Red Hat Integration 2021.Q4

Camel Extensions for Quarkus Reference

Camel Extensions for Quarkus provided by Red Hat

Last Updated: 2022-01-20

Red Hat Integration 2021.Q4 Camel Extensions for Quarkus Reference

Camel Extensions for Quarkus provided by Red Hat

Abstract

Camel Extensions for Quarkus provides Quarkus extensions for many of the Camel components. This reference describes the settings for each of the extensions supported by Red Hat.

Table of Contents

PREFACE 7
MAKING OPEN SOURCE MORE INCLUSIVE 7

CHAPTER 1. EXTENSIONS OVERVIEW 8
1.1. SUPPORT LEVEL DEFINITIONS 8
1.2. SUPPORTED EXTENSIONS 8
1.3. SUPPORTED DATA FORMATS 13
1.4. SUPPORTED LANGUAGES 14

CHAPTER 2. EXTENSIONS REFERENCE 17
2.1. AVRO 17
2.1.1. What’s inside 17
2.1.2. Maven coordinates 17
2.1.3. Additional Camel Quarkus configuration 17
2.1.3.1. Deprecated: @BuildTimeAvroDataFormat annotation. 18
2.2. AWS 2 DYNAMODB 18
2.2.1. What’s inside 18
2.2.2. Maven coordinates 18
2.2.3. SSL in native mode 19
2.2.4. Additional Camel Quarkus configuration 19
2.2.4.1. Optional integration with Quarkus Amazon DynamoDB 19
2.3. AWS 2 KINESIS 19
2.3.1. What’s inside 19
2.3.2. Maven coordinates 19
2.3.3. SSL in native mode 20
2.4. AWS 2 LAMBDA 20
2.4.1. What’s inside 20
2.4.2. Maven coordinates 20
2.4.3. Camel Quarkus limitations 20
2.4.4. SSL in native mode 21
2.4.5. Additional Camel Quarkus configuration 21
2.4.5.1. Not possible to leverage quarkus-amazon-lambda by Camel aws2-lambda extension 21
2.5. AWS 2 S3 STORAGE SERVICE 21
2.5.1. What’s inside 21
2.5.2. Maven coordinates 21
2.5.3. SSL in native mode 21
2.5.4. Additional Camel Quarkus configuration 21
2.5.4.1. Optional integration with Quarkus Amazon S3 21
2.6. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS) 22
2.6.1. What’s inside 22
2.6.2. Maven coordinates 22
2.6.3. SSL in native mode 22
2.6.4. Additional Camel Quarkus configuration 22
2.6.4.1. Optional integration with Quarkus Amazon SNS 22
2.7. AWS 2 SIMPLE QUEUE SERVICE (SQS) 23
2.7.1. What’s inside 23
2.7.2. Maven coordinates 23
2.7.3. SSL in native mode 23
2.7.4. Additional Camel Quarkus configuration 23
2.7.4.1. Optional integration with Quarkus Amazon SQS 23
2.8. BEAN 24

2.8.1. What's inside 24
2.8.2. Maven coordinates 24
2.8.3. Usage 24
2.9. BINDY 24
2.9.1. What's inside 25
2.9.2. Maven coordinates 25
2.9.3. Camel Quarkus limitations 25
2.10. CORE 25
2.10.1. What's inside 25
2.10.2. Maven coordinates 26
2.10.3. Additional Camel Quarkus configuration 26
2.10.3.1. Simple language 26
2.10.3.1.1. Using the OGNL notation 26
2.10.3.1.2. Using dynamic type resolution in native mode 26
2.10.3.1.3. Using the simple language with classpath resources in native mode 26
2.10.3.1.4. Configuring a custom bean via properties in native mode 27
2.11. DIRECT 32
2.11.1. What's inside 32
2.11.2. Maven coordinates 32
2.12. ELASTICSEARCH REST 33
2.12.1. What's inside 33
2.12.2. Maven coordinates 33
2.12.3. Usage 33
2.13. FILE 33
2.13.1. What's inside 34
2.13.2. Maven coordinates 34
2.14. FTP 34
2.14.1. What's inside 34
2.14.2. Maven coordinates 34
2.15. HL7 34
2.15.1. What's inside 34
2.15.2. Maven coordinates 35
2.15.3. Camel Quarkus limitations 35
2.16. HTTP 35
2.16.1. What's inside 35
2.16.2. Maven coordinates 35
2.16.3. SSL in native mode 35
2.16.4. transferException option in native mode 35
2.16.5. Additional Camel Quarkus configuration 36
2.17. JACKSON 36
2.17.1. What's inside 36
2.17.2. Maven coordinates 36
2.18. AVRO JACKSON 36
2.18.1. What's inside 36
2.18.2. Maven coordinates 36
2.19. PROTOBUF JACKSON 37
2.19.1. What's inside 37
2.19.2. Maven coordinates 37
2.20. JACKSONXML 37
2.20.1. What's inside 37
2.20.2. Maven coordinates 37
2.21. JIRA 37
2.21.1. What's inside 38

2.21.2. Maven coordinates 38
2.21.3. SSL in native mode 38
2.22. JMS 38
2.22.1. What's inside 38
2.22.2. Maven coordinates 38
2.22.3. Usage 38
2.22.3.1. Message mapping with org.w3c.dom.Node 38
2.22.3.2. Native mode support for javax.jms.ObjectMessage 39
2.22.4. transferException option in native mode 39
2.23. JSON PATH 39
2.23.1. What's inside 39
2.23.2. Maven coordinates 39
2.24. JTA 39
2.24.1. What's inside 39
2.24.2. Maven coordinates 39
2.24.3. Usage 40
2.25. KAFKA 40
2.25.1. What's inside 40
2.25.2. Maven coordinates 41
2.25.3. Additional Camel Quarkus configuration 41
2.26. KAMELET 41
2.26.1. What's inside 41
2.26.2. Maven coordinates 41
2.26.3. Usage 42
2.26.3.1. Pre-load Kamelets at build-time 42
2.26.4. Additional Camel Quarkus configuration 42
2.27. LOG 42
2.27.1. What's inside 42
2.27.2. Maven coordinates 42
2.28. MICROPROFILE HEALTH 42
2.28.1. What's inside 42
2.28.2. Maven coordinates 43
2.28.3. Usage 43
2.28.3.1. Provided health checks 43
2.28.3.1.1. Camel Context Health 43
2.28.3.1.2. Camel Route Health 43
2.28.4. Additional Camel Quarkus configuration 43
2.29. MICROPROFILE METRICS 44
2.29.1. What's inside 44
2.29.2. Maven coordinates 44
2.29.3. Usage 44
2.29.3.1. Camel Context metrics 44
2.29.3.2. Camel Route metrics 45
2.29.4. Additional Camel Quarkus configuration 46
2.30. MLLP 47
2.30.1. What's inside 47
2.30.2. Maven coordinates 47
2.30.3. Additional Camel Quarkus configuration 47
2.31. MOCK 47
2.31.1. What's inside 47
2.31.2. Maven coordinates 47
2.31.3. Usage 48
2.31.4. Camel Quarkus limitations 49

2.32. MONGODB 49
2.32.1. What's inside 49
2.32.2. Maven coordinates 49
2.32.3. Additional Camel Quarkus configuration 49
2.33. NETTY 50
2.33.1. What's inside 50
2.33.2. Maven coordinates 50
2.34. OPENAPI JAVA 50
2.34.1. What's inside 50
2.34.2. Maven coordinates 51
2.34.3. Camel Quarkus limitations 51
2.35. PLATFORM HTTP 51
2.35.1. What's inside 51
2.35.2. Maven coordinates 51
2.35.3. Usage 51
2.35.3.1. Basic Usage 51
2.35.3.2. Using platform-http via Camel REST DSL 52
2.35.3.3. Handling multipart/form-data file uploads 52
2.35.4. Additional Camel Quarkus configuration 52
2.35.4.1. Platform HTTP server configuration 52
2.35.4.2. Character encodings 53
2.36. REST 53
2.36.1. What's inside 53
2.36.2. Maven coordinates 53
2.36.3. Additional Camel Quarkus configuration 53
2.36.3.1. Path parameters containing special characters with platform-http 53
2.36.3.2. Configuring alternate REST transport providers 54
2.37. SALESFORCE 54
2.37.1. What's inside 54
2.37.2. Maven coordinates 54
2.37.3. Usage 55
2.37.3.1. Generating Salesforce DTOs with the salesforce-maven-plugin 55
2.37.3.2. Register additional Salesforce classes for reflection in native mode 55
2.37.4. SSL in native mode 56
2.38. XQUERY 56
2.38.1. What's inside 56
2.38.2. Maven coordinates 56
2.38.3. Additional Camel Quarkus configuration 56
2.39. SEDA 56
2.39.1. What's inside 57
2.39.2. Maven coordinates 57
2.40. SOAP DATAFORMAT 57
2.40.1. What's inside 57
2.40.2. Maven coordinates 57
2.41. SQL 57
2.41.1. What's inside 57
2.41.2. Maven coordinates 57
2.41.3. Additional Camel Quarkus configuration 58
2.41.3.1. Configuring a DataSource 58
2.41.3.1.1. Zero configuration with Quarkus Dev Services 58
2.41.3.2. SQL scripts 58
2.41.3.3. SQL Aggregator 58
2.42. TIMER 58

