Red Hat build of Apache Camel 4.4

Red Hat build of Apache Camel for Quarkus Reference

Red Hat build of Apache Camel for Quarkus provided by Red Hat

Last Updated: 2024-06-28

Red Hat build of Apache Camel 4.4 Red Hat build of Apache Camel for Quarkus Reference

Red Hat build of Apache Camel for Quarkus provided by Red Hat

Legal Notice

Abstract

Red Hat build of Apache Camel 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

PROVIDING FEEDBACK ON RED HAT BUILD OF APACHE CAMEL DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to create ticket
2. Enter a brief description of the issue in the Summary.
3. Provide a detailed description of the issue or enhancement in the Description. Include a URL to where the issue occurs in the documentation.
4. Clicking Submit creates and routes the issue to the appropriate documentation team.

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. SUPPORT LEVEL DEFINITIONS

New features, services, and components go through a number of support levels before inclusion in Red Hat build of Apache Camel 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 Red Hat build of Apache Camel for Quarkus technologies while providing feedback to help guide future development activities.

Table 1.1. Red Hat build of Apache Camel for Quarkus support levels

Type	Description
Community Support	As part of Red Hat's commitment to upstream first, integration of new extensions into our Red Hat build of Apache Camel 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.

CHAPTER 2. CAMEL QUARKUS EXTENSIONS OVERVIEW

2.1. SUPPORTED EXTENSIONS

There are 97 extensions.

Table 2.1. Red Hat build of Apache Camel for Quarkus Support Matrix

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
1	AMQP	camel-quarkus-amqp	Messaging with AMQP protocol using Apache QPid Client.	Production Support	Production Support	Yes
2	Attachments	camel-quarkus-attachments	Support for attachments on Camel messages	Production Support	Production Support	Yes
3	AWS2 CloudWatch	camel-quarkus-aws2-cw	Sending metrics to AWS CloudWatch using AWS SDK version 2.x.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
4	AWS2 DynamoDB	camel-quarkus-aws2-ddb	Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.	Production Support	Production Support	Yes
5	AWS2 Kinesis	camel-quarkus-aws2-kinesis	Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.	Production Support	Production Support	Yes
6	AWS2 Lambda	camel-quarkus-aws2-lambda	Manage and invoke AWS Lambda functions using AWS SDK version 2.x.	Production Support	Production Support	Yes
7	AWS2 S3 Storage	camel-quarkus-aws2-s3	Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Support on IBM Power and IBM Z
8	AWS2 Simple Notification System (SNS)	camel-quarkus-aws2-sns	Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.	Production Support	Production Support	Yes
9	AWS2 Simple Queue Service (SQS)	camel-quarkus-aws2-sqs	Send and receive messages to/from AWS SQS service using AWS SDK version 2.x.	Production Support	Production Support	Yes
10	Azure ServiceBus	camel-quarkus-azure-servicebus	Send and receive messages to/from Azure Service Bus.	Technology Preview	Technology Preview	Yes
11	Azure Storage Blob	camel-quarkus-azure-storage-blob	Store and retrieve blobs from Azure Storage Blob Service using SDK v12.	Technology Preview	Technology Preview	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
12	Azure Storage Queue	camel-quarkus-azure-storage-queue	The **azure-storage-queue** component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.	Technology Preview	Technology Preview	Yes
13	Bean	camel-quarkus-bean	Invoke methods of Java beans	Production Support	Production Support	Yes
14	Bean-validator	camel-quarkus-bean-validator	Validate the message body using the Java Bean Validation API.	Production Support	Production Support	Yes
15	Browse	camel-quarkus-browse	Inspect the messages received on endpoints supporting BrowsableEndpoint.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Support on IBM Power and IBM Z
16	Cassandra CQL	camel-quarkus-cassandraql	Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.	Production Support	Production Support	Yes
17	Cli-connector	camel-quarkus-cli-connector	Runtime adapter connecting with Camel CLI	Production Support	Production Support	Yes
18	Controlbus	camel-quarkus-controlbus	Manage and monitor Camel routes.	Production Support	Production Support	Yes
19	Core	camel-quarkus-core	Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Simple and Tokenize	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
20	Crypto	camel-quarkus-crypto	Sign and verify exchanges using the Signature Service of the Java Cryptographic Extension (JCE).	Production Support	Production Support	Yes
21	Cron	camel-quarkus-cron	A generic interface for triggering events at times specified through the Unix cron syntax.	Production Support	Production Support	Yes
22	CXF SOAP	camel-quarkus-cxf-soap	Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.	Production Support	Production Support	Yes
23	Dataformat	camel-quarkus-dataformat	Use a Camel Data Format as a regular Camel Component.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
24	Dataset	camel-quarkus-dataset	Provide data for load and soak testing of your Camel application.	Technology Previee	Technology Previee	Yes
25	Direct	camel-quarkus-direct	Call another endpoint from the same Camel Context synchronously.	Production Support	Production Support	Yes
26	FHIR	camel-quarkus-fhir	Exchange information in the healthcare domain using the FHIR (Fast Healthcare Interoperability Resources) standard. Marshall and unmarshall FHIR objects to/from JSON. Marshall and unmarshall FHIR objects to/from XML.	Production Support	Production Support	No
27	Flink	camel-quarkus-flink	Send DataSet jobs to an Apache Flink cluster.	Technology Preview	None	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
28	File	camel-quarkus-file	Read and write files.	Production Support	Production Support	Yes
29	FTP	camel-quarkus-ftp	Upload and download files to/from SFTP, FTP or SFTP servers	Production Support	Production Support	Yes
30	Google BigQuery	camel-quarkus-google-bigquery	Access Google Cloud BigQuery service using SQL queries or Google Client Services API	Production Support	Production Support	Yes
31	Google Pubsub	camel-quarkus-google-pubsub	Send and receive messages to/from Google Cloud Platform PubSub Service.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
32	gRPC	camel-quarkus-grpc	Expose gRPC endpoints and access external gRPC endpoints.	Production Support	Production Support	Yes
33	HTTP	camel-quarkus-http	Send requests to external HTTP servers using Apache HTTP Client 5.x.	Production Support	Production Support	Yes
34	Infinispan	camel-quarkus-infinispan	Read and write from/to Infinispan distributed key/value store and data grid.	Production Support	Production Support	No
35	Jasypt	camel-quarkus-jasypt	Security using Jasypt	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
36	Java JOOR DSL	camel-quarkus-java-joor-dsl	Support for parsing Java route definitions at runtime	Community Support	Community Support	Yes
37	JDBC	camel-quarkus-jdbc	Access databases through SQL and JDBC.	Production Support	Production Support	Yes
38	JIRA	camel-quarkus-jira	Interact with JIRA issue tracker.	Production Support	Production Support	Yes
39	JMS	camel-quarkus-jms	Sent and receive messages to/from a JMS Queue or Topic.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
40	JPA	camel-quarkus-jpa	Store and retrieve Java objects from databases using Java Persistence API (JPA).	Production Support	Production Support	Yes
41	JT400	camel-quarkus-jt400	Exchanges messages with an IBM i system using data queues, message queues, or program call. IBM i is the replacement for AS/400 and iSeries servers.	Production Support	Production Support	Yes
42	JTA	camel-quarkus-jta	Enclose Camel routes in transactions using Java Transaction API (JTA) and Narayana transaction manager	Production Support	Production Support	Yes
43	Kafka	camel-quarkus-kafka	Sent and receive messages to/from an Apache Kafka broker.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
44	Kamelet	camel-quarkus-kamelet	Materialize route templates	Production Support	Production Support	Yes
45	Kubernetes	camel-quarkus-kubernetes	Perform operations against Kubernetes API	Technology Preview	Technology Preview	Yes
46	Kudu	camel-kudu	Interact with Apache Kudu, a free and open source column-oriented data store of the Apache Hadoop ecosystem.	Production Support	Production Support	No
47	Language	camel-quarkus-language	Execute scripts in any of the languages supported by Camel.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
48	LDAP	camel-quarkus-ldap	Perform searches on LDAP servers.	Production Support	Production Support	Yes
49	Log	camel-quarkus-log	Log messages to the underlying logging mechanism.	Production Support	Production Support	Yes
50	LRA	camel-quarkus-lra	Camel saga binding for Long-Running-Action framework.	Production Support	Production Support	Yes
51	Mail	camel-quarkus-mail	Send and receive emails using imap, pop3 and smtp protocols. Marshal Camel messages with attachments into MIME-Multipart messages and back.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
52	Management	camel-quarkus-management	JMX management strategy and associated managed resources.	Production Support	Production Support	Yes
53	MapStruct	camel-quarkus-mapstruct	Type Conversion using Mapstruct	Production Support	Production Support	Yes
54	Master	camel-quarkus-master	Have only a single consumer in a cluster consuming from a given endpoint; with automatic failover if the JVM dies.	Production Support	Production Support	Yes
55	Micrometer	camel-quarkus-micrometer	Collect various metrics directly from Camel routes using the Micrometer library.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
56	MicroProfile Fault Tolerance	camel-quarkus-microprofile-fault-tolerance	Circuit Breaker EIP using Microprofile Fault Tolerance	Production Support	Production Support	Yes
57	MicroProfile Health	camel-quarkus-microprofile-health	Expose Camel health checks via MicroProfile Health	Production Support	Production Support	Yes
58	Minio	camel-quarkus-minio	Store and retrieve objects from Minio Storage Service using Minio SDK.	Production Support	Production Support	Yes
59	MLLP	camel-quarkus-mllp	Communicate with external systems using the MLLP protocol.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Support on IBM Power and IBM Z
60	MyBatis	camel-quarkus-mybatis	Performs a query, poll, insert, update or delete in a relational database using MyBatis.	Production Support	Production Support	Yes
61	Mock	camel-quarkus-mock	Test routes and mediation rules using mocks.	Production Support	Production Support	Yes
62	MongoDB	camel-quarkus-mongodb	Perform operations on MongoDB documents and collections.	Technology Preview	Technology Preview	Yes
63	Netty	camel-quarkus-netty	Socket level networking using TCP or UDP with Netty 4.x.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
64	Netty HTTP	camel-quarkus-netty-http	Netty HTTP server and client using the Netty 4.x.	Production Support	Production Support	Yes
65	Openapi Java	camel-quarkus-openapi-java	Expose OpenAPI resources defined in Camel REST DSL	Production Support	Production Support	Yes
66	OpenTelemetry	camel-quarkus-opentelemetry	Distributed tracing using OpenTelemetry	Production Support	Production Support	Yes
67	Quartz	camel-quarkus-quartz	Schedule sending of messages using the Quartz 2.x scheduler.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
68	Paho	camel-quarkus-paho	Communicate with MQTT message brokers using Eclipse Paho MQTT Client.	Production Support	Production Support	Yes
69	Paho MQTT5	camel-quarkus-paho-mqtt5	Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.	Production Support	Production Support	Yes
70	Platform HTTP	camel-quarkus-platform-http	Expose HTTP endpoints using the HTTP server available in the current platform.	Production Support	Production Support	Yes
71	Ref	camel-quarkus-ref	Route messages to an endpoint looked up dynamically by name in the Camel Registry.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
72	REST	camel-quarkus-rest	Expose REST services and their OpenAPI Specification or call external REST services.	Production Support	Production Support	Yes
73	REST OpenAPI	camel-quarkus-rest-openapi	Configure REST producers based on an OpenAPI specification document delegating to a component implementing the RestProducerFactory interface.	Production Support	Production Support	Yes
74	Salesforce	camel-quarkus-salesforce	Communicate with Salesforce using Java DTOs.	Production Support	Production Support	Yes
75	SAGA	camel-quarkus-saga	Execute custom actions within a route using the Saga EIP.	Technology Preview	Technology Preview	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
76	SAP	camel-quarkus-sap	Provides SAP Camel Component.	Production Support	None	Yes
77	Saxon	camel-quarkus-saxon	Query and/or transform XML payloads using XQuery and Saxon.	Production Support	Production Support	Yes
78	Scheduler	camel-quarkus-scheduler	Generate messages in specified intervals using java.util.concurrent.ScheduledExecutorService.	Production Support	Production Support	Yes
79	Seda	camel-quarkus-seda	Asynchronously call another endpoint from any Camel Context in the same JVM.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
80	Servlet	camel-quarkus-servlet	Serve HTTP requests by a Servlet.	Production Support	Production Support	Yes
81	Slack	camel-quarkus-slack	Send and receive messages to/from Slack.	Production Support	Production Support	Yes
82	SNMP	camel-quarkus-snmp	Receive traps and poll SNMP (Simple Network Management Protocol) capable devices.	Production Support	Technology Preview	Yes
83	Splunk	camel-quarkus-splunk	Publish or search for events in Splunk.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Support on IBM Power and IBM Z
84	Splunk HEC	camel-quarkus-splunk-hec	The splunk component allows to publish events in Splunk using the HTTP Event Collector.	Production Support	Production Support	Yes
85	SQL	camel-quarkus-sql	Perform SQL queries.	Production Support	Production Support	Yes
86	Telegram	camel-quarkus-telegram	Send and receive messages acting as a Telegram Bot Telegram Bot API.	Production Support	Production Support	Yes
87	Timer	camel-quarkus-timer	Generate messages in specified intervals using java.util.Timer.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
88	Validator	camel-quarkus-validator	Validate the payload using XML Schema and JAXP Validation.	Production Support	Production Support	Yes
89	Velocity	camel-quarkus-velocity	Transform messages using a Velocity template.	Production Support	Production Support	Yes
90	VertX HTTP	camel-quarkus-vertx-http	Camel HTTP client support with Vert.x	Production Support	Production Support	Yes
91	VertX Websocket	camel-quarkus-vertx-websocket	Camel WebSocket support with Vert.x	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
92	XJ	camel-quarkus-xj	Transform JSON and XML message using a XSLT	Production Support	Production Support	Yes
93	XML IO DSL	camel-quarkus-xml-io-dsl	An XML stack for parsing XML route definitions	Production Support	Production Support	Yes
94	XSLT	camel-quarkus-xslt	Transforms XML payload using an XSLT template.	Production Support	Production Support	Yes
95	XSLT Saxon	camel-quarkus-xslt-saxon	Transform XML payloads using an XSLT template using Saxon.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
96	Zip File	camel-quarkus-zipfile	Compression and decompress streams using java.util.zip.ZipStream.	Production Support	Production Support	Yes
97	Zip Deflate Compression	camel-quarkus-zip-deflater	Compress and decompress streams using java.util.zip.Deflater, java.util.zip.Inflater or java.util.zip.GZIPStream.	Production Support	Production Support	Yes

2.2. SUPPORTED LANGUAGES

There are 7 languages.

Table 2.2. Red Hat build of Apache Camel for Quarkus Support Matrix Languages

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
1	Bean	camel-quarkus-bean	Invoke methods of Java beans	Production Support	Production Support	Yes
2	Core	camel-quarkus-core	Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Simple and Tokenize	Production Support	Production Support	Yes
3	Hl7	camel-quarkus-hl7	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.	Production Support	Production Support	Yes
4	JSONPath	camel-quarkus-jsonpath	Evaluate a JSONPath expression against a JSON message body	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
5	Jslt	camel-quarkus-jslt	Query or transform JSON payloads using an JSLT.	Production Support	Production Support	Yes
6	Saxon	camel-quarkus-saxon	Query and/or transform XML payloads using XQuery and Saxon.	Production Support	Production Support	Yes
7	Xpath	camel-quarkus-xpath	Evaluates an XPath expression against an XML payload	Production Support	Production Support	Yes

2.3. SUPPORTED DATA FORMATS

There are 13 data formats.

