sets whether the header keys need to be encoded (connector specific) or not. The value is a boolean. Dashes need for instances to be encoded for JMS property keys.

98.1.1. Configuration

Add the **camel-opentelemetry** component in your POM, in addition to any specific dependencies associated with the chosen OpenTelemetry compliant Tracer.

To explicitly configure OpenTelemetry support, instantiate the **OpenTelemetryTracer** and initialize the camel context. You can optionally specify a **Tracer**, or alternatively it can be implicitly discovered using the **Registry**

```
OpenTelemetryTracer otelTracer = new OpenTelemetryTracer();
// By default it uses the DefaultTracer, but you can override it with a specific OpenTelemetry Tracer implementation.
otelTracer.setTracer(...);
// And then initialize the context
otelTracer.init(camelContext);
```

98.2. SPRING BOOT

Add the **camel-opentelemetry-starter** dependency, and then turn on the OpenTracing by annotating the main class with **@CamelOpenTelemetry**.

The **OpenTelemetryTracer** is implicitly obtained from the camel context's **Registry**, unless a **OpenTelemetryTracer** bean has been defined by the application.

98.3. JAVA AGENT

Download the latest version of Java agent .

This package includes the instrumentation agent as well as instrumentations for all supported libraries and all available data exporters. The package provides a completely automatic, out-of-the-box experience.

Enable the instrumentation agent using the **-javaagent** flag to the JVM.

```
java -javaagent:path/to/opentelemetry-javaagent.jar \
     -jar myapp.jar
```

By default, the OpenTelemetry Java agent uses OTLP exporter configured to send data to OpenTelemetry collector at http://localhost:4317.

Configuration parameters are passed as Java system properties (**-D** flags) or as environment variables. See Configuring the agent and OpenTelemetry auto-configurationfor the full list of configuration items. For example:

```
java -javaagent:path/to/opentelemetry-javaagent.jar \
     -Dotel.service.name=your-service-name \
     -Dotel.traces.exporter=jaeger \
     -jar myapp.jar
```

98.4. SPRING BOOT AUTO-CONFIGURATION

Add the following dependency to your **pom.xml** for this component:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-opentelemetry-starter</artifactId>
  <version>3.20.1.redhat-00047</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```

The component supports 2 options, which are listed below.

Name	Description	Default	Type
camel.opentelemetry.encoding	Activate or deactivate the dash encoding in headers (required by JMS) for messaging.		Boolean
camel.opentelemetry.exclude-patterns	Sets exclude pattern(s) that will disable the tracing for the Camel messages that matches the pattern.		Set

98.5. MDC LOGGING

When MDC Logging is enabled for the active Camel context, the Trace ID and Span ID are added and removed from the MDC for each route, where the keys are **trace_id** and **span_id**, respectively.

CHAPTER 99. SPRING SECURITY

Since Camel 2.3

The Camel Spring Security component provides role-based authorization for Camel routes. It leverages the authentication and user services provided by Spring Security (formerly Acegi Security) and adds a declarative, role-based policy system to control whether a route can be executed by a given principal.

If you are not familiar with the Spring Security authentication and authorization system, please review the current reference documentation on the SpringSource web site linked above.

99.1. CREATING AUTHORIZATION POLICIES

Access to a route is controlled by an instance of a **SpringSecurityAuthorizationPolicy** object. A policy object contains the name of the Spring Security authority (role) required to run a set of endpoints and references to Spring Security **AuthenticationManager** and **AccessDecisionManager** objects used to determine whether the current principal has been assigned that role. Policy objects may be configured as Spring beans or by using an **<authorizationPolicy>** element in Spring XML.

The **<authorizationPolicy>** element can contain the following attributes:

Name	Default Value	Description
id	**null**	The unique Spring bean identifier which is used to reference the policy in routes (required)
access	**null**	The Spring Security authority name that is passed to the access decision manager (required)
authenticationManager	**authenticationManager**	The name of the Spring Security **AuthenticationManager** object in the context
accessDecisionManager	**accessDecisionManager**	The name of the Spring Security **AccessDecisionManager** object in the context
authenticationAdapter	DefaultAuthenticationAdapter	The name of a **camel-spring-security AuthenticationAdapter** object in the context that is used to convert a **javax.security.auth.Subject** into a Spring Security **Authentication** instance.

Name	Default Value	Description
useThreadSecurityContext	**true**	If a **javax.security.auth.Subject** cannot be found in the In message header under Exchange.AUTHENTICATION, check the Spring Security **SecurityContextHolder** for an **Authentication** object.
alwaysReauthenticate	**false**	If set to true, the **SpringSecurityAuthorizationPolicy** will always call **AuthenticationManager.authenticate()** each time the policy is accessed.