2.42.1. What’s inside 58
2.42.2. Maven coordinates 59
2.43. XPATH 59
2.43.1. What’s inside 59
2.43.2. Maven coordinates 59
2.43.3. Additional Camel Quarkus configuration 59

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. EXTENSIONS OVERVIEW

1.1. SUPPORT LEVEL DEFINITIONS

New features, services, and components go through a number of support levels before inclusion in Camel Extensions for Quarkus as fully supported for production use. This is to ensure the right balance between providing the enterprise stability expected of our offerings with the need to allow our customers and partners to experiment with new Camel Extensions for Quarkus technologies while providing feedback to help guide future development activities.

Table 1.1. Camel Extensions for Quarkus support levels

Type	Description
Community Support	As part of Red Hat's commitment to upstream first, integration of new extensions into our Camel Extensions for Quarkus distribution begins in the upstream community. While these extensions have been tested and documented upstream, we have not reviewed the maturity of these extensions and they may not be formally supported by Red Hat in future product releases. **NOTE** Community extensions are listed on the extensions reference page of the Camel Quarkus community project.
Technology Preview	Technology Preview features provide early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. However, these features are not fully supported under Red Hat Subscription Level Agreements, may not be functionally complete, and are not intended for production use. As Red Hat considers making future iterations of Technology Preview features generally available, we will attempt to resolve any issues that customers experience when using these features.
Production Support	Production Support extensions are shipped in a formal Red Hat release and are fully supported. There are no documentation gaps and extensions have been tested on all supported configurations.

1.2. SUPPORTED EXTENSIONS

There are 33 extensions.

Table 1.2. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
AWS 2 DynamoDB	camel-quarkus-aws2-ddb	Production Support	Technology Preview	Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.
AWS 2 Kinesis	camel-quarkus-aws2-kinesis	Production Support	Technology Preview	Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.
AWS 2 Lambda	camel-quarkus-aws2-lambda	Production Support	Technology Preview	Manage and invoke AWS Lambda functions using AWS SDK version 2.x.
AWS 2 S3 Storage Service	camel-quarkus-aws2-s3	Production Support	Technology Preview	Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.
AWS 2 Simple Notification System (SNS)	camel-quarkus-aws2-sns	Production Support	Technology Preview	Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.
AWS 2 Simple Queue Service (SQS)	camel-quarkus-aws2-sqs	Technology Preview	Technology Preview	Sending and receive messages to/from AWS SQS service using AWS SDK version 2.x.
Bean	camel-quarkus-bean	Production Support	Technology Preview	Invoke methods of Java beans

Extension	Artifact	JVM Support Level	Native Support Level	Description
Core	camel-quarkus-core	Production Support	Technology Preview	Camel core functionality and basic Camel languages: Constant, ExchangeProperty , Header, Ref, Ref, Simple and Tokeinze
Direct	camel-quarkus-direct	Production Support	Technology Preview	Call another endpoint from the same Camel Context synchronously.
Elasticsearch Rest	camel-quarkus-elasticsearch-rest	Technology Preview	Technology Preview	Send requests to with an ElasticSearch via REST API.
File	camel-quarkus-file	Production Support	Technology Preview	Read and write files.
FTP	camel-quarkus-ftp	Production Support	Technology Preview	Upload and download files to/from FTP or SFTP servers.
HTTP	camel-quarkus-http	Production Support	Technology Preview	Send requests to external HTTP servers using Apache HTTP Client 4.x.
Jira	camel-quarkus-jira	Technology Preview	Technology Preview	Interact with JIRA issue tracker.
JMS	camel-quarkus-jms	Production Support	Technology Preview	Send and receive messages to/from a JMS Queue or Topic.

Extension	Artifact	JVM Support Level	Native Support Level	Description
JTA	camel-quarkus-jta	Production Support	Technology Preview	Enclose Camel routes in the transactions using Java Transaction API (JTA) and Narayana transaction manager
Kafka	camel-quarkus-kafka	Production Support	Technology Preview	Sent and receive messages to/from an Apache Kafka broker.
Kamelet	camel-quarkus-kamelet	Production Support	Technology Preview	The Kamelet Component provides support for interacting with the Camel Route Template engine.
Log	camel-quarkus-log	Production Support	Technology Preview	Log messages to the underlying logging mechanism.
MicroProfile Health	camel-quarkus-microprofile-health	Production Support	Technology Preview	Bridging Eclipse MicroProfile Health with Camel health checks.
MicroProfile Metrics	camel-quarkus-microprofile-metrics	Production Support	Technology Preview	Expose metrics from Camel routes.
MLLP	camel-quarkus-mllp	Production Support	Technology Preview	Communicate with external systems using the MLLP protocol.
Mock	camel-quarkus-mock	Production Support	Technology Preview	Test routes and mediation rules using mocks.

Extension	Artifact	JVM Support Level	Native Support Level	Description
MongoDB	camel-quarkus-mongodb	Technology Preview	Technology Preview	Perform operations on MongoDB documents and collections.
Netty	camel-quarkus-netty	Production Support	Technology Preview	Socket level networking using TCP or UDP with the Netty 4.x.
OpenAPI Java	camel-quarkus-openapi-java	Production Support	Technology Preview	Rest-dsl support for using OpenAPI doc
Platform HTTP	camel-quarkus-platform-http	Production Support	Technology Preview	Expose HTTP endpoints using the HTTP server available in the current platform.
Rest	camel-quarkus-rest	Production Support	Technology Preview	Expose REST services and their OpenAPI Specification or call external REST services.
Salesforce	camel-quarkus-salesforce	Production Support	Technology Preview	Communicate with Salesforce using Java DTOs.
SEDA	camel-quarkus-seda	Production Support	Technology Preview	Asynchronously call another endpoint from any Camel Context in the same JVM.
SQL	camel-quarkus-sql	Production Support	Technology Preview	Perform SQL queries using Spring JDBC.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Timer	camel-quarkus-timer	Production Support	Technology Preview	Generate messages in specified intervals using java.util.Timer.
XQuery	camel-quarkus-saxon	Production Support	Technology Preview	Query and/or transform XML payloads using XQuery and Saxon.

1.3. SUPPORTED DATA FORMATS

There are 8 data formats.

Table 1.3. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
Avro	camel-quarkus-avro	Production Support	Technology Preview	Serialize and deserialize messages using Apache Avro binary data format.
Avro Jackson	camel-quarkus-jackson-avro	Production Support	Technology Preview	Marshal POJOs to Avro and back using Jackson.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Bindy	camel-quarkus-bindy	Production Support	Technology Preview	Marshal and unmarshal between POJOs and Comma separated values (CSV) format using Camel Bindy Marshal and unmarshal between POJOs and fixed field length format using Camel Bindy Marshal and unmarshal between POJOs and key-value pair (KVP) format using Camel Bindy
HL7	camel-quarkus-hl7	Production Support	Technology Preview	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
Jackson	camel-quarkus-jackson	Production Support	Technology Preview	Marshal POJOs to JSON and back using Jackson
JacksonXML	camel-quarkus-jacksonxml	Production Support	Technology Preview	Unmarshal a XML payloads to POJOs and back using XMLMapper extension of Jackson.
Protobuf Jackson	camel-quarkus-jackson-protobuf	Production Support	Technology Preview	Marshal POJOs to Protobuf and back using Jackson.
SOAP dataformat	camel-quarkus-soap	Production Support	Technology Preview	Marshal Java objects to SOAP messages and back.