Table 2.3. Red Hat build of Apache Camel for Quarkus Support Matrix Data formats

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
1	Avro	camel-quarkus-avro	Serialize and deserialize messages using Apache Avro binary data format.	Production Support	Production Support	Yes
2	Bindy	camel-quarkus-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	Production Support	Production Support	Yes
3	Crypto	camel-quarkus-crypto	Java Cryptographic Extension: Symmetric (shared-key) encryption and decryption using Camel's marshal and unmarshal formatting mechanism.	Production Support	Production Support	Yes
4	Gson	camel-quarkus-gson	Marshal POJOs to JSON and back using Gson	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
5	HI7	camel-quarkus-hl7	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.	Production Support	Production Support	Yes
6	Jackson	camel-quarkus-jackson	Marshal POJOs to JSON and back using Jackson	Production Support	Production Support	Yes
7	Jackson Avro	camel-quarkus-jackson-avro	Marshal POJOs to Avro and back using Jackson.	Production Support	Production Support	Yes
8	Jackson ProtoBuf	camel-quarkus-jackson-protobuf	Marshal POJOs to Protobuf and back using Jackson.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
9	Jackson XML	camel-quarkus-jacksonxml	Unmarshal an XML payloads to POJOs and back using XMLMapper extension of Jackson.	Production Support	Production Support	Yes
10	Jaxb	camel-quarkus-jaxb	Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.	Production Support	Production Support	Yes
11	Xml-JAXP	camel-quarkus-xml-jaxp	Camel XML JAXP	Production Support	Production Support	Yes
12	PGP	camel-quarkus-crypto	Symmetric (shared-key) encryption and decryption using Camel's marshal and unmarshal formatting mechanism.	Production Support	Production Support	Yes

	Extension	Artifact	Description	JVM Support Level	Native Support Level	Supported on IBM Power and IBM Z
13	SOAP	camel-quarkus-soap	Marshal Java objects to SOAP messages and back.	Production Support	Production Support	Yes

CHAPTER 3. CAMEL QUARKUS EXTENSIONS REFERENCE

This chapter provides usage information for Red Hat build of Apache Camel for Quarkus.

3.1. AMQP

Messaging with AMQP protocol using Apache QPid Client.

3.1.1. What's inside

- AMQP component, URI syntax: **amqp:destinationType:destinationName**

Refer to the above link for usage and configuration details.

3.1.2. 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-amqp</artifactId>
</dependency>
```

3.1.3. Usage

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

The Camel AMQP component supports message mapping between **jakarta.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-xml-jaxp** extension is present on the classpath.

3.1.3.2. Native mode support for jakarta.jms.ObjectMessage

When sending JMS message payloads as **jakarta.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.

3.1.3.3. Connection Pooling

You can use the **quarkus-pooled-jms** extension to get pooling support for the connections. Refer to the quarkus-pooled-jms extension documentation for more information.

Just add the following dependency to your **pom.xml**:

```
<dependency>
  <groupId>io.quarkiverse.messaginghub</groupId>
  <artifactId>quarkus-pooled-jms</artifactId>
</dependency>
```

To enable the pooling support, you need to add the following configuration to your **application.properties**:

```
quarkus.qpid-jms.wrap=true
```

3.1.4. 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)
```

3.1.5. Additional Camel Quarkus configuration

The extension leverages the Quarkus Qpid JMS extension. A ConnectionFactory bean is automatically created and wired into the AMQP component for you. The connection factory can be configured via the Quarkus Qpid JMS configuration options.

3.2. ATTACHMENTS

Support for attachments on Camel messages

3.2.1. What's inside

- Attachments

Refer to the above link for usage and configuration details.

3.2.2. 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-attachments</artifactId>
</dependency>
```

3.3. AVRO

Serialize and deserialize messages using Apache Avro binary data format.

3.3.1. What's inside

- Avro data format

Refer to the above link for usage and configuration details.

3.3.2. 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-avro</artifactId>
</dependency>
```

3.3.3. 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.

The 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.

3.4. AWS 2 CLOUDWATCH

Sending metrics to AWS CloudWatch.

3.4.1. What's inside

- AWS CloudWatch component, URI syntax: **aws2-cw:namespace**

Refer to the above link for usage and configuration details.

3.4.2. 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-cw</artifactId>
</dependency>
```

3.4.3. 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.

3.5. AWS 2 DYNAMODB

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

3.5.1. What's inside

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

Refer to the above links for usage and configuration details.

3.5.2. 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-ddb</artifactId>
</dependency>
```

3.5.3. 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.

3.5.4. Additional Camel Quarkus configuration

3.5.4.1. 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 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 **DynamoDbClient**:

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

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

3.6. AWS 2 KINESIS

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

3.6.1. What's inside

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

Refer to the above links for usage and configuration details.

3.6.2. 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-kinesis</artifactId>
</dependency>
```

3.6.3. 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.

3.7. AWS 2 LAMBDA

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

3.7.1. What's inside

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

Refer to the above link for usage and configuration details.

3.7.2. 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-lambda</artifactId>
</dependency>
```

3.7.3. 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.

3.7.4. Additional Camel Quarkus configuration

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

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.

3.8. AWS 2 S3 STORAGE SERVICE

Store and retrieve objects from AWS S3 Storage Service.

3.8.1. What's inside

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

Refer to the above link for usage and configuration details.

3.8.2. 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-aws2-s3</artifactId>
</dependency>
```

3.8.3. 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.

3.8.4. Additional Camel Quarkus configuration

3.8.4.1. 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 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 **S3Client**:

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

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

3.9. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)

Send messages to an AWS Simple Notification Topic.

3.9.1. What's inside

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

Refer to the above link for usage and configuration details.

3.9.2. 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-aws2-sns</artifactId>
</dependency>
```

3.9.3. 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.

3.9.4. Additional Camel Quarkus configuration

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

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

3.10. AWS 2 SIMPLE QUEUE SERVICE (SQS)

Send and receive messages to/from AWS SQS service.

3.10.1. What's inside

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

Refer to the above link for usage and configuration details.

3.10.2. 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-aws2-sqs</artifactId>
</dependency>
```

3.10.3. 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.

3.10.4. Additional Camel Quarkus configuration

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

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

3.11. AZURE SERVICEBUS

Send and receive messages to/from Azure Service Bus.

3.11.1. What's inside

- Azure ServiceBus component, URI syntax: **azure-servicebus:topicOrQueueName**

Refer to the above link for usage and configuration details.

3.11.2. Maven coordinates

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

3.12. AZURE STORAGE BLOB SERVICE

Store and retrieve blobs from Azure Storage Blob Service using SDK v12.

3.12.1. What's inside

- Azure Storage Blob Service component, URI syntax: **azure-storage-blob:accountName/containerName**

Refer to the above link for usage and configuration details.

3.12.2. 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-azure-storage-blob</artifactId>
</dependency>
```

3.12.3. Usage

3.12.3.1. Micrometer metrics support

If you wish to enable the collection of Micrometer metrics for the Reactor Netty transports, then you should declare a dependency on **quarkus-micrometer** to ensure that they are available via the Quarkus metrics HTTP endpoint.

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

3.12.4. 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.

3.13. AZURE STORAGE QUEUE SERVICE

The azure-storage-queue component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.

3.13.1. What's inside

- Azure Storage Queue Service component, URI syntax: **azure-storage-queue:accountName/queueName**

Refer to the above link for usage and configuration details.

3.13.2. 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-azure-storage-queue</artifactId>
</dependency>
```

3.13.3. Usage

3.13.3.1. Micrometer metrics support

If you wish to enable the collection of Micrometer metrics for the Reactor Netty transports, then you should declare a dependency on **quarkus-micrometer** to ensure that they are available via the Quarkus metrics HTTP endpoint.

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer</artifactId>
</dependency>
```

3.13.4. 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.

3.14. BEAN VALIDATOR

Validate the message body using the Java Bean Validation API.

3.14.1. What's inside

- Bean Validator component, URI syntax: **bean-validator:label**

Refer to the above link for usage and configuration details.

3.14.2. 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-bean-validator</artifactId>
</dependency>
```

3.14.3. Usage

3.14.3.1. Configuring the ValidatorFactory

Implementation of this extension leverages the Quarkus Hibernate Validator extension.

Therefore it is not possible to configure the **ValidatorFactory** by Camel's properties (**constraintValidatorFactory**, **messageInterpolator**, **traversableResolver**, **validationProviderResolver** and **validatorFactory**).

You can configure the **ValidatorFactory** by the creation of beans which will be injected into the default **ValidatorFactory** (created by Quarkus). See the Quarkus CDI documentation for more information.

3.14.3.2. Custom validation groups in native mode

When using custom validation groups in native mode, all the interfaces need to be registered for reflection (see the documentation).

Example:

```
@RegisterForReflection
public interface OptionalChecks {
}
```

3.14.4. Camel Quarkus limitations

It is not possible to describe your constraints as XML (by providing the file META-INF/validation.xml), only Java annotations are supported. This is caused by the limitation of the Quarkus Hibernate Validator extension (see the issue).

3.15. BEAN

Invoke methods of Java beans

3.15.1. What's inside

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

Refer to the above links for usage and configuration details.

3.15.2. 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-bean</artifactId>
</dependency>
```

3.15.3. Usage

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

3.16. 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

3.16.1. What's inside

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

Refer to the above links for usage and configuration details.

3.16.2. 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-bindy</artifactId>
</dependency>
```

3.16.3. 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.

3.17. BROWSE

Inspect the messages received on endpoints supporting BrowsableEndpoint.

3.17.1. What's inside

- Browse component, URI syntax: **browse:name**

Refer to the above link for usage and configuration details.

3.17.2. 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-browse</artifactId>
</dependency>
```

3.18. CASSANDRA CQL

Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.

3.18.1. What's inside

- Cassandra CQL component, URI syntax: **cql:beanRef:hosts:port/keyspace**

Refer to the above link for usage and configuration details.

3.18.2. 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-cassandraql</artifactId>
</dependency>
```

3.18.3. Additional Camel Quarkus configuration

3.18.3.1. Cassandra aggregation repository in native mode

In order to use Cassandra aggregation repositories like **CassandraAggregationRepository** in native mode, you must enable native serialization support.

In addition, if your exchange bodies are custom types, then they must be registered for serialization by annotating their class declaration with **@RegisterForReflection(serialization = true)**.

3.19. CLI CONNECTOR

Runtime adapter connecting with Camel CLI

3.19.1. What's inside

- CLI Connector

Refer to the above link for usage and configuration details.

3.19.2. Maven coordinates

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

```
    <artifactId>camel-quarkus-cli-connector</artifactId>
</dependency>
```

3.20. CONTROL BUS

Manage and monitor Camel routes.

3.20.1. What's inside

- Control Bus component, URI syntax: **controlbus:command:language**

Refer to the above link for usage and configuration details.

3.20.2. 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-controlbus</artifactId>
</dependency>
```

3.20.3. Usage

3.20.3.1. Statistics

When using the **stats** command endpoint, the **camel-quarkus-management** extension must be added as a project dependency to enable JMX. Maven users will have to add the following to their **pom.xml**:

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

3.20.3.2. Languages

The following languages are supported for use in the Control Bus extension in Red Hat build of Apache Camel for Quarkus:

- Bean language
- ExchangeProperty language
- Header language
- Simple language

3.20.3.2.1. Bean

The Bean language can be used to invoke a method on a bean to control the state of routes. The **org.apache.camel.quarkus:camel-quarkus-bean** extension must be added to the classpath. Maven users must add the following dependency to the POM:

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

In native mode, the bean class must be annotated with **@RegisterForReflection**.

3.20.3.2.2. Simple

The Simple language can be used to control the state of routes. The following example uses a **ProducerTemplate** to stop a route with the id **foo**:

```
template.sendBody(
    "controlbus:language:simple",
    "${camelContext.getRouteController().stopRoute('foo')}"
);
```

To use the OGNL notation, the **org.apache.camel.quarkus:camel-quarkus-bean** extension must be added as a dependency.

In native mode, the classes used in the OGNL notation must be registered for reflection. In the above code snippet, the **org.apache.camel.spi.RouteController** class returned from **camelContext.getRouteController()** must be registered. As this is a third-party class, it cannot be annotated with **@RegisterForReflection** directly - instead you can annotate a different class and specifying the target classes to register. For example, the class defining the Camel routes could be annotated with **@RegisterForReflection(targets = { org.apache.camel.spi.RouteController.class })**.

Alternatively, add the following line to your **src/main/resources/application.properties**:

```
quarkus.camel.native.reflection.include-patterns = org.apache.camel.spi.RouteController
```

3.20.4. Camel Quarkus limitations

3.20.4.1. Statistics

The **stats** action is not available in native mode as JMX is not supported on GraalVM. Therefore, attempting to build a native image with the **camel-quarkus-management** extension on the classpath will result in a build failure.

This feature is not supported in Red Hat build of Apache Camel for Quarkus.

3.21. CORE

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

3.21.1. What's inside

- Constant language

- ExchangeProperty language
- File language
- Header language
- Ref language
- Simple language
- Tokenize language

Refer to the above links for usage and configuration details.

3.21.2. 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-core</artifactId>
</dependency>
```

3.21.3. Additional Camel Quarkus configuration

3.21.3.1. Simple language

3.21.3.1.1. 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.

3.21.3.1.2. Using dynamic type resolution in native mode

When dynamically resolving a type from simple expressions like:

- **simple("${mandatoryBodyAs(TYPE)}")**
- **simple("${type:package.Enum.CONSTANT}")**
- **from("...").split(bodyAs(TYPE.class))**

- **simple("${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.

3.21.3.1.3. 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
```

3.21.3.1.4. 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

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**	
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**	

Configuration property	Type	Default
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	
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**

Configuration property	Type	Default
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
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	

Configuration property	Type	Default
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 **$**. This option narrows down the set selected by **include-patterns**. By default, no classes are excluded. This option cannot be used to unregister classes which have been registered internally by Quarkus extensions.	string	
quarkus.camel.native.reflection.include-patterns A comma separated list of Ant-path style patterns to match class names that should be registered for reflection. Use the class name format as returned by the **java.lang.Class.getName()** method: package segments delimited by period **.** and inner classes by dollar sign **$**. 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	

Configuration property	Type	Default
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**

Configuration property	Type	Default
quarkus.camel.expression.on-build-time-analysis-failure What to do if it is not possible to extract expressions from a route definition at build time.	org.apache.camel.quarkus.core.CamelConfig.FailureRem	warn

Configuration property	Type	Default
quarkus.camel.expression.extraction-enabled Indicates whether the expression extraction from the route definitions at build time must be done. If disabled, the expressions are compiled at runtime.	boolean	true
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.source-location-enabled Build time configuration options for enable/disable camel source location	boolean	false
quarkus.camel.main.shutdown.timeout A timeout (with millisecond precision) to wait for **CamelMain#stop()** to finish	java.time.Duration	PT3S

Configuration property	Type	Default
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.FailureRem	warn

Configuration property	Type	Default

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

3.22. CRON

A generic interface for triggering events at times specified through the Unix cron syntax.