99.2. CONTROLLING ACCESS TO CAMEL ROUTES

A Spring Security **AuthenticationManager** and **AccessDecisionManager** are required to use this component. Here is an example of how to configure these objects in Spring XML using the Spring Security namespace:

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

  <bean id="accessDecisionManager"
class="org.springframework.security.access.vote.AffirmativeBased">
    <property name="allowIfAllAbstainDecisions" value="true"/>
    <property name="decisionVoters">
      <list>
        <bean class="org.springframework.security.access.vote.RoleVoter"/>
      </list>
    </property>
  </bean>

  <spring-security:authentication-manager alias="authenticationManager">
    <spring-security:authentication-provider user-service-ref="userDetailsService"/>
  </spring-security:authentication-manager>

  <spring-security:user-service id="userDetailsService">
    <spring-security:user name="jim" password="jimspassword" authorities="ROLE_USER,
ROLE_ADMIN"/>
    <spring-security:user name="bob" password="bobspassword" authorities="ROLE_USER"/>
  </spring-security:user-service>

</beans>
```

Now that the underlying security objects are set up, we can use them to configure an authorization policy and use that policy to control access to a route:

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

    <!-- import the Spring security configuration  -->
    <import resource=
"classpath:org/apache/camel/component/spring/security/commonSecurity.xml"/>

    <authorizationPolicy id="admin" access="ROLE_ADMIN"
      authenticationManager="authenticationManager"
      accessDecisionManager="accessDecisionManager"
      xmlns="http://camel.apache.org/schema/spring-security"/>

    <camelContext id="myCamelContext" xmlns="http://camel.apache.org/schema/spring">
      <route>
        <from uri="direct:start"/>
        <!-- The exchange should be authenticated with the role -->
        <!-- of ADMIN before it is send to mock:endpoint -->
        <policy ref="admin">
          <to uri="mock:end"/>
        </policy>
      </route>
    </camelContext>
</beans>
```

In this example, the endpoint **mock:end** is only executed if:

- The *admin* **SpringSecurityAuthorizationPolicy** can locate a Spring Security **Authentication** object, that:
  - can be authenticated or is possible to authenticate
  - contains the **ROLE_ADMIN** authority

99.3. AUTHENTICATION

The process of obtaining security credentials that are used for authorization is not specified by this component. You can write your own processors or components which get authentication information from the exchange depending on your needs. For example, you can create a processor that gets credentials from an HTTP request header originating in the Jetty component. No matter how the credentials are collected, they need to be placed in the **Inmessage** or the **SecurityContextHolder** so the Camel Spring Security component can access them.

```
import javax.security.auth.Subject;
```

```
import org.apache.camel.*;
import org.apache.commons.codec.binary.Base64;
import org.springframework.security.authentication.*;


public class MyAuthService implements Processor {
    public void process(Exchange exchange) throws Exception {
        // get the username and password from the HTTP header
        // http://en.wikipedia.org/wiki/Basic_access_authentication
        String userpass = new
String(Base64.decodeBase64(exchange.getIn().getHeader("Authorization", String.class)));
        String[] tokens = userpass.split(":");

        // create an Authentication object
        UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(tokens[0], tokens[1]);

        // wrap it in a Subject
        Subject subject = new Subject();
        subject.getPrincipals().add(authToken);

        // place the Subject in the In message
        exchange.getIn().setHeader(Exchange.AUTHENTICATION, subject);

        // you could also do this if useThreadSecurityContext is set to true
        // SecurityContextHolder.getContext().setAuthentication(authToken);
    }
}
```

The **SpringSecurityAuthorizationPolicy** automatically authenticates the **Authentication** object if necessary.

NOTE

Be aware of these two issues when you use the **SecurityContextHolder** instead of or in addition to the **Exchange.AUTHENTICATION** header:

1. The context holder uses a thread-local variable to hold the **Authentication** object. Any routes that cross thread boundaries, like **seda** or **jms**, lose the **Authentication** object.

2. The Spring Security system expects that an **Authentication** object in the context is already authenticated and has roles.
   See the Spring Technical Overview, section 5.3.1: "What is authentication in Spring Security?" for more details.

The default behavior of **camel-spring-security** is to look for a **Subject** in the **Exchange.AUTHENTICATION** header. This **Subject** must contain at least one principal, which must be a subclass of **org.springframework.security.core.Authentication**.

You can customize the mapping of **Subject** to **Authentication** object by providing an implementation of the **org.apache.camel.component.spring.security.AuthenticationAdapter** to your **<authorizationPolicy>** bean.

This can be useful if you are working with components that do not use Spring Security but do provide a **Subject**.

At this time, only the CXF component populates the **Exchange.AUTHENTICATION** header.

99.4. HANDLING AUTHENTICATION AND AUTHORIZATION ERRORS

If authentication or authorization fails in the **SpringSecurityAuthorizationPolicy**, a **CamelAuthorizationException** is thrown. This can be handled using Camel's standard exception handling methods, like the Exception Clause. The **CamelAuthorizationException** has a reference to the ID of the policy which threw the exception so you can handle errors based on the policy as well as the type of exception.

```
<onException>
  <exception>org.springframework.security.authentication.AccessDeniedException</exception>
  <choice>
    <when>
      <simple>${exception.policyId} == 'user'</simple>
      <transform>
        <constant>You do not have ROLE_USER access!</constant>
      </transform>
    </when>
    <when>
      <simple>${exception.policyId} == 'admin'</simple>
      <transform>
        <constant>You do not have ROLE_ADMIN access!</constant>
      </transform>
    </when>
  </choice>
</onException>
```