1.4. SUPPORTED LANGUAGES

There are 12 languages.

Table 1.4. Camel Extensions for Quarkus Support Matrix

Extension	Artifact	JVM Support Level	Native Support Level	Description
Bean method	camel-quarkus-bean	Production Support	Technology Preview	Invoke methods of Java beans
Constant	camel-quarkus-core	Production Support	Technology Preview	A fixed value set only once during the route startup.
ExchangeProperty	camel-quarkus-core	Production Support	Technology Preview	Get the value of named Camel Exchange property.
File	camel-quarkus-core	Production Support	Technology Preview	For expressions and predicates using the file/simple language.
Header	camel-quarkus-core	Production Support	Technology Preview	Get the value of the named Camel Message header.
HL7 Terser	camel-quarkus-hl7	Production Support	Technology Preview	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
Ref	camel-quarkus-core	Production Support	Technology Preview	Look up an expression in the Camel Registry and evaluate it.
Simple	camel-quarkus-core	Production Support	Technology Preview	Evaluate Camel's built-in Simple language expression against the Camel Exchange.
Tokenize	camel-quarkus-core	Production Support	Technology Preview	Tokenize text payloads using the specified delimiter patterns.

Extension	Artifact	JVM Support Level	Native Support Level	Description
JSON Path	camel-quarkus-jsonpath	Production Support	Technology Preview	Evaluate a JsonPath expression against a JSON message body.
XPath	camel-quarkus-xpath	Production Support	Technology Preview	Evaluate an XPath expression against an XML payload.
XQuery	camel-quarkus-saxon	Production Support	Technology Preview	Query and/or transform XML payloads using XQuery and Saxon.

CHAPTER 2. EXTENSIONS REFERENCE

This chapter provides reference information about Camel Extensions for Quarkus.

IMPORTANT

This Technology Preview release includes a targeted subset of the available Camel Quarkus extensions. Additional extensions will be added to our Camel Extensions for Quarkus distribution in future releases.

2.1. AVRO

Serialize and deserialize messages using Apache Avro binary data format.

2.1.1. What's inside

- Avro data format

Please refer to the above link for usage and configuration details.

2.1.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-avro</artifactId>
</dependency>
```

2.1.3. Additional Camel Quarkus configuration

Beyond standard usages known from vanilla Camel, Camel Quarkus adds the possibility to parse the Avro schema at build time both in JVM and Native mode.

Since Camel Quarkus 2.0.0, the preferred approach to generate Avro classes from Avro schema files is the one coined by the **quarkus-avro** extension. It requires the following:

1. Store ***.avsc** files in a folder named **src/main/avro** or **src/test/avro**
2. In addition to the usual **build** goal of **quarkus-maven-plugin**, add the **generate-code** goal:

```
<plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>generate-code-and-build</id>
            <goals>
                <goal>generate-code</goal>
                <goal>build</goal>
            </goals>
            </execution>
        </executions>
    </plugin>
```

Please see a working configuration in Camel Quarkus Avro integration test and Quarkus Avro integration test.

2.1.3.1. Deprecated: @BuildTimeAvroDataFormat annotation.

Before Camel Quarkus 2.0.0, the **@BuildTimeAvroDataFormat** annotation was the preferred way to generate Avro entities from Avro schema files.

In the example below, the **user.avsc** schema resource is first parsed at build time. Then, an **AvroDataFormat** instance using the previously parsed schema is injected in the **buildTimeAvroDataFormat** field at runtime. At the end of the day, the injected data format is used from the **configure()** method in order to marshal an incoming message.

```
import org.apache.camel.quarkus.component.avro.BuildTimeAvroDataFormat;
...
@BuildTimeAvroDataFormat("user.avsc")
AvroDataFormat buildTimeAvroDataFormat;

@Override
public void configure() {
  from("direct:marshalUsingBuildTimeAvroDataFormat").marshal(buildTimeAvroDataFormat);
}
```

Since Camel Quarkus 2.0.0, @BuildTimeAvroDataFormat is deprecated. The build time class generation approach from quarkus-avro is preferred. As such, it is advised to store *.avsc files in a folder named 'avro' to have @AvroGenerated class created at build-time by quarkus-avro.

Please see a running configuration at work in the Camel Quarkus Avro integration tests. There is also a quarkus-avro integration test here.

2.2. AWS 2 DYNAMODB

Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.

2.2.1. What's inside

- AWS DynamoDB component, URI syntax: **aws2-ddb:tableName**
- AWS DynamoDB Streams component, URI syntax: **aws2-ddbstream:tableName**

Please refer to the above links for usage and configuration details.

2.2.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
```

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

2.2.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.2.4. Additional Camel Quarkus configuration

2.2.4.1. Optional integration with Quarkus Amazon DynamoDB

If desired, it is possible to use the Quarkus Amazon DynamoDB extension in conjunction with Camel Quarkus AWS 2 DynamoDB. Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

   ```
   quarkus.dynamodb.sync-client.type=apache
   ```

2. The **DynamoDbClient** has to be "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that, for example, by adding a dummy bean injecting **DynamoDbClient**:

   ```
   import javax.enterprise.context.ApplicationScoped;
   import io.quarkus.arc.Unremovable;
   import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

   @ApplicationScoped
   @Unremovable
   class UnremovableDynamoDbClient {
       @Inject
       DynamoDbClient dynamoDbClient;
   }
   ```

2.3. AWS 2 KINESIS

Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.

2.3.1. What's inside

- AWS Kinesis component, URI syntax: **aws2-kinesis:streamName**
- AWS Kinesis Firehose component, URI syntax: **aws2-kinesis-firehose:streamName**

Please refer to the above links for usage and configuration details.

2.3.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.3.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.4. AWS 2 LAMBDA

Manage and invoke AWS Lambda functions using AWS SDK version 2.x.

2.4.1. What's inside

- AWS Lambda component, URI syntax: **aws2-lambda:function**

Please refer to the above link for usage and configuration details.

2.4.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.4.3. Camel Quarkus limitations

The **getAlias** and **listAliases** operations need to be used with **pojoRequest** in order to work. It implies that the request for those operations should be explicitly constructed by hand as shown below.

Example of creating a **getAlias** request by hand:

```
.process(new Processor() {
   public void process(Exchange exchange) {
     GetAliasRequest getAliasRequest =
GetAliasRequest.builder().functionName(functionName).name(aliasName).build();
     exchange.getIn().setBody(getAliasRequest);
  }})
.to("aws2-lambda:functionName?operation=getAlias&pojoRequest=true");
```

Example of creating a **listAliases** request by hand:

```
.process(new Processor() {
   public void process(Exchange exchange) {
     ListAliasesRequest listAliasesRequest =
ListAliasesRequest.builder().functionName(functionName).build();
```

```
        exchange.getIn().setBody(listAliasesRequest);
    }})
  .to("aws2-lambda:functionName?operation=listAliases&pojoRequest=true");
```

2.4.4. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.4.5. Additional Camel Quarkus configuration

2.4.5.1. Not possible to leverage quarkus-amazon-lambda by Camel aws2-lambda extension

The **quarkus-amazon-lambda** extension allows you to use Quarkus to build your AWS Lambdas, whereas Camel component manages (deploy, undeploy, …) existing functions. Therefore, it is not possible to use **quarkus-amazon-lambda** as a client for Camel **aws2-lambda** extension.

2.5. AWS 2 S3 STORAGE SERVICE

Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.

2.5.1. What's inside

- AWS S3 Storage Service component, URI syntax: **aws2-s3://bucketNameOrArn**

Please refer to the above link for usage and configuration details.

2.5.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.5.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.5.4. Additional Camel Quarkus configuration

2.5.4.1. Optional integration with Quarkus Amazon S3

If desired, it is possible to use the Quarkus Amazon S3 extension in conjunction with Camel Quarkus AWS 2 S3 Storage Service. Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

```
quarkus.s3.sync-client.type=apache
```

2. The **S3Client** has to be "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that, for example, by adding a dummy bean injecting **S3Client**:

```
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.s3.S3Client;

@ApplicationScoped
@Unremovable
class UnremovableS3Client {
    @Inject
    S3Client s3Client;
}
```

2.6. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)

Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.

2.6.1. What's inside

- AWS Simple Notification System (SNS) component , URI syntax: **aws2-sns:topicNameOrArn**

Please refer to the above link for usage and configuration details.