3.22.1. What's inside

- Cron component, URI syntax: **cron:name**

Refer to the above link for usage and configuration details.

3.22.2. 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-cron</artifactId>
</dependency>
```

3.22.3. Additional Camel Quarkus configuration

The cron component is a generic interface component, as such Camel Quarkus users will need to use the cron extension together with another extension offering an implementation.

3.23. CRYPTO (JCE)

Sign and verify exchanges using the Signature Service of the Java Cryptographic Extension (JCE).

3.23.1. What's inside

- Crypto (Java Cryptographic Extension) data format
- Crypto (JCE) component, URI syntax: **crypto:cryptoOperation:name**
- PGP data format

Refer to the above links for usage and configuration details.

3.23.2. 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-crypto</artifactId>
</dependency>
```

3.23.3. 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.

3.24. CXF

Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.

3.24.1. What's inside

- CXF component, URI syntax: **cxf:beanId:address**

Refer to the above link for usage and configuration details.

3.24.2. 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-cxf-soap</artifactId>
</dependency>
```

3.24.3. Usage

3.24.3.1. General

camel-quarkus-cxf-soap uses extensions from the CXF Extensions for Quarkus project - **quarkus-cxf**.

This means the set of supported use cases and WS specifications is largely given by **quarkus-cxf**.

IMPORTANT

== Supported extensions

Currently, **only** these quarkus-cxf extensions are supported:

Implicitly, as transitive dependencies of **camel-quarkus-cxf-soap**:

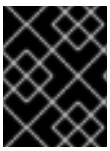
- **quarkus-cxf**
- **quarkus-cxf-rt-features-logging**

If you need WS-Security or other associated functionality, you can add the following supported extensions:

- **quarkus-cxf-rt-ws-security**
- **quarkus-cxf-services-sts**
- **quarkus-cxf-xjc-plugins**

=== WS-ReliableMessaging

Full support for CXF WS-ReliableMessaging is currently unavailable, and it remains in Technology Preview in version 3.8.



IMPORTANT

To learn about supported use cases and WS specifications, see the Quarkus CXF Reference.

3.24.3.2. Dependency management

Red Hat build of Apache Camel for Quarkus manages the CXF and **quarkus-cxf** versions. You do not need to select compatible versions for those projects.

3.24.3.3. Client

With **camel-quarkus-cxf-soap** (no additional dependencies required), you can use CXF clients as producers in Camel routes:

```
import org.apache.camel.builder.RouteBuilder;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.context.SessionScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class CxfSoapClientRoutes extends RouteBuilder {

    @Override
    public void configure() {

        /* You can either configure the client inline */
        from("direct:cxfUriParamsClient")
                .to("cxf://http://localhost:8082/calculator-ws?
```

```
wsdlURL=wsdl/CalculatorService.wsdl&dataFormat=POJO&serviceClass=org.foo.CalculatorService")
;

        /* Or you can use a named bean produced below by beanClient() method */
        from("direct:cxfBeanClient")
                .to("cxf:bean:beanClient?dataFormat=POJO");

    }

    @Produces
    @SessionScoped
    @Named
    CxfEndpoint beanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("http://localhost:8082/calculator-ws");
        result.setWsdlURL("wsdl/CalculatorService.wsdl"); // a resource in the class path
        return result;
    }
}
```

The **CalculatorService** may look like the following:

```
import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService(targetNamespace = CalculatorService.TARGET_NS) 1
public interface CalculatorService {

    public static final String TARGET_NS = "http://acme.org/wscalculator/Calculator";

    @WebMethod 2
    public int add(int intA, int intB);

    @WebMethod 3
    public int subtract(int intA, int intB);

    @WebMethod 4
    public int divide(int intA, int intB);

    @WebMethod 5
    public int multiply(int intA, int intB);
}
```

1 2 3 4 5 NOTE: JAX-WS annotations are required. The Simple CXF Frontend is not supported. Complex parameter types require JAXB annotations to work in properly in native mode.

TIP

You can test this client application against the quay.io/l2x6/calculator-ws:1.2 container that implements this service endpoint interface:

```
$ docker run -p 8082:8080 quay.io/l2x6/calculator-ws:1.2
```

NOTE

quarkus-cxf supports injecting SOAP clients using **@io.quarkiverse.cxf.annotation.CXFClient** annotation. Refer to the SOAP Clients chapter of **quarkus-cxf** user guide for more details.

3.24.3.4. Server

With **camel-quarkus-cxf-soap**, you can expose SOAP endpoints as consumers in Camel routes. No additional dependencies are required for this use case.

```
import org.apache.camel.builder.RouteBuilder;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class CxfSoapRoutes extends RouteBuilder {

    @Override
    public void configure() {
        /* A CXF Service configured through a CDI bean */
        from("cxf:bean:helloBeanEndpoint")
                .setBody().simple("Hello ${body} from CXF service");

        /* A CXF Service configured through Camel URI parameters */
        from("cxf:///hello-inline?wsdlURL=wsdl/HelloService.wsdl&serviceClass=org.foo.HelloService")
                        .setBody().simple("Hello ${body} from CXF service");
    }

    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        return result;
    }
}
```

The path under which these two services will be served depends on the value of **quarkus.cxf.path**configuration property which can for example be set in **application.properties**:

application.properties

```
quarkus.cxf.path = /soap-services
```

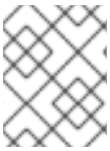
With this configuration in place, our two services can be reached under **http://localhost:8080/soap-services/hello-bean** and **http://localhost:8080/soap-services/hello-inline** respectively.

The WSDL can be accessed by adding **?wsdl** to the above URLs.

IMPORTANT

Do not use **quarkus.cxf.path =** / in your application unless you are 100% sure that no other extension will want to expose HTTP endpoints.

Before **quarkus-cxf** 2.0.0 (i.e. before Red Hat build of Apache Camel for Quarkus 3.0.0), the default value of **quarkus.cxf.path** was /. The default was changed because it prevented other Quarkus extensions from exposing any further HTTP endpoints. Among others, RESTEasy, Vert.x, SmallRye Health (no health endpoints exposed!) were impacted by this.



NOTE

quarkus-cxf supports alternative ways of exposing SOAP endpoints. Refer to the SOAP Services chapter of **quarkus-cxf** user guide for more details.

3.24.3.5. Logging of requests and responses

You can enable verbose logging of SOAP messages for both clients and servers with **org.apache.cxf.ext.logging.LoggingFeature**:

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.cxf.ext.logging.LoggingFeature;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.context.SessionScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

@ApplicationScoped
public class MyBeans {

    @Produces
    @ApplicationScoped
    @Named("prettyLoggingFeature")
    public LoggingFeature prettyLoggingFeature() {
        final LoggingFeature result = new LoggingFeature();
        result.setPrettyLogging(true);
        return result;
    }

    @Inject
    @Named("prettyLoggingFeature")
    LoggingFeature prettyLoggingFeature;

    @Produces
    @SessionScoped
    @Named
    CxfEndpoint cxfBeanClient() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(CalculatorService.class);
        result.setAddress("https://acme.org/calculator");
        result.setWsdlURL("wsdl/CalculatorService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }
```

```
    @Produces
    @ApplicationScoped
    @Named
    CxfEndpoint helloBeanEndpoint() {
        final CxfEndpoint result = new CxfEndpoint();
        result.setServiceClass(HelloService.class);
        result.setAddress("/hello-bean");
        result.setWsdlURL("wsdl/HelloService.wsdl");
        result.getFeatures().add(prettyLoggingFeature);
        return result;
    }
}
```

NOTE

The support for **org.apache.cxf.ext.logging.LoggingFeature** is provided by **io.quarkiverse.cxf:quarkus-cxf-rt-features-logging** as a **camel-quarkus-cxf-soap** dependency. You do not need to add it explicitly to your application.

3.24.3.6. WS Specifications

The extent of supported WS specifications is given by the Quarkus CXF project.

camel-quarkus-cxf-soap covers only the following specifications via the **io.quarkiverse.cxf:quarkus-cxf** extension:

- JAX-WS
- JAXB
- WS-Addressing
- WS-Policy
- MTOM

If your application requires some other WS specification, such as WS-Security or WS-Trust, you must add an additional Quarkus CXF dependency covering it. Refer to Quarkus CXF Reference page to see which WS specifications are covered by which Quarkus CXF extensions.

TIP

Both Red Hat build of Apache Camel for Quarkus and Quarkus CXF contain a number of integration tests which can serve as executable examples of applications that implement various WS specifications.

3.24.3.7. Tooling

quarkus-cxf wraps the following two CXF tools:

- **wsdl2Java** - for generating service classes from WSDL
- **java2ws** - for generating WSDL from Java classes

IMPORTANT

For **wsdl2Java** to work properly, your application will have to directly depend on **io.quarkiverse.cxf:quarkus-cxf**.

TIP

While **wsdlvalidator** is not supported, you can use **wsdl2Java** with the following configuration in **application.properties** to validate your WSDLs:

application.properties

```
quarkus.cxf.codegen.wsdl2java.additional-params = -validate
```

3.24.3.8. Possible DoS vector with CXF clients using java.net.http.HttpClient

If your CXF clients are using **java.net.http.HttpClient** as the underlying HTTP client, then due to CXF issue, the application may crash if many clients are created, as their threads do not terminated.

The problem occurs with **java.net.http.HttpClient** when CXF clients is created repeatedly, for example per request. If you keep the clients throughout the whole lifespan of the application, this issue does not occur.

Since Apache Camel for Quarkus 3.2.0 and Quarkus CXF 2.2.3, the selection of the HTTP client implementation for some specific CXF client is controlled via **quarkus.cxf.client.yourClient.http-conduit-factory** property. By default, the CXF clients created by Quarkus CXF use **java.net.HttpURLConnection** as an HTTP client and thus, this issue does not occur by default. This issue may occur if you set **quarkus.cxf.client.yourClient.http-conduit-factory=HttpClientHTTPConduitFactory**.

3.24.3.8.1. Mitigation of the DoS vector

- Only use **java.net.http.HttpClient**-backed CXF clients if you are absolutely certain that the clients are only created once during the lifespan of the application.
- Use CXF clients backed by different HTTP client implementations such as HC5 or **java.net.HttpURLConnection**.

3.25. DATA FORMAT

Use a Camel Data Format as a regular Camel Component.

For more details of the supported data formats in Red Hat build of Apache Camel for Quarkus, see Supported Data Formats.

3.25.1. What's inside

- Data Format component, URI syntax: **dataformat:name:operation**

Refer to the above link for usage and configuration details.

3.25.2. 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-dataformat</artifactId>
</dependency>
```

3.26. DATASET

Provide data for load and soak testing of your Camel application.

3.26.1. What's inside

- Dataset component, URI syntax: **dataset:name**
- DataSet Test component, URI syntax: **dataset-test:name**

Refer to the above links for usage and configuration details.

3.26.2. 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-dataset</artifactId>
</dependency>
```

3.27. DIRECT

Call another endpoint from the same Camel Context synchronously.

3.27.1. What's inside

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

Refer to the above link for usage and configuration details.

3.27.2. 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-direct</artifactId>
</dependency>
```

3.28. FHIR

Exchange information in the healthcare domain using the FHIR (Fast Healthcare Interoperability Resources) standard. Marshall and unmarshall FHIR objects to/from JSON. Marshall and unmarshall FHIR objects to/from XML.

3.28.1. What's inside

- FHIR component, URI syntax: **fhir:apiName/methodName**
- FHIR JSon data format
- FHIR XML data format

Refer to the above links for usage and configuration details.

3.28.2. 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-fhir</artifactId>
</dependency>
```

3.28.3. 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.

3.28.4. Additional Camel Quarkus configuration

By default, only FHIR versions **R4** & **DSTU3** are enabled in native mode, since they are the default values on the FHIR component and DataFormat.

Configuration property	Type	Default
quarkus.camel.fhir.enable-dstu2 Enable FHIR DSTU2 Specs in native mode.	**boolean**	**false**

Configuration property	Type	Default
quarkus.camel.fhir.enable-dstu2_hl7org Enable FHIR DSTU2_HL7ORG Specs in native mode.	boolean	false
quarkus.camel.fhir.enable-dstu2_1 Enable FHIR DSTU2_1 Specs in native mode.	boolean	false
quarkus.camel.fhir.enable-dstu3 Enable FHIR DSTU3 Specs in native mode.	boolean	false
quarkus.camel.fhir.enable-r4 Enable FHIR R4 Specs in native mode.	boolean	true
quarkus.camel.fhir.enable-r5 Enable FHIR R5 Specs in native mode.	boolean	false

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

3.29. FILE

Read and write files.

3.29.1. What's inside

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

Refer to the above link for usage and configuration details.

3.29.2. 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-file</artifactId>
</dependency>
```

3.29.3. Additional Camel Quarkus configuration

3.29.3.1. Having only a single consumer in a cluster consuming from a given endpoint

When the same route is deployed on multiple JVMs, it could be interesting to use this extension in conjunction with the Master one. In such a setup, a single consumer will be active at a time across the whole camel master namespace.

For instance, having the route below deployed on multiple JVMs:

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single JVM at a time");
```

It's possible to enable the file cluster service with a property like below:

```
quarkus.camel.cluster.file.enabled = true
quarkus.camel.cluster.file-root = target/cluster-folder-where-lock-file-will-be-held
```

As a result, a single consumer will be active across the **ns** camel master namespace. It means that, at a given time, only a single timer will generate exchanges across all JVMs. In other words, messages will be logged every 100ms on a single JVM at a time.

The file cluster service could further be tuned by tweaking **quarkus.camel.cluster.file.*** properties.

Configuration property	Type	Default

Configuration property	Type	Default
quarkus.camel.cluster.file.enabled Whether a File Lock Cluster Service should be automatically configured according to 'quarkus.camel.cluster.file.*' configurations.	boolean	false
quarkus.camel.cluster.file-id The cluster service ID (defaults to null).	string	
quarkus.camel.cluster.file-root The root path (defaults to null).	string	
quarkus.camel.cluster.file-order The service lookup order/priority (defaults to 2147482647).	java.lang.Integer	
quarkus.camel.cluster.file.acquire-lock-delay The time to wait before starting to try to acquire lock (defaults to 1000ms).	string	
quarkus.camel.cluster.file.acquire-lock-interval The time to wait between attempts to try to acquire lock (defaults to 10000ms).	string	

Configuration property	Type	Default
quarkus.camel.cluster.file.attributes The custom attributes associated to the service (defaults to empty map).	Map<String, String>	

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

3.30. FLINK

Send DataSet jobs to an Apache Flink cluster.

3.30.1. What's inside

- Flink component, URI syntax: **flink:endpointType**

Refer to the above link for usage and configuration details.

3.30.2. Maven coordinates

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

3.31. FTP

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

3.31.1. 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**

Refer to the above links for usage and configuration details.

3.31.2. 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-ftp</artifactId>
</dependency>
```

3.32. GOOGLE BIGQUERY

Access Google Cloud BigQuery service using SQL queries or Google Client Services API

3.32.1. What's inside

- Google BigQuery component, URI syntax: **google-bigquery:projectId:datasetId:tableId**
- Google BigQuery Standard SQL component, URI syntax: **google-bigquery-sql:projectId:queryString**

Refer to the above links for usage and configuration details.

3.32.2. 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-google-bigquery</artifactId>
</dependency>
```

3.32.3. Usage

If you want to read SQL scripts from the classpath with **google-bigquery-sql** in native mode, then you will need to ensure that they are added to the native image via the **quarkus.native.resources.includes** configuration property. Please check Quarkus documentation for more details.