99.5. SPRING BOOT AUTO-CONFIGURATION

When using spring-security with Spring Boot, use the following Maven dependency to enable support for auto-configuration:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-security-starter</artifactId>
</dependency>
```

The component has no Spring Boot auto configuration options.

CHAPTER 100. YAML DSL

Since Camel 3.9

The YAML DSL provides the capability to define your Camel routes, route templates & REST DSL configuration in YAML.

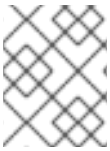
100.1. DEFINING A ROUTE

A route is a collection of elements defined as follows:

```
- from: 1
    uri: "direct:start"
    steps: 2
      - filter:
          expression:
            simple: "${in.header.continue} == true"
          steps:
            - to:
                uri: "log:filtered"
      - to:
          uri: "log:original"
```

Where,

1. Route entry point, by default **from** and **rest** are supported.
2. Processing steps



NOTE

Each step represents a YAML map that has a single entry where the field name is the EIP name.

As a general rule, each step provides all the parameters the related definition declares, but there are some minor differences/enhancements:

- Output Aware Steps

Some steps, such as **filter** and **split**, have their own pipeline when an exchange matches the filter expression or for the items generated by the split expression. You can define these pipelines in the **steps** field:

```
filter:
  expression:
    simple: "${in.header.continue} == true"
    steps:
      - to:
          uri: "log:filtered"
```

- Expression Aware Steps

Some EIP, such as **filter** and **split**, supports the definition of an expression through the **expression** field:

Explicit Expression field

```
filter:
  expression:
    simple: "${in.header.continue} == true"
```

To make the DSL less verbose, you can omit the **expression** field.

Implicit Expression field

```
filter:
  simple: "${in.header.continue} == true"
```

In general, expressions can be defined inline, such as within the examples above but if you need provide more information, you can 'unroll' the expression definition and configure any single parameter the expression defines.

Full Expression definition

```
filter:
  tokenize:
    token: "<"
    end-token: ">"
```

- Data Format Aware Steps

The EIP **marshal** and **unmarshal** supports the definition of data formats:

```
marshal:
  json:
    library: Gson
```

NOTE

In case you want to use the data-format's default settings, you need to place an empty block as data format parameters, like **json: {}**

100.2. DEFINING ENDPOINTS

To define an endpoint with the YAML DSL you have two options:

- Using a classic Camel URI:

```
- from:
    uri: "timer:tick?period=1s"
    steps:
      - to:
          uri: "telegram:bots?authorizationToken=XXX"
```

- Using URI and parameters:

```
- from:
```

```
    uri: "timer://tick"
    parameters:
      period: "1s"
    steps:
      - to:
          uri: "telegram:bots"
          parameters:
            authorizationToken: "XXX"
```

100.3. DEFINING BEANS

In addition to the general support for creating beans provided by Camel Main, the YAML DSL provide a convenient syntax to define and configure them:

```
- beans:
  - name: beanFromMap (1)
    type: com.acme.MyBean (2)
    properties: (3)
      foo: bar
```

Where,

1. The name of the bean which will bound the instance to the Camel Registry.
2. The full qualified class name of the bean
3. The properties of the bean to be set

The properties of the bean can be defined using either a map or properties style, as shown in the example below:

```
- beans:
  # map style
  - name: beanFromMap
    type: com.acme.MyBean
    properties:
      field1: 'f1'
      field2: 'f2'
      nested:
        field1: 'nf1'
        field2: 'nf2'
  # properties style
  - name: beanFromProps
    type: com.acme.MyBean
    properties:
      field1: 'f1_p'
      field2: 'f2_p'
      nested.field1: 'nf1_p'
      nested.field2: 'nf2_p'
```

NOTE

The **beans** elements is only used as root element.

100.4. CONFIGURING OPTIONS ON LANGUAGES

Some languages have additional configurations that you may need to use.

For example, the JSONPath can be configured to ignore JSON parsing errors. This is intended when you use a Content Based Router and want to route the message to different endpoints. The JSON payload of the message can be in different forms, meaning that the JSonPath expressions in some cases would fail with an exception, and other times not. In this situation you must set **suppress-exception** to true, as shown below:

```
- from:
    uri: "direct:start"
    steps:
      - choice:
          when:
          - jsonpath:
              expression: "person.middlename"
              suppress-exceptions: true
            steps:
            - to: "mock:middle"
          - jsonpath:
              expression: "person.lastname"
              suppress-exceptions: true
            steps:
            - to: "mock:last"
          otherwise:
            steps:
              - to: "mock:other"
```

In the route above, the following message would have failed the JSonPath expression **person.middlename** because the JSON payload does not have a **middlename** field. To remedy this, we have suppressed the exception.

```
{
  "person": {
    "firstname": "John",
    "lastname": "Doe"
  }
}
```

100.5. EXTERNAL EXAMPLES

You can find a set of examples using **main-yaml** in Camel examples that demonstrate how to create the Camel Routes with YAML. You can also refer to Camel Kamelets where each Kamelet is defined using YAML.