2.6.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.6.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.6.4. Additional Camel Quarkus configuration

2.6.4.1. Optional integration with Quarkus Amazon SNS

If desired, it is possible to use the Quarkus Amazon SNS extension in conjunction with Camel Quarkus AWS 2 Simple Notification System (SNS). Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

```
quarkus.sns.sync-client.type=apache
```

2. The **SnsClient** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **SnsClient**:

```
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.sns.SnsClient;

@ApplicationScoped
@Unremovable
class UnremovableSnsClient {
    @Inject
    SnsClient snsClient;
}
```

2.7. AWS 2 SIMPLE QUEUE SERVICE (SQS)

Sending and receive messages to/from AWS SQS service using AWS SDK version 2.x.

2.7.1. What's inside

- AWS Simple Queue Service (SQS) component, URI syntax: **aws2-sqs:queueNameOrArn**

Please refer to the above link for usage and configuration details.

2.7.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.7.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.7.4. Additional Camel Quarkus configuration

2.7.4.1. Optional integration with Quarkus Amazon SQS

If desired, it is possible to use the Quarkus Amazon SQS extension in conjunction with Camel Quarkus AWS 2 Simple Queue Service (SQS). Note that this is fully optional and not mandatory at all. Please follow the Quarkus documentation but beware of the following caveats:

1. The client type **apache** has to be selected by configuring the following property:

```
quarkus.sqs.sync-client.type=apache
```

2. The **SqsClient** has to be made "unremovable" in the sense of Quarkus CDI reference so that Camel Quarkus is able to look it up at runtime. You can reach that e.g. by adding a dummy bean injecting **SqsClient**:

```
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.arc.Unremovable;
import software.amazon.awssdk.services.sqs.SqsClient;

@ApplicationScoped
@Unremovable
class UnremovableSqsClient {
    @Inject
    SqsClient sqsClient;
}
```

2.8. BEAN

Invoke methods of Java beans

2.8.1. What's inside

- Bean component, URI syntax: **bean:beanName**
- Bean method language
- Class component, URI syntax: **class:beanName**

Please refer to the above links for usage and configuration details.

2.8.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

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

2.8.3. Usage

Except for invoking methods of beans available in Camel registry, Bean component and Bean method language can also invoke Quarkus CDI beans.

2.9. BINDY

Marshal and unmarshal between POJOs on one side and Comma separated values (CSV), fixed field length or key-value pair (KVP) formats on the other side using Camel Bindy

2.9.1. What’s inside

- Bindy CSV data format
- Bindy Fixed Length data format
- Bindy Key Value Pair data format

Please refer to the above links for usage and configuration details.

2.9.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-bindy</artifactId>
</dependency>
```

2.9.3. Camel Quarkus limitations

When using camel-quarkus-bindy in native mode, only the build machine’s locale is supported.

For instance, on build machines with french locale, the code below:

```
BindyDataFormat dataFormat = new BindyDataFormat();
dataFormat.setLocale("ar");
```

formats numbers the arabic way in JVM mode as expected. However, it formats numbers the french way in native mode.

Without further tuning, the build machine’s default locale would be used. Another locale could be specified with the quarkus.native.user-language and quarkus.native.user-country configuration properties.

2.10. CORE

Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Ref, Simple and Tokenize

2.10.1. What’s inside

- Constant language
- ExchangeProperty language
- File language
- Header language
- Ref language

- Simple language
- Tokenize language

Please refer to the above links for usage and configuration details.

2.10.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-core</artifactId>
</dependency>
```

2.10.3. Additional Camel Quarkus configuration

2.10.3.1. Simple language

2.10.3.1.1. Using the OGNL notation

When using the OGNL notation from the simple language, the **camel-quarkus-bean** extension should be used.

For instance, the simple expression below is accessing the **getAddress()** method on the message body of type **Client**.

```
---
simple("${body.address}")
---
```

In such a situation, one should take an additional dependency on the camel-quarkus-bean extension as described here. Note that in native mode, some classes may need to be registered for reflection. In the example above, the **Client** class needs to be registered for reflection.

2.10.3.1.2. Using dynamic type resolution in native mode

When dynamically resolving a type from simple expressions like **${mandatoryBodyAs(TYPE)}**, **${type:package.Enum.CONSTANT}** or **${body} is TYPE**, it may be needed to register some classes for reflection manually.

For instance, the simple expression below is dynamically resolving the type **java.nio.ByteBuffer** at runtime:

```
---
simple("${body} is 'java.nio.ByteBuffer'")
---
```

As such, the class **java.nio.ByteBuffer** needs to be registered for reflection.

2.10.3.1.3. Using the simple language with classpath resources in native mode

If your route is supposed to load a Simple script from classpath, like in the following example

```
from("direct:start").transform().simple("resource:classpath:mysimple.txt");
```

then you need to use Quarkus **quarkus.native.resources.includes** property to include the resource in the native executable as demonstrated below:

```
quarkus.native.resources.includes = mysimple.txt
```

2.10.3.1.4. Configuring a custom bean via properties in native mode

When specifying a custom bean via properties in native mode with configuration like **#class:*** or **#type:***, it may be needed to register some classes for reflection manually.

For instance, the custom bean definition below involves the use of reflection for bean instantiation and setter invocation:

```
---
camel.beans.customBeanWithSetterInjection =
#class:org.example.PropertiesCustomBeanWithSetterInjection
camel.beans.customBeanWithSetterInjection.counter = 123
---
```

As such, the class **PropertiesCustomBeanWithSetterInjection** needs to be registered for reflection, note that field access could be omitted in this case.

Configuration property	Type	Default
🔒 quarkus.camel.bootstrap.enabled When set to true, the **CamelRuntime** will be started automatically.	**boolean**	**true**
🔒 quarkus.camel.service.discovery.exclude-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will not be discoverable via the **org.apache.camel.spi.FactoryFinder** mechanism. The excludes have higher precedence than includes. The excludes defined here can also be used to veto the discoverability of services included by Camel Quarkus extensions. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	

Configuration property	Type	Default
🔒 quarkus.camel.service.discovery.include-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will be discoverable via the **org.apache.camel.spi.FactoryFinder** mechanism unless the given file is excluded via **exclude-patterns**. Note that Camel Quarkus extensions may include some services by default. The services selected here added to those services and the exclusions defined in **exclude-patterns** are applied to the union set. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	
🔒 quarkus.camel.service.registry.exclude-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will not be added to Camel registry during application’s static initialization. The excludes have higher precedence than includes. The excludes defined here can also be used to veto the registration of services included by Camel Quarkus extensions. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	
🔒 quarkus.camel.service.registry.include-patterns A comma-separated list of Ant-path style patterns to match Camel service definition files in the classpath. The services defined in the matching files will be added to Camel registry during application’s static initialization unless the given file is excluded via **exclude-patterns**. Note that Camel Quarkus extensions may include some services by default. The services selected here added to those services and the exclusions defined in **exclude-patterns** are applied to the union set. Example values: **META-INF/services/org/apache/camel/foo/*,META-INF/services/org/apache/camel/foo/**/bar**	**string**	

Configuration property	Type	Default
🔒 **quarkus.camel.runtime-catalog.components** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel components available in the application; otherwise component JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	**boolean**	**true**
🔒 **quarkus.camel.runtime-catalog.languages** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel languages available in the application; otherwise language JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	**boolean**	**true**
🔒 **quarkus.camel.runtime-catalog.dataformats** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel data formats available in the application; otherwise data format JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	**boolean**	**true**
🔒 **quarkus.camel.runtime-catalog-models** If **true** the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel EIP models available in the application; otherwise EIP model JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **false** except for making the behavior consistent with native mode.	**boolean**	**true**
🔒 **quarkus.camel.routes-discovery.enabled** Enable automatic discovery of routes during static initialization.	**boolean**	**true**