3.33. GOOGLE PUBSUB

Send and receive messages to/from Google Cloud Platform PubSub Service.

3.33.1. What's inside

- Google Pubsub component, URI syntax: **google-pubsub:projectId:destinationName**

Refer to the above link for usage and configuration details.

3.33.2. 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-google-pubsub</artifactId>
</dependency>
```

3.33.3. Camel Quarkus limitations

By default, the Camel PubSub component uses JDK object serialization via **ObjectOutputStream** whenever the message body is anything other than **String** or **byte[]**.

Since such serialization is not yet supported by GraalVM, this extension provides a custom Jackson based serializer to serialize complex message payloads as JSON.

If your payload contains binary data, then you will need to handle that by creating a custom Jackson Serializer / Deserializer. Refer to the Quarkus Jackson guide for information on how to do this.

3.34. GRPC

Expose gRPC endpoints and access external gRPC endpoints.

3.34.1. What's inside

- gRPC component, URI syntax: **grpc:host:port/service**

Refer to the above link for usage and configuration details.

3.34.2. 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-grpc</artifactId>
</dependency>
```

3.34.3. Usage

3.34.3.1. Protobuf generated code

Camel Quarkus gRPC can generate gRPC service stubs for **.proto** files. When using Maven, ensure that you have enabled the **generate-code** goals of the **quarkus-maven-plugin** in your project build.

```
<build>
  <plugins>
    <plugin>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

With this configuration, you can put your service and message definitions into the **src/main/proto** directory and the **quarkus-maven-plugin** will generate code from your **.proto** files.

3.34.3.1.1. Scanning proto files with imports

The Protocol Buffers specification provides a way to import proto files. You can control the scope of dependencies to scan by adding configuration property **quarkus.camel.grpc.codegen.scan-for-imports** property to **application.properties**. The available options are outlined below.

- **all** - Scan all dependencies
- **none** - Disable dependency scanning. Use only the proto definitions defined in **src/main/proto** or **src/test/proto**
- **groupId1:artifactId1,groupId2:artifactId2** - Scan only the dependencies matching the **groupId** and **artifactId** list

The default value is **com.google.protobuf:protobuf-java**.

3.34.3.1.2. Scanning proto files from dependencies

If you have proto files shared across multiple dependencies, you can generate gRPC service stubs for them by adding configuration property **quarkus.camel.grpc.codegen.scan-for-proto** to **application.properties**.

First add a dependency for the artifact(s) containing proto files to your project. Next, enable proto file dependency scanning.

```
quarkus.camel.grpc.codegen.scan-for-proto=org.my.groupId1:my-artifact-id-1,org.my.groupId2:my-artifact-id-2
```

It is possible to include / exclude specific proto files from dependency scanning via configuration properties.

The configuration property name suffix is the Maven **groupId** / **artifactId** for the dependency to configure includes / excludes on. Paths are relative to the classpath location of the proto files within the dependency. Paths can be an explicit path to a proto file, or as glob patterns to include / exclude multiple files.

```
quarkus.camel.grpc.codegen.scan-for-proto-includes."<groupId>\:<artifactId>"=foo/**,bar/**,baz/a-proto.proto
quarkus.camel.grpc.codegen.scan-for-proto-excludes."<groupId>\:<artifactId>"=foo/private/**,baz/another-proto.proto
```

NOTE

The **:** character within property keys must be escaped with ****.

3.34.3.2. Accessing classpath resources in native mode

The gRPC component has various options where resources are resolved from the classpath:

- **keyCertChainResource**
- **keyResource**
- **serviceAccountResource**
- **trustCertCollectionResource**

When using these options in native mode, you must ensure that any such resources are included in the native image.

This can be accomplished by adding the configuration property **quarkus.native.resources.includes** to **application.properties**. For example, to include SSL / TLS keys and certificates.

```
quarkus.native.resources.includes = certs/*.pem,certs.*.key
```

3.34.4. Camel Quarkus limitations

3.34.4.1. Integration with Quarkus gRPC is not supported

At present there is no support for integrating Camel Quarkus gRPC with Quarkus gRPC. If you have both the **camel-quarkus-grpc** and **quarkus-grpc** extension dependency on the classpath, you are likely to encounter problems at build time when compiling your application.

3.34.5. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.grpc.codegen.enabled If **true**, Camel Quarkus gRPC code generation is run for .proto files discovered from the **proto** directory, or from dependencies specified in the **scan-for-proto** or **scan-for-imports** options. When **false**, code generation for .proto files is disabled.	**boolean**	**true**
quarkus.camel.grpc.codegen.scan-for-proto Camel Quarkus gRPC code generation can scan application dependencies for .proto files to generate Java stubs from them. This property sets the scope of the dependencies to scan. Applicable values: - *none* - default - don't scan dependencies - a comma separated list of *groupId:artifactId* coordinates to scan - *all* - scan all dependencies	**string**	**none**
quarkus.camel.grpc.codegen.scan-for-imports Camel Quarkus gRPC code generation can scan dependencies for .proto files that can be imported by protos in this applications. Applicable values: - *none* - default - don't scan dependencies - a comma separated list of *groupId:artifactId* coordinates to scan - *all* - scan all dependencies The default is *com.google.protobuf:protobuf-java*.	**string**	**com.google.protobuf:protobuf-java**

Configuration property	Type	Default
quarkus.camel.grpc.codegen.scan-for-proto-includes Package path or file glob pattern includes per dependency containing .proto files to be considered for inclusion.	Map<String, List<String>>	
quarkus.camel.grpc.codegen.scan-for-proto-excludes Package path or file glob pattern includes per dependency containing .proto files to be considered for exclusion.	Map<String, List<String>>	

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

3.35. GSON

Marshal POJOs to JSON and back using Gson

3.35.1. What's inside

- JSON Gson data format

Refer to the above link for usage and configuration details.

3.35.2. 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-gson</artifactId>
</dependency>
```

3.35.3. Additional Camel Quarkus configuration

3.35.3.1. Marshaling/Unmarshaling objects in native mode

When marshaling/unmarshaling objects in native mode, all the serialized classes need to be registered for reflection. As such, when using **GsonDataFormat.setUnmarshalType(...)**, **GsonDataFormat.setUnmarshalTypeName(...)** and even **GsonDataFormat.setUnmarshalGenericType(...)**, the unmarshal type as well as sub field types should be registered for reflection. See a working example in this integration test.

3.36. HL7

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

3.36.1. What's inside

- HL7 data format
- HL7 Terser language

Refer to the above links for usage and configuration details.

3.36.2. 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-hl7</artifactId>
</dependency>
```

3.36.3. Camel Quarkus limitations

For MLLP with TCP, Netty is the only supported means of running an HI7 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**.

3.37. HTTP

Send requests to external HTTP servers using Apache HTTP Client 5.x.

3.37.1. What's inside

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

Refer to the above links for usage and configuration details.

3.37.2. 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-http</artifactId>
</dependency>
```

3.37.3. 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.

3.37.4. Additional Camel Quarkus configuration

- Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

3.38. INFINISPAN

Read and write from/to Infinispan distributed key/value store and data grid.

3.38.1. What's inside

- Infinispan component, URI syntax: **infinispan:cacheName**

Refer to the above link for usage and configuration details.

3.38.2. 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-infinispan</artifactId>
</dependency>
```

3.38.3. Additional Camel Quarkus configuration

3.38.3.1. Infinispan Client Configuration

You can either configure the Infinispan client via the relevant Camel Infinispan component & endpoint options, or you may use the Quarkus Infinispan extension configuration properties .

Note that if you choose to use Quarkus Infinispan configuration properties, you **must** add an injection point for the **RemoteCacheManager** in order for it to be discoverable by the Camel Infinispan component. For example:

```
public class Routes extends RouteBuilder {
    // Injects the default unnamed RemoteCacheManager
    @Inject
    RemoteCacheManager cacheManager;

    // If configured, injects an optional named RemoteCacheManager
    @Inject
    @InfinispanClientName("myNamedClient")
    RemoteCacheManager namedCacheManager;

    @Override
    public void configure() {
        // Route configuration here...
    }
}
```

3.38.3.2. Camel Infinispan **InfinispanRemoteAggregationRepository** in native mode

If you chose to use the **InfinispanRemoteAggregationRepository** in native mode, then you must enable native serialization support.

3.39. AVRO JACKSON

Marshal POJOs to Avro and back using Jackson.

3.39.1. What's inside

- Avro Jackson data format

Refer to the above link for usage and configuration details.

3.39.2. 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-jackson-avro</artifactId>
</dependency>
```

3.40. PROTOBUF JACKSON

Marshal POJOs to Protobuf and back using Jackson.

3.40.1. What's inside

- Protobuf Jackson data format

Refer to the above link for usage and configuration details.

3.40.2. 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-jackson-protobuf</artifactId>
</dependency>
```

3.41. JACKSON

Marshal POJOs to JSON and back using Jackson

3.41.1. What's inside

- JSON Jackson data format

Refer to the above link for usage and configuration details.

3.41.2. 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-jackson</artifactId>
</dependency>
```

3.41.3. Usage

3.41.3.1. Configuring the Jackson ObjectMapper

There are a few ways of configuring the **ObjectMapper** that the **JacksonDataFormat** uses. These are outlined below.

3.41.3.1.1. ObjectMapper created internally by JacksonDataFormat

By default, **JacksonDataFormat** will create its own **ObjectMapper** and use the various configuration options on the **DataFormat** to configure additional Jackson modules, pretty printing and other features.

3.41.3.1.2. Custom ObjectMapper for JacksonDataFormat

You can pass a custom **ObjectMapper** instance to **JacksonDataFormat** as follows.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        ObjectMapper mapper = new ObjectMapper();
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

3.41.3.1.3. Using the Quarkus Jackson ObjectMapper with JacksonDataFormat

The Quarkus Jackson extension exposes an **ObjectMapper** CDI bean which can be discovered by the **JacksonDataFormat**.

```
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

public class Routes extends RouteBuilder {
    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        // Make JacksonDataFormat discover the Quarkus Jackson `ObjectMapper` from the Camel
registry
        dataFormat.setAutoDiscoverObjectMapper(true);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

If you are using the JSON binding mode in the Camel REST DSL and want to use the Quarkus Jackson **ObjectMapper**, it can be achieved as follows.

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

```
@ApplicationScoped
public class Routes extends RouteBuilder {
    public void configure() {
        restConfiguration().dataFormatProperty("autoDiscoverObjectMapper", "true");
        // REST definition follows...
    }
}
```

You can perform customizations on the Quarkus **ObjectMapper** with a **ObjectMapperCustomizer**.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import io.quarkus.jackson.ObjectMapperCustomizer;

@Singleton
public class RegisterCustomModuleCustomizer implements ObjectMapperCustomizer {
    public void customize(ObjectMapper mapper) {
        mapper.registerModule(new CustomModule());
    }
}
```

It's also possible to **@Inject** the Quarkus **ObjectMapper** and pass it to the **JacksonDataFormat**.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.jackson.JacksonDataFormat;

@ApplicationScoped
public class Routes extends RouteBuilder {
    @Inject
    ObjectMapper mapper;

    public void configure() {
        JacksonDataFormat dataFormat = new JacksonDataFormat();
        dataFormat.setObjectMapper(mapper);
        // Use the dataFormat instance in a route definition
        from("direct:my-direct").marshal(dataFormat)
    }
}
```

3.42. JACKSONXML

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

3.42.1. What's inside

- Jackson XML data format

Refer to the above link for usage and configuration details.

3.42.2. 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-jacksonxml</artifactId>
</dependency>
```

3.43. JASYPT

Security using Jasypt

3.43.1. What's inside

- Jasypt

Refer to the above link for usage and configuration details.

3.43.2. 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-jasypt</artifactId>
</dependency>
```

3.43.3. Usage

The configuration of Jasypt in Camel Quarkus is driven by configuration properties.

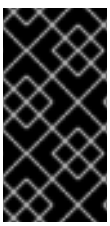
The minimum expectation is that you provide a master password for Jasypt decryption with configuration property **quarkus.camel.jasypt.password**.

You can choose the encryption algorithm and other aspects of the Jasypt configuration via the **quarkus.camel.jasypt** options described below.

By default, you do not need to write custom code to configure the Camel **JasyptPropertiesParser** or **PropertiesComponent**. This is done for you automatically.

Any Camel configuration property added to **application.properties** can be secured with Jasypt. To encrypt a value, there is a utility that can be run with JBang.

```
jbang org.apache.camel:camel-jasypt:{camel-version} -c encrypt -p secret-password -i "Some secret content"
```



IMPORTANT

If you choose to use a different Jasypt algorithm to the default (**PBEWithMD5AndDES**), you must provide **-a** (algorithm), **-riga** (IV generator algorithm) & **-rsga** (Salt generator algorithm) arguments to set the correct algorithms used in encryption. Else your application will not be able to decrypt configuration values.

Alternatively, when running in dev mode, open the Dev UI and click the 'utilities' link in the Camel Jasypt pane. Next, select either the 'Decrypt' or 'Encrypt' action, enter some text and click the submit button. The result of the action is output together with a button to copy it to the clipboard.

Configuration properties can be added to **application.properties** with the encrypted value enclosed within **ENC()** For example.

```
my.secret = ENC(BoDSRQfdBME4V/AcugPOkaR+IcyKufGz)
```

In your Camel routes, you can refer to the property name using the standard placeholder syntax and its value will get decrypted.

```
public class MySecureRoute extends RouteBuilder {
    @Override
    public void configure() {
        from("timer:tick?period=5s")
            .to("{{my.secret}}");
    }
}
```

TIP

You can use the ability to mask security sensitive configuration in Camel by suffixing property values with **.secret**. You can also disable the startup configuration summary with the configuration **camel.main.autoConfigurationLogSummary = false**.

3.43.3.1. Injecting encrypted configuration

You can use the **@ConfigProperty** annotation to inject encrypted configuration into your Camel routes or CDI beans.

```
@ApplicationScoped
public class MySecureRoute extends RouteBuilder {
    @ConfigInject("my.secret")
    String mySecret;

    @Override
    public void configure() {
        from("timer:tick?period=5s")
            .to(mySecret);
    }
}
```

3.43.3.1.1. Securing alternate configuration sources

If you prefer to keep your secret configuration in a file separate to **application.properties**, you can use the **quarkus.config.locations** configuration option to specify additional configuration files.

In native mode you must also add any additional configuration file resource paths to **quarkus.native.resources.includes**.

3.43.3.1.2. Finer control of Jasypt configuration

If you require finer control of the Jasypt configuration than that provided by the default configuration, the following options are available.

3.43.3.1.2.1. JasyptConfigurationCustomizer

Implement a **JasyptConfigurationCustomizer** class to customize any aspect of the Jasypt **EnvironmentStringPBEConfig**.