Configuration property	Type	Default
🔒 **quarkus.camel.routes-discovery.exclude-patterns** Used for exclusive filtering scanning of RouteBuilder classes. The exclusive filtering takes precedence over inclusive filtering. The pattern is using Ant-path style pattern. Multiple patterns can be specified separated by comma. For example to exclude all classes starting with Bar use: **/Bar* To exclude all routes form a specific package use: com/mycompany/bar/* To exclude all routes form a specific package and its sub-packages use double wildcards: com/mycompany/bar/** And to exclude all routes from two specific packages use: com/mycompany/bar/*,com/mycompany/stuff/*	**string**	
🔒 **quarkus.camel.routes-discovery.include-patterns** Used for inclusive filtering scanning of RouteBuilder classes. The exclusive filtering takes precedence over inclusive filtering. The pattern is using Ant-path style pattern. Multiple patterns can be specified separated by comma. For example to include all classes starting with Foo use: **/Foo* To include all routes form a specific package use: com/mycompany/foo/* To include all routes form a specific package and its sub-packages use double wildcards: com/mycompany/foo/** And to include all routes from two specific packages use: com/mycompany/foo/*,com/mycompany/stuff/*	**string**	
🔒 **quarkus.camel.native.resources.exclude-patterns** Replaced by **quarkus.native.resources.excludes** in Camel Quarkus 2.0.0. Using this property throws an exception at build time.	**string**	
🔒 **quarkus.camel.native.resources.include-patterns** Replaced by **quarkus.native.resources.includes** in Camel Quarkus 2.0.0. Using this property throws an exception at build time.	**string**	
🔒 **quarkus.camel.native.reflection.exclude-patterns** A comma separated list of Ant-path style patterns to match class names that should be excluded from registering for reflection. Use the class name format as returned by the **java.lang.Class.getName()** method: package segments delimited by period **.** and inner classes by dollar sign **$**. This option narrows down the set selected by **include-patterns**. By default, no classes are excluded. This option cannot be used to unregister classes which have been registered internally by Quarkus extensions.	**string**	

Configuration property	Type	Default
🔒 **quarkus.camel.native.reflection.include-patterns** A comma separated list of Ant-path style patterns to match class names that should be registered for reflection. Use the class name format as returned by the **java.lang.Class.getName()** method: package segments delimited by period **.** and inner classes by dollar sign **$**. By default, no classes are included. The set selected by this option can be narrowed down by **exclude-patterns**. Note that Quarkus extensions typically register the required classes for reflection by themselves. This option is useful in situations when the built in functionality is not sufficient. Note that this option enables the full reflective access for constructors, fields and methods. If you need a finer grained control, consider using **io.quarkus.runtime.annotations.RegisterForReflection** annotation in your Java code. For this option to work properly, at least one of the following conditions must be satisfied: - There are no wildcards (***** or **/**) in the patterns - The artifacts containing the selected classes contain a Jandex index (**META-INF/jandex.idx**) - The artifacts containing the selected classes are registered for indexing using the **quarkus.index-dependency.*** family of options in**application.properties** - e.g. quarkus.index-dependency.my-dep.group-id = org.my-group quarkus.index-dependency.my-dep.artifact-id = my-artifact where **my-dep** is a label of your choice to tell Quarkus that **org.my-group** and with **my-artifact** belong together.	**string**	
🔒 **quarkus.camel.native.reflection.serialization-enabled** If **true**, basic classes are registered for serialization; otherwise basic classes won't be registered automatically for serialization in native mode. The list of classes automatically registered for serialization can be found in CamelSerializationProcessor.BASE_SERIALIZATION_CLASSES. Setting this to **false** helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to **true** except for making the behavior consistent with native mode.	**boolean**	**false**
🔒 **quarkus.camel.csimple.on-build-time-analysis-failure** What to do if it is not possible to extract CSimple expressions from a route definition at build time.	**org.apache.camel.quarkus.core.CamelConfig.FailureRemedy**	**warn**

Configuration property	Type	Default
🔒 quarkus.camel.event-bridge.enabled Whether to enable the bridging of Camel events to CDI events. This allows CDI observers to be configured for Camel events. E.g. those belonging to the **org.apache.camel.quarkus.core.events**, **org.apache.camel.quarkus.main.events** & **org.apache.camel.impl.event** packages. Note that this configuration item only has any effect when observers configured for Camel events are present in the application.	**boolean**	**true**
🔒 quarkus.camel.main.enabled If **true** all **camel-main** features are enabled; otherwise no **camel-main** features are enabled.	**boolean**	**true**
🔒 quarkus.camel.main.shutdown.timeout A timeout (with millisecond precision) to wait for **CamelMain#stop()** to finish	**java.time.Duration**	**PT3S**
🔒 quarkus.camel.main.arguments.on-unknown The action to take when **CamelMain** encounters an unknown argument. fail - Prints the **CamelMain** usage statement and throws a **RuntimeException** ignore - Suppresses any warnings and the application startup proceeds as normal warn - Prints the **CamelMain** usage statement but allows the application startup to proceed as normal	**org.apache.camel.quarkus.core.CamelConfig.FailureRemedy**	**warn**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.11. DIRECT

Call another endpoint from the same Camel Context synchronously.

2.11.1. What's inside

- Direct component, URI syntax: **direct:name**

Please refer to the above link for usage and configuration details.

2.11.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-direct</artifactId>
</dependency>
```

2.12. ELASTICSEARCH REST

Send requests to with an ElasticSearch via REST API.

2.12.1. What's inside

- Elasticsearch Rest component, URI syntax: **elasticsearch-rest:clusterName**

Please refer to the above link for usage and configuration details.

2.12.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-elasticsearch-rest</artifactId>
</dependency>
```

2.12.3. Usage

This extension leverages the Quarkus ElasticSearch REST Client.

You can choose to configure ElasticSearch via the Quarkus configuration properties and the **RestClient** will be autowired into the Camel ElasticSearch component.

Or you can configure ElasticSearch via the Camel ElasticSearch component / endpoint options. When doing this, you must disable autowiring in one of the ways outlined below.

Disabling autowiring at the component level.

```
camel.component.elasticsearch-rest.autowired-enabled = false
```

Disabling autowiring at the endpoint level.

```
from("direct:search")
    .to("elasticsearch-rest://elasticsearch?
hostAddresses=localhost:9200&operation=Search&indexName=index&autowiredEnabled=false")
```

Globally disabling autowiring. Note that this disables autowiring for all components.

```
camel.main.autowired-enabled = false
```

2.13. FILE

Read and write files.

2.13.1. What's inside

- File component, URI syntax: **file:directoryName**

Please refer to the above link for usage and configuration details.

2.13.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-file</artifactId>
</dependency>
```

2.14. FTP

Upload and download files to/from SFTP, FTP or SFTP servers

2.14.1. What's inside

- FTP component, URI syntax: **ftp:host:port/directoryName**
- FTPS component, URI syntax: **ftps:host:port/directoryName**
- SFTP component, URI syntax: **sftp:host:port/directoryName**

Please refer to the above links for usage and configuration details.

2.14.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-ftp</artifactId>
</dependency>
```

2.15. HL7

Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.

2.15.1. What's inside

- HL7 data format

- HL7 Terser language

Please refer to the above links for usage and configuration details.

2.15.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-hl7</artifactId>
</dependency>
```

2.15.3. Camel Quarkus limitations

For MLLP with TCP, Netty is the only supported means of running an Hl7 MLLP listener. Mina is not supported since it has no GraalVM native support at present.

Optional support for **HL7MLLPNettyEncoderFactory** & **HL7MLLPNettyDecoderFactory** codecs can be obtained by adding a dependency in your project **pom.xml** to **camel-quarkus-netty**.

2.16. HTTP

Send requests to external HTTP servers using Apache HTTP Client 4.x.

2.16.1. What's inside

- HTTP component, URI syntax: **http://httpUri**
- HTTPS (Secure) component, URI syntax: **https://httpUri**

Please refer to the above links for usage and configuration details.

2.16.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-http</artifactId>
</dependency>
```

2.16.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.16.4. transferException option in native mode

To use the **transferException** option in native mode, you must enable support for object serialization. For more information, refer to the Registering Classes for Serialization section in the *Developing Applications with Camel Extensions for Quarkus* guide.

You will also need to enable serialization for the exception classes that you intend to serialize. For example.

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

2.16.5. Additional Camel Quarkus configuration

- Check the Character Encodings section of the *Developing Applications with Camel Extensions for Quarkus* guide if you expect your application to send or receive requests using non-default encodings.

2.17. JACKSON

Marshal POJOs to JSON and back using Jackson

2.17.1. What's inside

- JSON Jackson data format

Please refer to the above link for usage and configuration details.

2.17.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson</artifactId>
</dependency>
```

2.18. AVRO JACKSON

Marshal POJOs to Avro and back using Jackson.

2.18.1. What's inside

- Avro Jackson data format

Please refer to the above link for usage and configuration details.

2.18.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-avro</artifactId>
</dependency>
```

2.19. PROTOBUF JACKSON

Marshal POJOs to Protobuf and back using Jackson.

2.19.1. What's inside

- Protobuf Jackson data format

Please refer to the above link for usage and configuration details.

2.19.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jackson-protobuf</artifactId>
</dependency>
```

2.20. JACKSONXML

Unmarshal a XML payloads to POJOs and back using XMLMapper extension of Jackson.

2.20.1. What's inside

- JacksonXML data format

Please refer to the above link for usage and configuration details.

2.20.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jacksonxml</artifactId>
</dependency>
```

2.21. JIRA

Interact with JIRA issue tracker.

2.21.1. What’s inside

- Jira component, URI syntax: **jira:type**

Please refer to the above link for usage and configuration details.

2.21.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jira</artifactId>
</dependency>
```

2.21.3. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.22. JMS

Sent and receive messages to/from a JMS Queue or Topic.

2.22.1. What’s inside

- JMS component, URI syntax: **jms:destinationType:destinationName**

Please refer to the above link for usage and configuration details.

2.22.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jms</artifactId>
</dependency>
```

2.22.3. Usage

2.22.3.1. Message mapping with org.w3c.dom.Node

The Camel JMS component supports message mapping between **javax.jms.Message** and **org.apache.camel.Message**. When wanting to convert a Camel message body type of **org.w3c.dom.Node**, you must ensure that the **camel-quarkus-jaxp** extension is present on the classpath.

2.22.3.2. Native mode support for javax.jms.ObjectMessage

When sending JMS message payloads as **javax.jms.ObjectMessage**, you must annotate the relevant classes to be registered for serialization with **@RegisterForReflection(serialization = true)**. Note that this extension automatically sets **quarkus.camel.native.reflection.serialization-enabled = true** for you. Refer to the native mode user guide for more information.

2.22.4. transferException option in native mode

To use the **transferException** option in native mode, you must enable support for object serialization. Refer to the native mode user guide for more information.

You will also need to enable serialization for the exception classes that you intend to serialize. For example.

```
@RegisterForReflection(targets = { IllegalStateException.class, MyCustomException.class },
serialization = true)
```

2.23. JSON PATH

Evaluates a JsonPath expression against a JSON message body.

2.23.1. What's inside

- JsonPath language

Please refer to the above link for usage and configuration details.

2.23.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jsonpath</artifactId>
</dependency>
```

2.24. JTA

Enclose Camel routes in transactions using Java Transaction API (JTA) and Narayana transaction manager

2.24.1. What's inside

- JTA

Please refer to the above link for usage and configuration details.

2.24.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-jta</artifactId>
</dependency>
```

2.24.3. Usage

This extension should be added when you need to use the **transacted()** EIP in the router. It leverages the transaction capabilities provided by the narayana-jta extension in Quarkus.

Refer to the Quarkus Transaction guide for the more details about transaction support. For a simple usage:

```
from("direct:transaction")
    .transacted()
    .to("sql:INSERT INTO A TABLE ...?dataSource=ds1")
    .to("sql:INSERT INTO A TABLE ...?dataSource=ds2")
    .log("all data are in the ds1 and ds2")
```

Support is provided for various transaction policies.

Policy	Description
PROPAGATION_MANDATORY	Support a current transaction; throw an exception if no current transaction exists.
PROPAGATION_NEVER	Do not support a current transaction; throw an exception if a current transaction exists.
PROPAGATION_NOT_SUPPORTED	Do not support a current transaction; rather always execute non-transactionally.
PROPAGATION_REQUIRED	Support a current transaction; create a new one if none exists.
PROPAGATION_REQUIRES_NEW	Create a new transaction, suspending the current transaction if one exists.
PROPAGATION_SUPPORTS	Support a current transaction; execute non-transactionally if none exists.

2.25. KAFKA

Sent and receive messages to/from an Apache Kafka broker.

2.25.1. What's inside

- Kafka component, URI syntax: **kafka:topic**

Please refer to the above link for usage and configuration details.

2.25.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-kafka</artifactId>
</dependency>
```

2.25.3. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.kafka.kubernetes-service-binding.merge-configuration If **true** then any Kafka configuration properties discovered by the Quarkus Kubernetes Service Binding extension (if configured) will be merged with those set via Camel Kafka component or endpoint options. If **false** then any Kafka configuration properties discovered by the Quarkus Kubernetes Service Binding extension are ignored, and all of the Kafka component configuration is driven by Camel.	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.26. KAMELET

Materialize route templates

2.26.1. What's inside

- Kamelet component, URI syntax: **kamelet:templateId/routeId**

Please refer to the above link for usage and configuration details.

2.26.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-kamelet</artifactId>
</dependency>
```

2.26.3. Usage

2.26.3.1. Pre-load Kamelets at build-time

This extension allows you to pre-load a set of Kamelets at build time using the **quarkus.camel.kamelet.identifiers** property.

2.26.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.kamelet.identifiers** List of kamelets identifiers to pre-load at build time. Each individual identifier is used to set the related **org.apache.camel.model.RouteTemplateDefinition** id.	**string**	

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.27. LOG

Log messages to the underlying logging mechanism.

2.27.1. What's inside

- Log component, URI syntax: **log:loggerName**

Please refer to the above link for usage and configuration details.

2.27.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-log</artifactId>
</dependency>
```

2.28. MICROPROFILE HEALTH

Bridging Eclipse MicroProfile Health with Camel health checks

2.28.1. What's inside

- Microprofile Health

Please refer to the above link for usage and configuration details.

2.28.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-health</artifactId>
</dependency>
```

2.28.3. Usage

By default, classes extending **AbstractHealthCheck** are registered as both liveness and readiness checks. You can override the **isReadiness** method to control this behaviour.

Any checks provided by your application are automatically discovered and bound to the Camel registry. They will be available via the Quarkus health endpoints **/q/health/live** and **/q/health/ready**.

You can also provide custom **HealthCheckRepository** implementations and these are also automatically discovered and bound to the Camel registry for you.

Refer to the Quarkus health guide for further information.

2.28.3.1. Provided health checks

Some checks are automatically registered for your application.

2.28.3.1.1. Camel Context Health

Inspects the Camel Context status and causes the health check status to be **DOWN** if the status is anything other than 'Started'.

2.28.3.1.2. Camel Route Health

Inspects the status of each route and causes the health check status to be **DOWN** if any route status is not 'Started'.

2.28.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.health.enabled** Set whether to enable Camel health checks	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.29. MICROPROFILE METRICS

Expose metrics from Camel routes.

2.29.1. What's inside

- MicroProfile Metrics component, URI syntax: **microprofile-metrics:metricType:metricName**

Please refer to the above link for usage and configuration details.