```
package org.acme;

import org.apache.camel.quarkus.component.jasypt.JasyptConfigurationCustomizer;
import org.jasypt.encryption.pbe.config.EnvironmentStringPBEConfig;
import org.jasypt.iv.RandomIvGenerator;
import org.jasypt.salt.RandomSaltGenerator;

public class JasyptConfigurationCustomizer implements JasyptConfigurationCustomizer {
    public void customize(EnvironmentStringPBEConfig config) {
        // Custom algorithms
        config.setAlgorithm("PBEWithHmacSHA256AndAES_256");
        config.setSaltGenerator(new RandomSaltGenerator("PKCS11"));
        config.setIvGenerator(new RandomIvGenerator("PKCS11"));
        // Additional customizations...
    }
}
```

In **application.properties** add the **quarkus.camel.jasypt.configuration-customizer-class-name** configuration property.

```
quarkus.camel.jasypt.configuration-customizer-class-name =
org.acme.MyJasyptEncryptorCustomizer
```

3.43.3.1.2.2. Disabling automatic Jasypt configuration

If you prefer to use the 'classic' Java DSL way of configuring Camel Jasypt, you can disable the automatic configuration with **quarkus.camel.jasypt.enabled = false**.

This allows you to configure the Camel **JasyptPropertiesParser** and **PropertiesComponent** manually.

NOTE

In this mode, you cannot use the **@ConfigProperty** annotation to inject encrypted configuration properties.

```
import org.apache.camel.CamelContext;
import org.apache.camel.component.jasypt.JasyptPropertiesParser;
import org.apache.camel.component.properties.PropertiesComponent;

public class MySecureRoute extends RouteBuilder {
    @Override
    public void configure() {
        JasyptPropertiesParser jasypt = new JasyptPropertiesParser();
        jasypt.setPassword("secret");

        PropertiesComponent component = (PropertiesComponent)
```

```
getContext().getPropertiesComponent();
        jasypt.setPropertiesComponent(component);
        component.setPropertiesParser(jasypt);

        from("timer:tick?period=5s")
            .to("{{my.secret}}");
    }
}
```

NOTE

If you call **setLocation(...)** on the **PropertiesComponent** to specify a custom configuration file location using the **classpath:** prefix, you must add the file to **quarkus.native.resources.includes** so that it can be loaded in native mode.

3.43.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.jasypt.enabled Setting this option to false will disable Jasypt integration with Quarkus SmallRye configuration. You can however, manually configure Jasypt with Camel in the 'classic' way of manually configuring JasyptPropertiesParser and PropertiesComponent. Refer to the usage section for more details.	**boolean**	**true**
quarkus.camel.jasypt.algorithm The algorithm to be used for decryption.	**string**	**PBEWithMD5AndDES**
quarkus.camel.jasypt.password The master password used by Jasypt for decrypting configuration values. This option supports prefixes which influence the master password lookup behaviour. **sys:** will to look up the value from a JVM system property.**sysenv:** will look up the value from the OS system environment with the given key.	**string**	
quarkus.camel.jasypt.random-iv-generator-algorithm Configures the Jasypt StandardPBEStringEncryptor with a RandomIvGenerator using the given algorithm.	**string**	**SHA1PRNG**

Configuration property	Type	Default
quarkus.camel.jasypt.random-salt-generator-algorithm Configures the Jasypt StandardPBEStringEncryptor with a RandomSaltGenerator using the given algorithm.	string	SHA1PRNG
quarkus.camel.jasypt.configuration-customizer-class-name The fully qualified class name of an org.apache.camel.quarkus.component.jasypt.JasyptConfigurationCustomizer implementation. This provides the optional capability of having full control over the Jasypt configuration.	string	

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

3.43.5. Camel Quarkus limitations

3.43.6. Jasypt: quarkus.camel.jasypt.enabled=false not working

The **camel-quarkus-jasypt** extension has an issue resolving configuration properties when they are provided from system properties or environment variables. For example, passing **-Dquarkus.camel.jasypt.enabled=false** or **-Dquarkus.camel.jasypt.password=my-password** does not work.

To work around this you can specify **quarkus.camel.jasypt.enabled** in **application.properties**.

Disable jasypt

```
quarkus.camel.jasypt.enabled = false
```

To override **quarkus.camel.jasypt.password** from a system property or environment variable you can configure **application.properties** as follows:

Either hard code the password in application.properties

Hardcoded password

```
quarkus.camel.jasypt.password = my-password
```

Or it's possible to have the password resolved from a system property or environment variable using the 'sys' or 'sysenv' prefix.

Password with sys prefix

```
quarkus.camel.jasypt.password = sys:jasyptPassword
```

Then build the application with your desired password.

Building with password

```
mvn clean package -DjasyptPassword=my-password
```

And when running the application JAR.

Application JAR

```
java -DjasyptPassword=my-password -jar target/quarkus-app/quarkus-run.jar
```

Or in native mode:

Native mode

```
target/my-native-application-runner -DjasyptPassword=my-password
```

3.44. JAVA JOOR DSL

Support for parsing Java route definitions at runtime

3.44.1. What's inside

- Java DSL (runtime compiled)

Refer to the above link for usage and configuration details.

3.44.2. 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-java-joor-dsl</artifactId>
</dependency>
```

3.44.3. Camel Quarkus limitations

The annotations added to the classes to be compiled by the component are ignored by Quarkus. The only annotation that is partially supported by the extension is the annotation **RegisterForReflection** to ease the configuration of the reflection for the native mode however please note that the element **registerFullHierarchy** is not supported.

3.45. JAXB

Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.

3.45.1. What's inside

- JAXB data format

Refer to the above link for usage and configuration details.

3.45.2. 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-jaxb</artifactId>
</dependency>
```

3.45.3. Usage

3.45.3.1. Native mode ObjectFactory instantiation of non-JAXB annotated classes

When performing JAXB marshal operations with a custom **ObjectFactory** to instantiate POJO classes that do not have JAXB annotations, you must register those POJO classes for reflection in order for them to be instantiated in native mode. E.g via the **@RegisterForReflection** annotation or configuration property **quarkus.camel.native.reflection.include-patterns**.

Refer to the Native mode user guide for more information.

3.46. JDBC

Access databases through SQL and JDBC.

3.46.1. What's inside

- JDBC component, URI syntax: **jdbc:dataSourceName**

Refer to the above link for usage and configuration details.

3.46.2. 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-jdbc</artifactId>
</dependency>
```

3.46.3. Additional Camel Quarkus configuration

3.46.3.1. Configuring a DataSource

This extension leverages Quarkus Agroal for **DataSource** support. Setting up a **DataSource** can be achieved via configuration properties. It is recommended that you explicitly name the datasource so that it can be referenced in the JDBC endpoint URI. E.g like **to("jdbc:camel")**.

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

If you choose to not name the datasource, you can resolve the default **DataSource** by defining your endpoint like **to("jdbc:default")**.

3.46.3.1.1. Zero configuration with Quarkus Dev Services

In dev and test mode you can take advantage of Configuration Free Databases. All you need to do is reference the default database in your routes. E.g **to("jdbc:default")**.

3.47. JIRA

Interact with JIRA issue tracker.

3.47.1. What's inside

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

Refer to the above link for usage and configuration details.

3.47.2. 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>
```

NOTE

Applications using the camel-quarkus-jira extension require an additional Maven repository https://packages.atlassian.com/maven-external/ to be configured either in the Maven settings.xml file or in the pom.xml file of the application project.

3.47.3. 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.

3.48. JMS

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

3.48.1. What's inside

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

Refer to the above link for usage and configuration details.

3.48.2. 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>
```

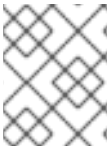
3.48.3. Usage

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

The Camel JMS component supports message mapping between **jakarta.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-xml-jaxp** extension is present on the classpath.

3.48.3.2. Native mode support for jakarta.jms.ObjectMessage

When sending JMS message payloads as **jakarta.jms.ObjectMessage**, you must annotate the relevant classes to be registered for serialization with **@RegisterForReflection(serialization = true)**.



NOTE

This extension automatically sets **quarkus.camel.native.reflection.serialization-enabled = true** for you. Refer to the native mode user guide for more information.

3.48.3.3. Support for Connection pooling and X/Open XA distributed transactions

NOTE

To use connection pooling in the **camel-quarkus-jms** components, you must add **io.quarkiverse.artemis:quarkus-artemis** and **io.quarkiverse.messaginghub:quarkus-pooled-jms** to your pom.xml and set the following configuration:

```
quarkus.pooled-jms.max-connections = 8
```

You can use the **quarkus-pooled-jms** extension to get pooling and XA support for JMS connections. Refer to the quarkus-pooled-jms extension documentation for more information. Currently, it can work with **quarkus-artemis-jms**, **quarkus-qpid-jms** and **ibmmq-client**. Just add the dependency to your **pom.xml**:

```
<dependency>
```

```
    <groupId>io.quarkiverse.messaginghub</groupId>
    <artifactId>quarkus-pooled-jms</artifactId>
</dependency>
```

Pooling is enabled by default.

NOTE

clientID and **durableSubscriptionName** are not supported in pooling connections. If **setClientID** is called on a **reused** connection from the pool, an **IllegalStateException** will be thrown. You will get some error messages such like **Cause: setClientID can only be called directly after the connection is created**

To enable XA, you need to add **quarkus-narayana-jta** extension:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-narayana-jta</artifactId>
</dependency>
```

and add the following configuration to your **application.properties**:

```
quarkus.pooled-jms.transaction=xa
quarkus.transaction-manager.enable-recovery=true
```

XA support is only available with **quarkus-artemis-jms** and **ibmmq-client**.

We strongly recommend that you enable *transaction recovery*.

Since there currently exists no quarkus extension for **ibmmq-client**, you need to create a custom **ConnectionFactory** and wrap it yourself.

Here is an example:

Wrapper example: ConnectionFactory for ibmmq-client

```
@Produces
public ConnectionFactory createXAConnectionFactory(PooledJmsWrapper wrapper) {
    MQXAConnectionFactory mq = new MQXAConnectionFactory();
    try {
        mq.setHostName(ConfigProvider.getConfig().getValue("ibm.mq.host", String.class));
        mq.setPort(ConfigProvider.getConfig().getValue("ibm.mq.port", Integer.class));
        mq.setChannel(ConfigProvider.getConfig().getValue("ibm.mq.channel", String.class));
        mq.setQueueManager(ConfigProvider.getConfig().getValue("ibm.mq.queueManagerName",
String.class));
        mq.setTransportType(WMQConstants.WMQ_CM_CLIENT);
        mq.setStringProperty(WMQConstants.USERID,
            ConfigProvider.getConfig().getValue("ibm.mq.user", String.class));
        mq.setStringProperty(WMQConstants.PASSWORD,
            ConfigProvider.getConfig().getValue("ibm.mq.password", String.class));
    } catch (Exception e) {
        throw new RuntimeException("Unable to create new IBM MQ connection factory", e);
```

```
    }
    return wrapper.wrapConnectionFactory(mq);
}
```

NOTE

If you use **ibmmq-client** to consume messages and enable XA, you need to configure **TransactionManager** in the camel route like this:

```
@Inject
TransactionManager transactionManager;

@Override
public void configure() throws Exception {
    from("jms:queue:DEV.QUEUE.XA?transactionManager=#jtaTransactionManager");
}

@Named("jtaTransactionManager")
public PlatformTransactionManager getTransactionManager() {
    return new JtaTransactionManager(transactionManager);
}
```

Otherwise, you will get an exception like **MQRC_SYNCPOINT_NOT_AVAILABLE**.

NOTE

When you are using **ibmmq-client** and rollback a transaction, there will be a WARN message like:

```
WARN  [com.arj.ats.jta] (executor-thread-1) ARJUNA016045: attempted rollback of <
formatId=131077, gtrid_length=35, bqual_length=36,
tx_uid=0:ffffc0a86510:aed3:650915d7:16, node_name=quarkus,
branch_uid=0:ffffc0a86510:aed3:650915d7:1f, subordinatenodename=null,
eis_name=0 > (com.ibm.mq.jmqi.JmqiXAResource@79786dde) failed with exception
code XAException.XAER_NOTA: javax.transaction.xa.XAException: The method
'xa_rollback' has failed with errorCode '-4'.
```

you can ignore it and assume that MQ has discarded the transaction's work. Refer to Red Hat Knowledgebase for more information.

3.48.4. 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)
```

3.49. JPA

Store and retrieve Java objects from databases using Java Persistence API (JPA).

3.49.1. What's inside

- JPA component, URI syntax: **jpa:entityType**

Refer to the above link for usage and configuration details.

3.49.2. 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-jpa</artifactId>
</dependency>
```

3.49.3. Additional Camel Quarkus configuration

The extension leverages Quarkus Hibernate ORM to provide the JPA implementation via Hibernate.

Refer to the Quarkus Hibernate ORM documentation to see how to configure Hibernate and your datasource.

Also, it leverages Quarkus TX API to provide **TransactionStrategy** implementation.

When a single persistence unit is used, the Camel Quarkus JPA extension will automatically configure the JPA component with a **EntityManagerFactory** and **TransactionStrategy**.

3.49.3.1. Configuring JpaMessageIdRepository

It needs to use **EntityManagerFactory** and **TransactionStrategy** from the CDI container to configure the **JpaMessageIdRepository**:

```
@Inject
EntityManagerFactory entityManagerFactory;

@Inject
TransactionStrategy transactionStrategy;

from("direct:idempotent")
    .idempotentConsumer(
        header("messageId"),
        new JpaMessageIdRepository(entityManagerFactory, transactionStrategy,
"idempotentProcessor"));
```

NOTE

Since it excludes the **spring-orm** dependency, some options such as **sharedEntityManager**, **transactionManager** are not supported.

3.50. JSLT

Query or transform JSON payloads using an JSLT.

3.50.1. What's inside

- JSLT component, URI syntax: **jslt:resourceUri**

Refer to the above link for usage and configuration details.

3.50.2. 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-jslt</artifactId>
</dependency>
```

3.50.3. allowContextMapAll option in native mode

The **allowContextMapAll** option is not supported in native mode as it requires reflective access to security sensitive camel core classes such as **CamelContext** & **Exchange**. This is considered a security risk and thus access to the feature is not provided by default.

3.50.4. Additional Camel Quarkus configuration

3.50.4.1. Loading JSLT templates from classpath in native mode

This component typically loads the templates from classpath. To make it work also in native mode, you need to explicitly embed the templates files in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load the JSLT schema from a classpath resource named **transformation.json**:

```
from("direct:start").to("jslt:transformation.json");
```

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

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

3.50.4.2. Using JSLT functions in native mode

When using JSLT functions from camel-quarkus in native mode, the classes hosting the functions would need to be registered for reflection. When registering the target function is not possible, one may end up writing a stub as below.

```
@RegisterForReflection
```

```
public class MathFunctionStub {
    public static double pow(double a, double b) {
        return java.lang.Math.pow(a, b);
    }
}
```

The target function **Math.pow(...)** is now accessible through the **MathFunctionStub** class that could be registered in the component as below:

```
@Named
JsltComponent jsltWithFunction() throws ClassNotFoundException {
    JsltComponent component = new JsltComponent();
    component.setFunctions(singleton(wrapStaticMethod("power",
"org.apache.cq.example.MathFunctionStub", "pow")));
    return component;
}
```

3.51. JSON PATH

Evaluate a JSONPath expression against a JSON message body

3.51.1. What's inside

- JSONPath language

Refer to the above link for usage and configuration details.

3.51.2. 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>
```

3.52. JTA

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

3.52.1. What's inside

- JTA

Refer to the above link for usage and configuration details.

3.52.2. 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>
```

3.52.3. 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.