2.29.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-microprofile-metrics</artifactId>
</dependency>
```

2.29.3. Usage

The microprofile-metrics component automatically exposes a set of Camel application metrics. Some of these include:

2.29.3.1. Camel Context metrics

Metric Name	Type
camel.context.status The status of the Camel Context represented by the **ServiceStatus** enum ordinal	Gauge
camel.context.uptime The Camel Context uptime in milliseconds	Gauge
camel.context.exchanges.completed.total The total number of completed exchanges	Counter
camel.context.exchanges.failed.total The total number of failed exchanges	Counter

Metric Name	Type
camel.context.exchanges.inflight.total The total number of inflight exchanges	Gauge
camel.context.exchanges.total The total number of all exchanges	Counter
camel.context.externalRedeliveries.total The total number of all external redeliveries	Counter
camel.context.failuresHandled.total The total number of all failures handled	Counter

2.29.3.2. Camel Route metrics

Metric Name	Type
camel.route.count The number of routes	Gauge
camel.route.running.count The number of running routes	Gauge
camel.route.exchanges.completed.total The total number of completed exchanges for the route	Counter
camel.route.exchanges.failed.total The total number of failed exchanges for the route	Counter
camel.route.exchanges.inflight.total The total number of inflight exchanges for the route	Gauge
camel.route.exchanges.total The total number of all exchanges for the route	Counter
camel.route.externalRedeliveries.total The total number of all external redeliveries for the route	Counter

Metric Name	Type
camel.route.failuresHandled.total The total number of all failures handled for the route	Counter

All metrics are tagged with the name of the Camel Context and the id of the route where applicable.

You can also produce your own customized metrics in your Camel routes. For more information, refer to the microprofile-metrics component documentation.

Metrics are exposed to Quarkus as application metrics and they can be browsed at http://localhost:8080/q/metrics/application.

2.29.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
🔒 **quarkus.camel.metrics.enable-route-policy** Set whether to enable the MicroProfileMetricsRoutePolicyFactory for capturing metrics on route processing times.	**boolean**	**true**
🔒 **quarkus.camel.metrics.enable-message-history** Set whether to enable the MicroProfileMetricsMessageHistoryFactory for capturing metrics on individual route node processing times. Depending on the number of configured route nodes, there is the potential to create a large volume of metrics. Therefore, this option is disabled by default.	**boolean**	**false**
🔒 **quarkus.camel.metrics.enable-exchange-event-notifier** Set whether to enable the MicroProfileMetricsExchangeEventNotifier for capturing metrics on exchange processing times.	**boolean**	**true**
🔒 **quarkus.camel.metrics.enable-route-event-notifier** Set whether to enable the MicroProfileMetricsRouteEventNotifier for capturing metrics on the total number of routes and total number of routes running.	**boolean**	**true**

Configuration property	Type	Default
🔒 **quarkus.camel.metrics.enable-camel-context-event-notifier** Set whether to enable the MicroProfileMetricsCamelContextEventNotifier for capturing metrics about the CamelContext, such as status and uptime.	**boolean**	**true**

🔒 Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.30. MLLP

Communicate with external systems using the MLLP protocol.

2.30.1. What's inside

- MLLP component, URI syntax: **mllp:hostname:port**

Please refer to the above link for usage and configuration details.

2.30.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mllp</artifactId>
</dependency>
```

2.30.3. Additional Camel Quarkus configuration

2.31. MOCK

Test routes and mediation rules using mocks.

2.31.1. What's inside

- Mock component, URI syntax: **mock:name**

Please refer to the above link for usage and configuration details.

2.31.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mock</artifactId>
</dependency>
```

2.31.3. Usage

To use camel-mock capabilities in tests it is required to get access to MockEndpoint instances.

CDI injection could be used for accessing instances (see Quarkus documentation). You can inject camelContext into test using **@Inject** annotation. Camel context can be then used for obtaining mock endpoints. See the following example:

```
import javax.inject.Inject;

import org.apache.camel.CamelContext;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.component.mock.MockEndpoint;
import org.junit.jupiter.api.Test;

import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class MockJvmTest {

    @Inject
    CamelContext camelContext;

    @Inject
    ProducerTemplate producerTemplate;

    @Test
    public void test() throws InterruptedException {

        producerTemplate.sendBody("direct:start", "Hello World");

        MockEndpoint mockEndpoint = camelContext.getEndpoint("mock:result", MockEndpoint.class);
        mockEndpoint.expectedBodiesReceived("Hello World");

        mockEndpoint.assertIsSatisfied();
    }
}
```

Route used for the example test:

```
import javax.enterprise.context.ApplicationScoped;

import org.apache.camel.builder.RouteBuilder;

@ApplicationScoped
public class MockRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:start").to("mock:result");
    }
}
```

2.31.4. Camel Quarkus limitations

Injection of CDI beans (described in Usage) does not work in native mode.

In the native mode the test and the application under test are running in two different processes and it is not possible to share a mock bean between them (see Quarkus documentation).

2.32. MONGODB

Perform operations on MongoDB documents and collections.

2.32.1. What's inside

- MongoDB component, URI syntax: **mongodb:connectionBean**

Please refer to the above link for usage and configuration details.

2.32.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-mongodb</artifactId>
</dependency>
```

2.32.3. Additional Camel Quarkus configuration

The extension leverages the Quarkus MongoDB Client extension. The Mongo client can be configured via the Quarkus MongoDB Client configuration options.

The Camel Quarkus MongoDB extension automatically registers a MongoDB client bean named **camelMongoClient**. This can be referenced in the mongodb endpoint URI **connectionBean** path parameter. For example:

```
from("direct:start")
.to("mongodb:camelMongoClient?database=myDb&collection=myCollection&operation=findAll")
```

If your application needs to work with multiple MongoDB servers, you can create a "named" client and reference in your route by injecting a client and the related configuration as explained in the Quarkus MongoDB extension client injection. For example:

```
//application.properties
quarkus.mongodb.mongoClient1.connection-string = mongodb://root:example@localhost:27017/
```

```
//Routes.java

  @ApplicationScoped
  public class Routes extends RouteBuilder {
    @Inject
    @MongoClientName("mongoClient1")
    MongoClient mongoClient1;

    @Override
    public void configure() throws Exception {
      from("direct:defaultServer")
        .to("mongodb:camelMongoClient?
database=myDb&collection=myCollection&operation=findAll")

      from("direct:otherServer")
        .to("mongodb:mongoClient1?
database=myOtherDb&collection=myOtherCollection&operation=findAll");
    }
  }
```

Note that when using named clients, the "default" **camelMongoClient** bean will still be produced. Refer to the Quarkus documentation on Multiple MongoDB Clients for more information.

2.33. NETTY

Socket level networking using TCP or UDP with the Netty 4.x.

2.33.1. What's inside

- Netty component, URI syntax: **netty:protocol://host:port**

Please refer to the above link for usage and configuration details.

2.33.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-netty</artifactId>
</dependency>
```

2.34. OPENAPI JAVA

Expose OpenAPI resources defined in Camel REST DSL

2.34.1. What's inside

- Openapi Java

Please refer to the above link for usage and configuration details.

2.34.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-openapi-java</artifactId>
</dependency>
```

2.34.3. Camel Quarkus limitations

The **apiContextIdListing** configuration option is not supported. Since multiple **CamelContexts** are not supported and Quarkus applications run standalone, there is no scenario where attempting to resolve OpenApi specifications for a specific **CamelContext** would be useful. It also introduces some additional overhead of requiring JMX (which is not supported in native mode) & additional Camel Quarkus extensions for processing XML.

2.35. PLATFORM HTTP

This extension allows for creating HTTP endpoints for consuming HTTP requests.

It is built on top of Eclipse Vert.x Web service provided by the **quarkus-vertx-web** extension.

2.35.1. What's inside

- Platform HTTP component, URI syntax: **platform-http:path**

Please refer to the above link for usage and configuration details.

2.35.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-platform-http</artifactId>
</dependency>
```

2.35.3. Usage

2.35.3.1. Basic Usage

Serve all HTTP methods on the /**hello** endpoint:

```
from("platform-http:/hello").setBody(simple("Hello ${header.name}"));
```

Serve only GET requests on the /**hello** endpoint:

```
from("platform-http:/hello?httpMethodRestrict=GET").setBody(simple("Hello ${header.name}"));
```

2.35.3.2. Using **platform-http** via Camel REST DSL

To be able to use Camel REST DSL with the **platform-http** component, add **camel-quarkus-rest** in addition to **camel-quarkus-platform-http** to your **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest</artifactId>
</dependency>
```

Then you can use the Camel REST DSL:

```
rest()
    .get("/my-get-endpoint")
        .route()
        .setBody(constant("Hello from /my-get-endpoint"))
        .endRest()
    .post("/my-post-endpoint")
        .route()
        .setBody(constant("Hello from /my-post-endpoint"))
        .endRest();
```

2.35.3.3. Handling **multipart/form-data** file uploads

You can restrict the uploads to certain file extensions by white listing them:

```
from("platform-http:/upload/multipart?fileNameExtWhitelist=html,txt&httpMethodRestrict=POST")
    .to("log:multipart")
    .process(e -> {
        final AttachmentMessage am = e.getMessage(AttachmentMessage.class);
        if (am.hasAttachments()) {
            am.getAttachments().forEach((fileName, dataHandler) -> {
                try (InputStream in = dataHandler.getInputStream()) {
                    // do something with the input stream
                } catch (IOException ioe) {
                    throw new RuntimeException(ioe);
                }
            });
        }
    });
```

Also check the **quarkus.http.body.*** configuration options in Quarkus documentation, esp. **quarkus.http.body.handle-file-uploads**, **quarkus.http.body.uploads-directory** and **quarkus.http.body.delete-uploaded-files-on-end**.