3.53. JT400

Exchanges messages with an IBM i system using data queues, message queues, or program call. IBM i is the replacement for AS/400 and iSeries servers.

3.53.1. What's inside

- JT400 component, URI syntax: **jt400:userID:password@systemName/QSYS.LIB/objectPath.type**

Refer to the above link for usage and configuration details.

3.53.2. Maven coordinates

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-jt400</artifactId>
</dependency>
```

WARNING

When using the native extension, you may get an error like

> Resource bundle lookup must be loaded during native image generation:

This is caused by missing native registration (https://github.com/apache/camel-quarkus/pull/6029)

As a workaround, you can include multiple resource bundles:

```
quarkus.native.additional-build-args = -
H:IncludeResourceBundles=com.ibm.as400.access.JDMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.SVMRI_en,-
H:IncludeResourceBundles=com.ibm.as400.access.MRI2,-
H:IncludeResourceBundles=com.ibm.as400.access.JDMRI2,-
H:IncludeResourceBundles=com.ibm.as400.access.SVMRI,-
H:IncludeResourceBundles=com.ibm.as400.data.DAMRI,-
H:IncludeResourceBundles=com.ibm.as400.security.SecurityMRI,-
H:IncludeResourceBundles=com.ibm.as400.util.commtrace.CTMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.CoreMRI,-
H:IncludeResourceBundles=com.ibm.as400.resource.ResourceMRI,-
H:IncludeResourceBundles=com.ibm.as400.access.MRI
```

3.54. KAFKA

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

3.54.1. What's inside

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

Refer to the above link for usage and configuration details.

3.54.2. 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>
```

3.54.3. Usage

3.54.3.1. Quarkus Kafka Dev Services

Camel Quarkus Kafka can take advantage of Quarkus Kafka Dev services to simplify development and testing with a local containerized Kafka broker.

Kafka Dev Services is enabled by default in dev & test mode. The Camel Kafka component is automatically configured so that the **brokers** component option is set to point at the local containerized Kafka broker. Meaning that there's no need to configure this option yourself.

This functionality can be disabled with the configuration property **quarkus.kafka.devservices.enabled=false**.

3.54.4. 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.

3.55. KAMELET

Materialize route templates

3.55.1. What's inside

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

Refer to the above link for usage and configuration details.

3.55.2. 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>
```

3.55.3. Usage

3.55.3.1. Pre-load Kamelets at build-time

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

3.55.3.2. Using the Kamelet Catalog

A set of pre-made Kamelets can be found on the /camel-kamelets/latest[Kamelet Catalog]. To use the Kamelet from the catalog you need to copy their yaml definition (that you can find in the camel-kamelet repo) on your project in the classpath. Alternatively you can add the **camel-kamelets-catalog** artifact to your **pom.xml**:

```
<dependency>
    <groupId>org.apache.camel.kamelets</groupId>
    <artifactId>camel-kamelets-catalog</artifactId>
</dependency>
```

This artifact add all the kamelets available in the catalog to your Camel Quarkus application for build time processing. If you include it with the scope **provided** the artifact should not be part of the runtime classpath, but at build time, all the kamelets listed via **quarkus.camel.kamelet.identifiers** property should be preloaded.

3.55.4. 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.

3.56. KUBERNETES

Perform operations against Kubernetes API

3.56.1. 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-kubernetes</artifactId>
</dependency>
```

3.56.2. Additional Camel Quarkus configuration

IMPORTANT

In this release of Red Hat build of Apache Camel for Quarkus, the **camel-quarkus-kubernetes** extension is only supported when used with the **camel-quarkus-master** extension as a cluster service. Additionally, in order for the **camel-quarkus-kubernetes** extension to be supported, you must explicitly add a dependency on the **quarkus-openshift-client** extension in your application.

3.56.2.1. Automatic registration of a Kubernetes Client instance

The extension automatically registers a Kubernetes Client bean named **kubernetesClient**. You can reference the bean in your routes like this:

```
from("direct:pods")
    .to("kubernetes-pods:///?kubernetesClient=#kubernetesClient&operation=listPods")
```

By default the client is configured from the local kubeconfig file. You can customize the client configuration via properties within **application.properties**:

```
quarkus.kubernetes-client.master-url=https://my.k8s.host
quarkus.kubernetes-client.namespace=my-namespace
```

The full set of configuration options are documented in the Quarkus Kubernetes Client guide.

3.56.2.2. Having only a single consumer in a cluster consuming from a given endpoint

When the same route is deployed on multiple pods, it could be interesting to use this extension in conjunction with the Master one. In such a setup, a single consumer will be active at a time across the whole camel master namespace.

For instance, having the route below deployed on multiple pods:

```
from("master:ns:timer:test?period=100").log("Timer invoked on a single pod at a time");
```

It's possible to enable the kubernetes cluster service with a property like below:

```
quarkus.camel.cluster.kubernetes.enabled = true
```

As a result, a single consumer will be active across the **ns** camel master namespace. It means that, at a given time, only a single timer will generate exchanges across the whole cluster. In other words, messages will be logged every 100ms on a single pod at a time.

The kubernetes cluster service could further be tuned by tweaking **quarkus.camel.cluster.kubernetes.*** properties.

Configuration property	Type	Default
quarkus.camel.cluster.kubernetes.enabled Whether a Kubernetes Cluster Service should be automatically configured according to 'quarkus.camel.cluster.kubernetes.*' configurations.	**boolean**	**false**
quarkus.camel.cluster.kubernetes-id The cluster service ID (defaults to null).	**string**	
quarkus.camel.cluster.kubernetes.master-url The URL of the Kubernetes master (read from Kubernetes client properties by default).	**string**	
quarkus.camel.cluster.kubernetes.connection-timeout-millis The connection timeout in milliseconds to use when making requests to the Kubernetes API server.	**java.lang.Integer**	

Configuration property	Type	Default
quarkus.camel.cluster.kubernetes.namespace The name of the Kubernetes namespace containing the pods and the configmap (autodetected by default).	string	
quarkus.camel.cluster.kubernetes.pod-name The name of the current pod (autodetected from container host name by default).	string	
quarkus.camel.cluster.kubernetes.jitter-factor The jitter factor to apply in order to prevent all pods to call Kubernetes APIs in the same instant (defaults to 1.2).	java.lang.Double	
quarkus.camel.cluster.kubernetes.lease-duration-millis The default duration of the lease for the current leader (defaults to 15000).	java.lang.Long	

Configuration property	Type	Default
quarkus.camel.cluster.kubernetes.renew-deadline-millis The deadline after which the leader must stop its services because it may have lost the leadership (defaults to 10000).	java.lang.Long	
quarkus.camel.cluster.kubernetes.retry-period-millis The time between two subsequent attempts to check and acquire the leadership. It is randomized using the jitter factor (defaults to 2000).	java.lang.Long	
quarkus.camel.cluster.kubernetes-order Service lookup order/priority (defaults to 2147482647).	java.lang.Integer	

Configuration property	Type	Default
quarkus.camel.cluster.kubernetes.resource-name The name of the lease resource used to do optimistic locking (defaults to 'leaders'). The resource name is used as prefix when the underlying Kubernetes resource can manage a single lock.	**string**	
quarkus.camel.cluster.kubernetes.lease-resource-type The lease resource type used in Kubernetes, either 'config-map' or 'lease' (defaults to 'lease').	**org.apache.camel.component.kubernetes.cluster.**	

Configuration property	Type	Default
	LoadBalancerResourceType	
quarkus.camel.cluster.kubernetes.rebalancing Whether the camel master namespace leaders should be distributed evenly across all the camel contexts in the cluster.	boolean	true
quarkus.camel.cluster.kubernetes-labels The labels key/value used to identify the pods composing the cluster, defaults to empty map.	Map<String, String>	

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

3.57. KUDU

Interact with Apache Kudu, a free and open source column-oriented data store of the Apache Hadoop ecosystem.

3.57.1. What's inside

- Kudu component, URI syntax: **kudu:host:port/tableName**

Refer to the above link for usage and configuration details.

3.57.2. 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-kudu</artifactId>
</dependency>
```

3.57.3. 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.

NOTE

Kudu is not supported for IBM Z and IBM Power.

3.58. LANGUAGE

Execute scripts in any of the languages supported by Camel.

3.58.1. What's inside

- Language component, URI syntax: **language:languageName:resourceUri**

Refer to the above link for usage and configuration details.

3.58.2. 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-language</artifactId>
</dependency>
```

3.58.3. Usage

3.58.3.1. Required Dependencies

The Language extension only handles the passing of an Exchange to a script for execution. The extension implementing the language must be added as a dependency. The following list of languages are implemented in Core:

- Constant

- ExchangeProperty
- File
- Header
- Ref
- Simple
- Tokenize

To use any other language, you must add the corresponding dependency. Consult the Languages Guide for details.

3.58.3.2. Native Mode

When loading scripts from the classpath in native mode, the path to the script file must be specified in the **quarkus.native.resources.includes** property of the **application.properties** file. For example:

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

3.58.4. allowContextMapAll option in native mode

The **allowContextMapAll** option is not supported in native mode as it requires reflective access to security sensitive camel core classes such as **CamelContext** & **Exchange**. This is considered a security risk and thus access to the feature is not provided by default.

3.59. LDAP

Perform searches on LDAP servers.

3.59.1. What's inside

- LDAP component, URI syntax: **ldap:dirContextName**

Refer to the above link for usage and configuration details.

3.59.2. 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-ldap</artifactId>
</dependency>
```

3.59.3. Usage

3.59.3.1. Using SSL in Native Mode

When using a custom **SSLSocketFactory** in native mode, such as the one in the Configuring SSL section, you need to register the class for reflection otherwise the class will not be made available on the classpath. Add the **@RegisterForReflection** annotation above the class definition, as follows:

```
@RegisterForReflection
public class CustomSSLSocketFactory extends SSLSocketFactory {
    // The class definition is the same as in the above link.
}
```

3.60. LRA

Camel saga binding for Long-Running-Action framework

3.60.1. What's inside

- LRA

Refer to the above link for usage and configuration details.

3.60.2. 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-lra</artifactId>
</dependency>
```

3.61. LOG

Log messages to the underlying logging mechanism.

3.61.1. What's inside

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

Refer to the above link for usage and configuration details.

3.61.2. 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>
```

3.62. MAIL

Send and receive emails using imap, pop3 and smtp protocols. Marshal Camel messages with attachments into MIME-Multipart messages and back.

3.62.1. What's inside

- IMAP component, URI syntax: **imap:host:port**
- IMAPS (Secure) component, URI syntax: **imaps:host:port**
- MIME Multipart data format
- POP3 component, URI syntax: **pop3:host:port**
- POP3S component, URI syntax: **pop3s:host:port**
- SMTP component, URI syntax: **smtp:host:port**
- SMTPS component, URI syntax: **smtps:host:port**

Refer to the above links for usage and configuration details.

3.62.2. 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-mail</artifactId>
</dependency>
```

3.63. MANAGEMENT

JMX management strategy and associated managed resources.

3.63.1. 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-management</artifactId>
</dependency>
```

3.63.2. Usage

For information on using Managed Beans in Camel, consult the JMX section of the Camel Manual .

3.63.2.1. Enabling and Disabling JMX

JMX can be enabled or disabled in Camel-Quarkus by any of the following methods:

1. Adding or removing the **camel-quarkus-management** extension.
2. Setting the **camel.main.jmxEnabled** configuration property to a boolean value.
3. Setting the system property **-Dorg.apache.camel.jmx.disabled** to a boolean value.

3.63.2.2. Native mode

Experimental JMX support was added for native executables in GraalVM for JDK 17/20 / Mandrel 23.0. You can enable this feature by adding the following configuration property to **application.properties**.

```
quarkus.native.monitoring=jmxserver
```

If you want the native application to be discoverable by tools such as JConsole and VisualVM, append the **jvmstat** option to the above mentioned configuration.

For more information, refer to the Quarkus native guide.

3.64. MAPSTRUCT

Type Conversion using Mapstruct

3.64.1. What's inside

- MapStruct component, URI syntax: **mapstruct:className**

Refer to the above link for usage and configuration details.

3.64.2. 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-mapstruct</artifactId>
</dependency>
```

3.64.3. Usage

3.64.3.1. Annotation Processor

To use MapStruct, you must configure your build to use an annotation processor.

3.64.3.1.1. Maven

```
<plugins>
    <plugin>
```

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <annotationProcessorPaths>
            <path>
              <groupId>org.mapstruct</groupId>
              <artifactId>mapstruct-processor</artifactId>
              <version>{mapstruct-version}</version>
            </path>
          </annotationProcessorPaths>
        </configuration>
      </plugin>
    </plugins>
```

3.64.3.1.2. Gradle

```
dependencies {
    annotationProcessor 'org.mapstruct:mapstruct-processor:{mapstruct-version}'
    testAnnotationProcessor 'org.mapstruct:mapstruct-processor:{mapstruct-version}'
}
```

3.64.3.2. Mapper definition discovery

By default, Red Hat build of Apache Camel for Quarkus will automatically discover the package paths of your **@Mapper** annotated interfaces or abstract classes and pass them to the Camel MapStruct component.

If you want finer control over the specific packages that are scanned, then you can set a configuration property in **application.properties**.

```
camel.component.mapstruct.mapper-package-name = com.first.package,org.second.package
```

3.65. MASTER

Have only a single consumer in a cluster consuming from a given endpoint; with automatic failover if the JVM dies.

3.65.1. What's inside

- Master component, URI syntax: **master:namespace:delegateUri**

Refer to the above link for usage and configuration details.

3.65.2. 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-master</artifactId>
</dependency>
```

3.65.3. Additional Camel Quarkus configuration

This extension can be used in conjunction with extensions below:

- Camel Quarkus File
- Camel Quarkus Kubernetes

3.66. MICROMETER

Collect various metrics directly from Camel routes using the Micrometer library.

3.66.1. What's inside

- Micrometer component, URI syntax: **micrometer:metricsType:metricsName**

Refer to the above link for usage and configuration details.

3.66.2. 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-micrometer</artifactId>
</dependency>
```

3.66.3. Usage

This extension leverages Quarkus Micrometer. Quarkus supports a variety of Micrometer metric registry implementations.

Your application should declare the following dependency or one of the dependencies listed in the quarkiverse documentation, depending on the monitoring solution you want to work with.

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

If no dependency is declared, the Micrometer extension creates a **SimpleMeterRegistry** instance, suitable mainly for testing.

3.66.4. Camel Quarkus limitations

3.66.4.1. Exposing Micrometer statistics in JMX

Exposing Micrometer statistics in JMX is not available in native mode as **quarkus-micrometer-registry-jmx** does not have native support at present.

3.66.4.2. Decrement header for Counter is ignored by Prometheus

Prometheus backend ignores negative values during increment of Counter metrics.

3.66.4.3. Exposing statistics in JMX

In Red Hat build of Apache Camel for Quarkus, registering a **JmxMeterRegistry** is simplified. Add a dependency for **io.quarkiverse.micrometer.registry:quarkus-micrometer-registry-jmx** and a **JmxMeterRegistry** will automatically get created for you.

3.66.5. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.metrics.enable-route-policy Set whether to enable the MicrometerRoutePolicyFactory for capturing metrics on route processing times.	boolean	**true**
quarkus.camel.metrics.enable-message-history Set whether to enable the MicrometerMessageHistoryFactory 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 MicrometerExchangeEventNotifier for capturing metrics on exchange processing times.	boolean	**true**
quarkus.camel.metrics.enable-route-event-notifier Set whether to enable the MicrometerRouteEventNotifier 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-instrumented-thread-pool-factory Set whether to gather performance information about Camel Thread Pools by injecting an InstrumentedThreadPoolFactory.	boolean	false

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

NOTE

If you are migrating to **micrometer** from **smallrye-metrics**, you may need to manually define some beans as scoped.

In **smallrye-metrics**, classes that are registered for metrics (for example with **@COUNTED**, **@METRIC**), but not registered as scoped beans, are registered automatically. This does not happen in **micrometer**.

In **micrometer** you need to manually register beans accessed via CDI, by for example adding a **@Dependent** annotation.

3.67. MICROPROFILE FAULT TOLERANCE

Circuit Breaker EIP using Microprofile Fault Tolerance

3.67.1. What's inside

- Microprofile Fault Tolerance

Refer to the above link for usage and configuration details.

3.67.2. 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-fault-tolerance</artifactId>
</dependency>
```

3.68. MICROPROFILE HEALTH

Expose Camel health checks via MicroProfile Health

3.68.1. What's inside

- Microprofile Health

Refer to the above link for usage and configuration details.

3.68.2. 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>
```

3.68.3. 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.

3.68.3.1. Provided health checks

Some checks are automatically registered for your application.

3.68.3.1.1. 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'.

3.68.3.1.2. 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'.

3.68.4. 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.

3.69. MINIO

Store and retrieve objects from Minio Storage Service using Minio SDK.

3.69.1. What's inside

- Minio component, URI syntax: **minio:bucketName**

Refer to the above link for usage and configuration details.

3.69.2. 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-minio</artifactId>
</dependency>
```

3.69.3. Additional Camel Quarkus configuration

Depending on Minio configuration, this extension may require SSL encryption on its connections. In such cases, you will need to add **quarkus.ssl.native=true** to your **application.properties**. See also Quarkus native SSL guide and Native mode section of Camel Quarkus user guide.

There are two different configuration approaches:

- Minio client can be defined via quarkus properties leveraging the Quarkiverse Minio (see documentation). Camel will autowire client into the Minio component. This configuration allows definition of only one minio client, therefore it isn't possible to define several different minio endpoints, which run together.

- Provide client/clients for camel registry (e.g. CDI producer/bean) and reference them from endpoint.

```
minio:foo?minioClient=#minioClient
```

3.70. MLLP

Communicate with external systems using the MLLP protocol.

3.70.1. What's inside

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

Refer to the above link for usage and configuration details.

3.70.2. 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>
```

3.70.3. Additional Camel Quarkus configuration

- Check the Character encodings section of the Native mode guide if you wish to use the **defaultCharset** component option.

3.71. MOCK

Test routes and mediation rules using mocks.

3.71.1. What's inside

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

Refer to the above link for usage and configuration details.

3.71.2. 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>
```

3.71.3. 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 jakarta.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 jakarta.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");
    }
}
```

3.71.4. 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).

3.72. MONGODB

Perform operations on MongoDB documents and collections.

3.72.1. What's inside

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

Refer to the above link for usage and configuration details.

3.72.2. 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>
```

3.72.3. 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.

3.73. MYBATIS

Performs a query, poll, insert, update or delete in a relational database using MyBatis.

3.73.1. What's inside

- MyBatis component, URI syntax: **mybatis:statement**
- MyBatis Bean component, URI syntax: **mybatis-bean:beanName:methodName**

Refer to the above links for usage and configuration details.

3.73.2. 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-mybatis</artifactId>
</dependency>
```

3.73.3. Additional Camel Quarkus configuration

Refer to Quarkus MyBatis for configuration. It must enable the following options.

```
quarkus.mybatis.xmlconfig.enable=true
quarkus.mybatis.xmlconfig.path=SqlMapConfig.xml
```

TIP

quarkus.mybatis.xmlconfig.path must be the same with **configurationUri** param in the mybatis endpoint.

3.74. NETTY HTTP

The Netty HTTP extension provides HTTP transport on top of the Netty extension.

3.74.1. What's inside

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

Refer to the above link for usage and configuration details.

3.74.2. 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-http</artifactId>
</dependency>
```

3.74.3. 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)
```

3.74.4. Additional Camel Quarkus configuration

- Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

3.75. NETTY

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

3.75.1. What's inside

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

Refer to the above link for usage and configuration details.

3.75.2. 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>
```

3.76. OPENAPI JAVA

Expose OpenAPI resources defined in Camel REST DSL

3.76.1. What's inside

- Openapi Java

Refer to the above link for usage and configuration details.

3.76.2. 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>
```

3.76.3. Usage

You can use this extension to expose REST DSL services to Quarkus OpenAPI. With **quarkus-smallrye-openapi**, you can access them by **/q/openapi?format=json**.

Refer to the Quarkus OpenAPI guide for further information.

This is an experimental feature. You can enable it by

```
quarkus.camel.openapi.expose.enabled=true
```

WARNING

It's the user's responsibility to use **@RegisterForReflection** to register all model classes for reflection.

It doesn't support the rest services used in **org.apache.camel.builder.LambdaRouteBuilder** right now. Also, it can not use CDI injection in the RouteBuilder **configure()** since we get the rest definitions at build time while CDI is unavailable.

3.77. OPENTELEMETRY

Distributed tracing using OpenTelemetry

3.77.1. What's inside

- OpenTelemetry

Refer to the above link for usage and configuration details.

3.77.2. 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-opentelemetry</artifactId>
</dependency>
```

3.77.3. Usage

The extension automatically creates a Camel **OpenTelemetryTracer** and binds it to the Camel registry.

In order to send the captured traces to a tracing system, you need to configure some properties within **application.properties** like those below.

```
# Identifier for the origin of spans created by the application
quarkus.application.name=my-camel-application

# OTLP exporter endpoint
quarkus.opentelemetry.tracer.exporter.otlp.endpoint=http://localhost:4317
```

Refer to the Quarkus OpenTelemetry guide for a full list of configuration options.

Route endpoints can be excluded from tracing by configuring a property named **quarkus.camel.opentelemetry.exclude-patterns** in **application.properties**. For example:

```
# Exclude all direct & netty-http endpoints from tracing
quarkus.camel.opentelemetry.exclude-patterns=direct:*,netty-http:*
```

3.77.3.1. Exporters

Quarkus OpenTelemetry defaults to the standard OTLP exporter defined in OpenTelemetry. Additional exporters will be available in the Quarkiverse quarkus-opentelemetry-exporter project.

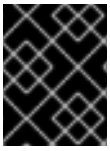
3.77.3.2. Tracing CDI bean method execution

When instrumenting the execution of CDI bean methods from Camel routes, you should annotate such methods with **io.opentelemetry.extension.annotations.WithSpan**. Methods annotated with **@WithSpan** will create a new Span and establish any required relationships with the current Trace context.

For example, to instrument a CDI bean from a Camel route, first ensure the appropriate methods are annotated with **@WithTrace**.

```
@ApplicationScoped
@Named("myBean")
public class MyBean {
    @WithSpan
    public String greet() {
        return "Hello World!";
    }
}
```

Next, use the bean in your Camel route.



IMPORTANT

To ensure that the sequence of recorded spans is correct, you must use the full **to("bean:")** endpoint URI and not the shortened **.bean()** EIP DSL method.

```
public class MyRoutes extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("direct:executeBean")
                .to("bean:myBean?method=greet");
    }
}
```

There is more information about CDI instrumentation in the Quarkus OpenTelemetry guide.

3.77.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.opentelemetry.encoding Sets whether header names need to be encoded. Can be useful in situations where OpenTelemetry propagators potentially set header name values in formats that are not compatible with the target system. E.g for JMS where the specification mandates header names are valid Java identifiers.	**boolean**	**false**

Configuration property	Type	Default
quarkus.camel.opentelemetry.exclude-patterns Sets whether to disable tracing for endpoint URIs that match the given comma separated patterns. The pattern can take the following forms: 1. An exact match on the endpoint URI. E.g platform-http:/some/path 2. A wildcard match. E.g platform-http:* 3. A regular expression matching the endpoint URI. E.g platform-http:/prefix/.*	string	

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

3.78. PAHO MQTT5

Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.

3.78.1. What's inside

- Paho MQTT 5 component, URI syntax: **paho-mqtt5:topic**

Refer to the above link for usage and configuration details.

3.78.2. 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-paho-mqtt5</artifactId>
</dependency>
```

3.79. PAHO

Communicate with MQTT message brokers using Eclipse Paho MQTT Client.

3.79.1. What's inside

- Paho component, URI syntax: **paho:topic**

Refer to the above link for usage and configuration details.

3.79.2. 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-paho</artifactId>
</dependency>
```

3.80. PLATFORM HTTP

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

It is built on top of the Eclipse Vert.x HTTP server provided by the **quarkus-vertx-http** extension.

3.80.1. What's inside

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

Refer to the above link for usage and configuration details.

3.80.2. 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>
```

3.80.3. Usage

3.80.3.1. 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}"));
```

3.80.3.2. Using platform-http via Camel REST DSL

To be able to use Camel REST DSL with the **platform-http** component, add **camel-quarkus-rest** 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")
    .to("direct:handleGetRequest");

  .post("/my-post-endpoint")
    .to("direct:handlePostRequest");
```

3.80.3.3. 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);
        }
      });
    }
  });
```

3.80.3.4. Securing platform-http endpoints

Quarkus provides a variety of security and authentication mechanisms which can be used to secure **platform-http** endpoints. Refer to the Quarkus Security documentation for further details.

Within a route, it is possible to obtain the authenticated user and its associated **SecurityIdentity** and **Principal**:

```
from("platform-http:/secure")
  .process(e -> {
    Message message = e.getMessage();
    QuarkusHttpUser user =
message.getHeader(VertxPlatformHttpConstants.AUTHENTICATED_USER,
QuarkusHttpUser.class);
    SecurityIdentity securityIdentity = user.getSecurityIdentity();
    Principal principal = securityIdentity.getPrincipal();
    // Do something useful with SecurityIdentity / Principal. E.g check user roles etc.
  });
```

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**.

3.80.3.5. Implementing a reverse proxy

Platform HTTP component can act as a reverse proxy, in that case **Exchange.HTTP_URI**, **Exchange.HTTP_HOST** headers are populated from the absolute URL received on the request line of the HTTP request.

Here's an example of a HTTP proxy that simply redirects the Exchange to the origin server.

```
from("platform-http:proxy")
    .toD("http://"
        + "${headers." + Exchange.HTTP_HOST + "}");
```

3.80.4. Additional Camel Quarkus configuration

3.80.4.1. 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.

3.80.4.2. Character encodings

Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

3.81. QUARTZ

Schedule sending of messages using the Quartz 2.x scheduler.

3.81.1. What's inside

- Quartz component, URI syntax: **quartz:groupName/triggerName**

Refer to the above link for usage and configuration details.

3.81.2. 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-quartz</artifactId>
</dependency>
```

3.81.3. Usage

3.81.3.1. Clustering

Support for Quartz clustering is provided by the Quarkus Quartz extension. The following steps outline how to configure Quarkus Quartz for use with Camel.

1. Enable Quartz clustered mode and configure a **DataSource** as a persistence Quartz job store. An example configuration is as follows.

```
# Quartz configuration
quarkus.quartz.clustered=true
quarkus.quartz.store-type=jdbc-cmt
quarkus.scheduler.start-mode=forced

# Datasource configuration
quarkus.datasource.db-kind=postgresql
quarkus.datasource.username=quarkus_test
quarkus.datasource.password=quarkus_test
quarkus.datasource.jdbc.url=jdbc:postgresql://localhost/quarkus_test

# Optional automatic creation of Quartz tables
quarkus.flyway.connect-retries=10
quarkus.flyway.table=flyway_quarkus_history
quarkus.flyway.migrate-at-start=true
quarkus.flyway.baseline-on-migrate=true
quarkus.flyway.baseline-version=1.0
quarkus.flyway.baseline-description=Quartz
```

2. Add the correct JDBC driver extension to your application that corresponds to the value of **quarkus.datasource.db-kind**. In the above example **postgresql** is used, therefore the following JDBC dependency would be required. Adjust as necessary for your needs. Agroal is also required for **DataSource** support.

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-jdbc-postgresql</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-agroal</artifactId>
</dependency>
```

3. Quarkus Flyway can automatically create the necessary Quartz database tables for you. Add **quarkus-flyway** to your application (optional).

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-flyway</artifactId>
</dependency>
```

   Also add a Quartz database creation script for your chosen database kind. The Quartz project provides ready made scripts that can be copied from here. Add the SQL script to **src/main/resources/db/migration/V1.0.0__QuarkusQuartz.sql**. Quarkus Flyway will detect it on startup and will proceed to create the Quartz database tables.

4. Configure the Camel Quartz component to use the Quarkus Quartz scheduler.

```
@Produces
@Singleton
@Named("quartz")
public QuartzComponent quartzComponent(Scheduler scheduler) {
    QuartzComponent component = new QuartzComponent();
    component.setScheduler(scheduler);
    return component;
}
```

Further customization of the Quartz scheduler can be done via various configuration properties. Refer to to the Quarkus Quartz Configuration guide for more information.

3.82. REF

Route messages to an endpoint looked up dynamically by name in the Camel Registry.

3.82.1. What's inside

- Ref component, URI syntax: **ref:name**

Refer to the above link for usage and configuration details.

3.82.2. 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-ref</artifactId>
</dependency>
```

3.82.3. Usage

CDI producer methods can be harnessed to bind endpoints to the Camel registry, so that they can be resolved using the **ref** URI scheme in Camel routes.

For example, to produce endpoint beans:

```
@ApplicationScoped
public class MyEndpointProducers {
    @Inject
    CamelContext context;

    @Singleton
    @Produces
    @Named("endpoint1")
    public Endpoint directStart() {
        return context.getEndpoint("direct:start");
    }
```

```
    @Singleton
    @Produces
    @Named("endpoint2")
    public Endpoint logEnd() {
        return context.getEndpoint("log:end");
    }
}
```

Use **ref:** to refer to the names of the CDI beans that were bound to the Camel registry:

```
public class MyRefRoutes extends RouteBuilder {
    @Override
    public void configure() {
        // direct:start -> log:end
        from("ref:endpoint1")
            .to("ref:endpoint2");
    }
}
```

3.83. REST OPENAPI

Configure REST producers based on an OpenAPI specification document delegating to a component implementing the RestProducerFactory interface.

3.83.1. What's inside

- REST OpenApi component, URI syntax: **rest-openapi:specificationUri#operationId**

Refer to the above link for usage and configuration details.