2.35.4. Additional Camel Quarkus configuration

2.35.4.1. Platform HTTP server configuration

Configuration of the platform HTTP server is managed by Quarkus. Refer to the Quarkus HTTP configuration guide for the full list of configuration options.

To configure SSL for the Platform HTTP server, follow the secure connections with SSL guide. Note that configuring the server for SSL with **SSLContextParameters** is not currently supported.

2.35.4.2. Character encodings

- Check the Character Encodings section of the *Developing Applications with Camel Extensions for Quarkus* guide if you expect your application to send or receive requests using non-default encodings.

2.36. REST

Expose REST services and their OpenAPI Specification or call external REST services.

2.36.1. What's inside

- REST component, URI syntax: **rest:method:path:uriTemplate**
- REST API component, URI syntax: **rest-api:path/contextIdPattern**

Please refer to the above links for usage and configuration details.

2.36.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-rest</artifactId>
</dependency>
```

2.36.3. Additional Camel Quarkus configuration

This extension depends on the Platform HTTP extension and configures it as the component that provides the REST transport.

2.36.3.1. Path parameters containing special characters with platform-http

When using the **platform-http** REST transport, some characters are not allowed within path parameter names. This includes the '-' and '$' characters.

In order to make the below example REST **/dashed/param** route work correctly, a system property is required **io.vertx.web.route.param.extended-pattern=true**.

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        rest("/api")
            // Dash '-' is not allowed by default
            .get("/dashed/param/{my-param}")
            .route()
                .setBody(constant("Hello World"))
            .endRest()

            // The non-dashed path parameter works by default
            .get("/undashed/param/{myParam}")
            .route()
                .setBody(constant("Hello World"))
            .endRest();
    }
}
```

There is some more background to this in the Vert.x Web documentation.

2.36.3.2. Configuring alternate REST transport providers

To use another REST transport provider, such as **netty-http** or **servlet**, you need to add the respective extension as a dependency to your project and set the provider in your **RouteBuilder**. E.g. for **servlet**, you'd have to add the **org.apache.camel.quarkus:camel-quarkus-servlet** dependency and the set the provider as follows:

```
import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

    @Override
    public void configure() {
        restConfiguration()
                .component("servlet");
        ...
    }
}
```

2.37. SALESFORCE

Communicate with Salesforce using Java DTOs.

2.37.1. What's inside

- Salesforce component, URI syntax: **salesforce:operationName:topicName**

Please refer to the above link for usage and configuration details.

2.37.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-salesforce</artifactId>
</dependency>
```

2.37.3. Usage

2.37.3.1. Generating Salesforce DTOs with the salesforce-maven-plugin

NOTE

The **camel-salesforce-maven-plugin** is only covered by community support.

To generate Salesforce DTOs for your project, use the **salesforce-maven-plugin**. The example code snippet below creates a single DTO for the **Account** object.

```
<plugin>
    <groupId>org.apache.camel.maven</groupId>
    <artifactId>camel-salesforce-maven-plugin</artifactId>
    <version>3.11.1</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <clientId>${env.SALESFORCE_CLIENTID}</clientId>
                <clientSecret>${env.SALESFORCE_CLIENTSECRET}</clientSecret>
                <userName>${env.SALESFORCE_USERNAME}</userName>
                <password>${env.SALESFORCE_PASSWORD}</password>
                <loginUrl>https://login.salesforce.com</loginUrl>

<packageName>org.apache.camel.quarkus.component.salesforce.generated</packageName>
                <outputDirectory>src/main/java</outputDirectory>
                <includes>
                    <include>Account</include>
                </includes>
            </configuration>
        </execution>
    </executions>
</plugin>
```

2.37.3.2. Register additional Salesforce classes for reflection in native mode

For native mode, it is necessary to register some additional classes for reflection.

1. Classes in package **org.apache.camel.component.salesforce.api.dto**
2. DTO classes generated by the **camel-salesforce-maven-plugin**

To do this, add the following configuration property to **application.properties**. Replace **org.my.custom.dto.package** with your custom DTO package (if applicable, othwerwise it can be be removed).

```
quarkus.camel.native.reflection.include-
patterns=org.apache.camel.component.salesforce.api.dto.*,org.my.custom.dto.package.*
```

2.37.4. SSL in native mode

This extension auto-enables SSL support in native mode. Hence you do not need to add **quarkus.ssl.native=true** to your **application.properties** yourself. See also Quarkus SSL guide.

2.38. XQUERY

Query and/or transform XML payloads using XQuery and Saxon.

2.38.1. What's inside

- XQuery component, URI syntax: **xquery:resourceUri**
- XQuery language

Please refer to the above links for usage and configuration details.

2.38.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-saxon</artifactId>
</dependency>
```

2.38.3. Additional Camel Quarkus configuration

This component is able to load XQuery definitions from classpath. To make it work also in native mode, you need to explicitly embed the queries in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the two routes below load an XQuery script from two classpath resources named **myxquery.txt** and **another-xquery.txt** respectively:

```
from("direct:start").transform().xquery("resource:classpath:myxquery.txt", String.class);
from("direct:start").to("xquery:another-xquery.txt");
```

To include these (an possibly other queries stored in **.txt** files) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = *.txt
```

2.39. SEDA

Asynchronously call another endpoint from any Camel Context in the same JVM.

2.39.1. What's inside

- SEDA component, URI syntax: **seda:name**

Please refer to the above link for usage and configuration details.

2.39.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-seda</artifactId>
</dependency>
```

2.40. SOAP DATAFORMAT

Marshal Java objects to SOAP messages and back.

2.40.1. What's inside

- SOAP data format

Please refer to the above link for usage and configuration details.

2.40.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-soap</artifactId>
</dependency>
```

2.41. SQL

Perform SQL queries.

2.41.1. What's inside

- SQL component, URI syntax: **sql:query**
- SQL Stored Procedure component, URI syntax: **sql-stored:template**

Please refer to the above links for usage and configuration details.

2.41.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-sql</artifactId>
</dependency>
```

2.41.3. Additional Camel Quarkus configuration

2.41.3.1. Configuring a DataSource

This extension leverages Quarkus Agroal for **DataSource** support. Setting up a **DataSource** can be achieved via configuration properties.

```
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=your-username
quarkus.datasource.password=your-password
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/your-database
quarkus.datasource.jdbc.max-size=16
```

The Camel SQL component will automatically resolve the **DataSource** bean from the registry. When configuring multiple datasources, you can specify which one is to be used on an SQL endpoint via the URI options **datasource** or **dataSourceRef**. Refer to the SQL component documentation for more details.

2.41.3.1.1. Zero configuration with Quarkus Dev Services

In dev and test mode you can take advantage of Configuration Free Databases. The Camel SQL component will be automatically configured to use a **DataSource** that points to a local containerized instance of the database matching the JDBC driver type that you have selected.

2.41.3.2. SQL scripts

When configuring **sql** or **sql-stored** endpoints to reference script files from the classpath, set the following configuration property to ensure that they are available in native mode.

```
quarkus.native.resources.includes = queries.sql, sql/*.sql
```

2.41.3.3. SQL Aggregator

If your exchanges in native mode contain objects, which are not automatically registered for serialization (see documentation), you have to register them manually (see documentation)

2.42. TIMER

Generate messages in specified intervals using java.util.Timer.

2.42.1. What’s inside

- Timer component, URI syntax: **timer:timerName**

Please refer to the above link for usage and configuration details.

2.42.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-timer</artifactId>
</dependency>
```

2.43. XPATH

Evaluates an XPath expression against an XML payload.

2.43.1. What's inside

- XPath language

Please refer to the above link for usage and configuration details.

2.43.2. Maven coordinates

Create a new project with this extension on code.quarkus.redhat.com

Or add the coordinates to your existing project:

```
<dependency>
    <groupId>org.apache.camel.quarkus</groupId>
    <artifactId>camel-quarkus-xpath</artifactId>
</dependency>
```

2.43.3. Additional Camel Quarkus configuration

This component is able to load xpath expressions from classpath resources. To make it work also in native mode, you need to explicitly embed the expression files in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load an XPath expression from a classpath resource named **myxpath.txt**:

```
from("direct:start").transform().xpath("resource:classpath:myxpath.txt");
```

To include this (an possibly other expressions stored in **.txt** files) in the native image, you would have to add something like the following to your **application.properties** file:

```
quarkus.native.resources.includes = *.txt
```