3.83.2. 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-openapi</artifactId>
</dependency>
```

3.83.3. Usage

3.83.3.1. Required Dependencies

A **RestProducerFactory** implementation must be available when using the rest-openapi extension. The currently known extensions are:

- camel-quarkus-http
- camel-quarkus-netty-http

Maven users will need to add one of these dependencies to their **pom.xml**, for example:

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

Depending on which mechanism is used to load the OpenApi specification, additional dependencies may be required. When using the **file** resource locator, the **org.apache.camel.quarkus:camel-quarkus-file** extension must be added as a project dependency. When using **ref** or **bean** to load the specification, not only must the **org.apache.camel.quarkus:camel-quarkus-bean** dependency be added, but the bean itself must be annotated with **@RegisterForReflection**.

When using the **classpath** resource locator with native code, the path to the OpenAPI specification must be specified in the **quarkus.native.resources.includes** property of the **application.properties** file. For example:

```
quarkus.native.resources.includes=openapi.json
```

3.84. REST

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

3.84.1. What's inside

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

Refer to the above links for usage and configuration details.

3.84.2. 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>
```

3.84.3. Additional Camel Quarkus configuration

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

3.84.3.1. 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}")
            .to("direct:greet")

            // The non-dashed path parameter works by default
            .get("/undashed/param/{myParam}")
            .to("direct:greet");

            from("direct:greet")
                .setBody(constant("Hello World"));
    }
}
```

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

3.84.3.2. 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");
        ...
    }
}
```

3.85. SALESFORCE

Communicate with Salesforce using Java DTOs.

3.85.1. What's inside

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

Refer to the above link for usage and configuration details.

3.85.2. 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>
```

3.85.3. Usage

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

Test content.

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>{camel-version}</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>
```

3.85.3.2. Native mode support for Pub / Sub API with POJO pubSubDeserializeType

When using the Camel Salesforce Pub / Sub API and **pubSubDeserializeType** is configured as **POJO**, you must register any classes configured on the **pubSubPojoClass** option for reflection.

For example, given the following route.

```
from("salesforce:pubSubSubscribe:/event/TestEvent__e?
pubSubDeserializeType=POJO&pubSubPojoClass=org.foo.TestEvent")
    .log("Received Salesforce POJO topic message: ${body}");
```

Class **org.foo.TestEvent** would need to be registered for reflection.

```
package org.foo;

import io.quarkus.runtime.annotations.RegisterForReflection;

@RegisterForReflection
public class TestEvent {
    // Getters / setters etc
}
```

Refer to the Native mode user guide for more information.

3.85.4. 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.

3.86. SAGA

Execute custom actions within a route using the Saga EIP.

3.86.1. What's inside

- Saga component, URI syntax: **saga:action**

Refer to the above link for usage and configuration details.

3.86.2. 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-saga</artifactId>
</dependency>
```

3.87. SAP

Provides SAP Camel Component

3.87.1. What's inside

The SAP extension 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.

See below for details.

3.87.2. Maven coordinates

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

3.87.2.1. SAP Extension Camel Quarkus limitations

The SAP extension does not support the packaging type **uber-jar**, which causes the application to throw a runtime exception similar to this:

```
Caused by: java.lang.ExceptionInInitializerError: JCo initialization failed with
java.lang.ExceptionInInitializerError: Illegal JCo archive "sap-1.0.0-SNAPSHOT-runner.jar". It is not
allowed to rename or repackage the original archive "sapjco3.jar".
```

3.87.2.2. 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.

3.87.2.3. 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 3.1. 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**

NOTE

In order to pull the correct SAP libraries, you must include the following dependencies in your **pom.xml**

```
<dependency>
    <groupId>com.sap.conn.jco</groupId>
    <artifactId>sapjco3</artifactId>
    <version>3.1.4</version>
    <scope>system</scope>
    <systemPath>${basedir}/lib/sapjco3.jar</systemPath>
  </dependency>
  <dependency>
    <groupId>com.sap.conn.idoc</groupId>
    <artifactId>sapidoc3</artifactId>
    <version>3.1.1</version>
    <scope>system</scope>
    <systemPath>${basedir}/lib/sapidoc3.jar</systemPath>
  </dependency>
```

3.87.3. 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.

3.87.3.1. 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

3.87.3.2. 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.

3.87.3.3. 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.

3.87.3.4. 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.

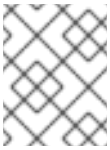
NOTE

The sRFC protocol used by this component delivers requests and responses to and from an SAP system, using **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

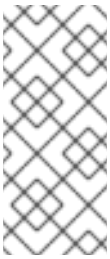
Camel SAP Transactional Remote Function Call Destination Camel component. This endpoint should be used in cases where requests must be delivered to the receiving SAP system **at most once**. To

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.

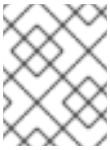


NOTE

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.

3.87.3.5. 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.

3.87.3.6. 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.

3.87.3.7. 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.

3.87.3.8. 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*.

3.87.3.9. 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.

3.87.4. 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. The *repository data store* must be configured directly on the relevant SAP component.

3.87.4.1. Configuration Overview

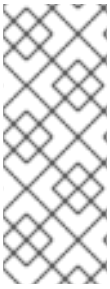
The SAP component maintains three maps to store destination data, server data, and repository data. The component's property, **destinationDataStore**, stores destination data keyed by destination name, the property, **serverDataStore**, stores server data keyed by server name and the property, **repositoryDataStore**, 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. The **sap-configuration** bean (of type **SapConnectionConfiguration**) will be automatically injected into any SAP component that is used in this application.

```
public class SAPRouteBuilder extends RouteBuilder {
 @BindToRegistry("sap-configuration")
 public SapConnectionConfiguration sapConfiguration() {
  SapConnectionConfiguration configuration = new SapConnectionConfiguration();
  configuration.setDestinationDataStore(destinationData());
  configuration.setServerDataStore(serverData());
  return configuration;
 }

 /**
  * Configures an Inbound SAP Connection
  * Please enter the connection property values for your environment
  */
 private Map<String, ServerData> serverData() {
  ServerData data = new ServerDataImpl();
  data.setGwhost("example.com");
  data.setGwserv("3300");
  data.setProgid("QUICKSTART");
  data.setRepositoryDestination("quickstartDest");
  data.setConnectionCount("2");
  return Map.of("quickstartServer", data);
 }

 /**
  * Configures an Outbound SAP Connection
  * Please enter the connection property values for your environment
  */
 private Map<String, DestinationData> destinationData() {
  DestinationData data = new DestinationDataImpl();
  data.setAshost("example.com");
  data.setSysnr("00");
  data.setClient("000");
  data.setUser("username");
  data.setPasswd("password");
  data.setLang("en");
  return Map.of("quickstartDest", data);
 }

 @Override
 public void configure() throws Exception {
```

```
    // Routes definitions
  }
}
```



NOTE

The values can be supplied from the **application.properties** file. In that case, you can use the property name instead of the hardcoded value.

For example:

ConfigProvider.getConfig().getValue("<property name>", String.class)

3.87.4.2. 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 code shows how to configure a sample destination with the name, **quickstartDest**:

```
    @BindToRegistry("sap-configuration")
    public SapConnectionConfiguration sapConfiguration() {
        SapConnectionConfiguration configuration = new SapConnectionConfiguration();
        configuration.setDestinationDataStore(destinationData());
        return configuration;
    }

    private Map<String, DestinationData> destinationData() {
        DestinationData data = new DestinationDataImpl();
        data.setAshost("example.com");
        data.setSysnr("00");
        data.setClient("000");
        data.setUser("username");
        data.setPasswd("password");
        data.setLang("en");
        return Map.of("quickstartDest", data);
    }

    @Override
    public void configure() throws Exception {
        ((DefaultCamelContext) getCamelContext()).addInterceptStrategy(new
CurrentProcessorDefinitionInterceptStrategy());
        // Routes definitions
    }
```

After configuring the destination as shown above, you can invoke the **BAPI_FLCUST_GETLIST** remote function call on the **quickstartDest** destination with the following URI:

```
sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST
```

3.87.4.2.1. 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.

3.87.4.2.2. 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**).

3.87.4.2.3. 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**.

3.87.4.2.4. 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.

3.87.4.2.5. 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.

3.87.4.2.6. 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.

3.87.4.2.7. 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]**.

3.87.4.3. 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 code shows how to create a sample server configuration with the name, **quickstartServer**.

```
@BindToRegistry("sap-configuration")
public SapConnectionConfiguration sapConfiguration() {
 SapConnectionConfiguration configuration = new SapConnectionConfiguration();
 configuration.setDestinationDataStore(destinationData());
 configuration.setServerDataStore(serverData());
 return configuration;
}

/**
 * Configures an Inbound SAP Connection
 * Please enter the connection property values for your environment
 */
private Map<String, ServerData> serverData() {
 ServerData data = new ServerDataImpl();
 data.setGwhost("example.com");
 data.setGwserv("3300");
 data.setProgid("QUICKSTART");
 data.setRepositoryDestination("quickstartDest");
 data.setConnectionCount("2");
 return Map.of("quickstartServer", data);
}

/**
 * Configures an Outbound SAP Connection
 * Please enter the connection property values for your environment
 */
private Map<String, DestinationData> destinationData() {
 DestinationData data = new DestinationDataImpl();
 data.setAshost("example.com");
 data.setSysnr("00");
 data.setClient("000");
 data.setUser("username");
 data.setPasswd("password");
 data.setLang("en");
 return Map.of("quickstartDest", data);
}
```

NOTE

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.

After configuring the destination as shown above, you can handle the **BAPI_FLCUST_GETLIST** remote function call on the **quickstartDest** remote function call from an invoking client, using the following URI:

```
sap-srfc-server:quickstartServer:BAPI_FLCUST_GETLIST
```

3.87.4.3.1. 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.

3.87.4.3.2. 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.

3.87.4.3.3. 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**.

3.87.4.4. 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:

```
@BindToRegistry("sap-configuration")
public SapConnectionConfiguration sapConfiguration() {
    SapConnectionConfiguration configuration = new SapConnectionConfiguration();
    configuration.setRepositoryDataStore(repositoryData());
    return configuration;
}

private Map<String, RepositoryData> repositoryData() {
    RepositoryData data = new RepositoryDataImpl();
    FunctionTemplate bookFlightFunctionTemplate = new FunctionTemplateImpl();
    data.setFunctionTemplates(Map.of("BOOK_FLIGHT", bookFlightFunctionTemplate));
    return Map.of("nplServer", data);
}
```

3.87.4.4.1. 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:

```
FunctionTemplate bookFlightFunctionTemplate = new FunctionTemplateImpl();

List<ListFieldMetaData> metaDataList = new ArrayList<>();
ListFieldMetaData metaData = new ListFieldMetaDataImpl();

// configure values
metaData.setName("example");
metaDataList.add(metaData);
```

```
bookFlightFunctionTemplate.setImportParameterList(metaDataList);

// in the same way you can configure other parameters
bookFlightFunctionTemplate.setExportParameterList(...);
bookFlightFunctionTemplate.setChangingParameterList(...);
bookFlightFunctionTemplate.setExceptionList(...);
bookFlightFunctionTemplate.setTableParameterList(...);
```

3.87.4.4.2. 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**:

```
ListFieldMetaData metaData = new ListFieldMetaDataImpl();
metaData.setName("TICKET_PRICE");
metaData.setType(DataType.BCD);
metaData.setByteLength(12);
metaData.setUnicodeByteLength(24);
metaData.setDecimals(2);
metaData.setOptional(true);
```

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:

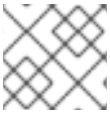
```
ListFieldMetaData metaData = new ListFieldMetaDataImpl();
metaData.setName("CONNINFO");
metaData.setType(DataType.TABLE);
RecordMetaData connectionInfo = new RecordMetaDataImpl();
metaData.setRecordMetaData(connectionInfo);
```

3.87.4.4.3. 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:

```
RecordMetaData connectionInfo = new RecordMetaDataImpl();
connectionInfo.setName("CONNECTION_INFO");
connectionInfo.setRecordFieldMetaData(...);
```

3.87.4.4.4. 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.

```
FieldMetaData fieldMetaData = new FieldMetaDataImpl();
fieldMetaData.setName("FLTINFO");
fieldMetaData.setType(DataType.STRUCTURE);
fieldMetaData.setByteOffset(0);
fieldMetaData.setUnicodeByteOffset(0);
RecordMetaData flightInfo = new RecordMetaDataImpl();
fieldMetaData.setRecordMetaData(flightInfo);
```

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.

```
FieldMetaData fieldMetaData = new FieldMetaDataImpl();
fieldMetaData.setName("FLTINFO");
fieldMetaData.setType(DataType.STRUCTURE);
fieldMetaData.setByteOffset(0);
fieldMetaData.setUnicodeByteOffset(0);
RecordMetaData flightInfo = new RecordMetaDataImpl();
fieldMetaData.setRecordMetaData(flightInfo);
```

3.87.5. 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.

3.87.6. 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.

3.87.7. Message Body for RFC

3.87.7.1. 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;
    ...
}
```

3.87.7.2. 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);

}
```

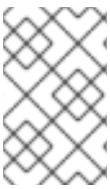
The field values in a structure object are accessed through the field's getter methods in the map interface. In addition, the structure interface provides a type-restricted method to retrieve field values.

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. 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.

3.87.7.3. 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.

3.87.7.3.1. 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

3.87.7.3.2. 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.

3.87.7.3.3. 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**.

3.87.7.3.4. 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.

3.87.7.3.5. 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.

3.87.7.3.6. 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

3.87.7.3.7. 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**.

3.87.7.3.8. 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**.

3.87.7.3.9. 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. 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.

3.87.8. Message Body for IDoc

3.87.8.1. 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
```

3.87.8.2. 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 getApplicationRrelease();

    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&gt; {
    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();
```

3.87.8.3. 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 3.1. 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");
```

3.87.9. Document attributes

IDoc Document Attributes table shows the control record attributes that you can set on the **Document** object.

Table 3.2. 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

Attribute	Length	SAP Field	Description
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
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)

Attribute	Length	SAP Field	Description
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)
serialization	20	**SERIAL**	EDI/ALE: Serialization field
status	2	**STATUS**	Status of IDoc
testFlag	1	**TEST**	Test flag

3.87.9.1. 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");
```

3.87.9.2. 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>
```

3.87.10. Transaction Support

3.87.10.1. 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.

3.87.10.2. 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 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.

3.87.10.3. 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.

3.87.10.4. 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 destination 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.

3.87.10.5. 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.

3.87.11. 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.

3.87.11.1. 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.

3.87.11.2. 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>
```

3.87.11.3. 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.

3.87.11.4. 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.

3.87.11.5. 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"/>
```

3.87.11.6. 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"
```

3.87.12. 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.

3.87.12.1. 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///

3.87.12.2. 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">
```

3.87.12.3. 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 3.2. 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>
```

3.87.13. 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.

3.87.13.1. 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");
```

3.87.13.2. 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>
```

3.87.13.3. **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);
}
```

3.87.13.4. 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);

    }
```

3.87.14. 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.

3.87.14.1. 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");
```

3.87.14.2. 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>
```

3.87.14.3. 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.

3.87.14.4. 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.

3.87.15. 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.

3.87.15.1. 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);
```

3.87.15.2. 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>
```

3.87.15.3. BookFlightRequest bean

The following listing illustrates a JAXB bean which unmarshals from the serialized form of an SAP **BOOK_FLIGHT** request object:

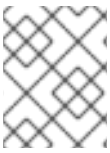
```
@XmlRootElement(name="Request",
namespace="http://sap.fusesource.org/rfc/nplServer/BOOK_FLIGHT")
@XmlAccessorType(XmlAccessType.FIELD)
public class BookFlightRequest {

    @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;
```

```
    ...
}
```

3.87.15.4. 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.

3.87.15.5. 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;

    ...
}
```

3.87.15.6. 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;

    ...
}
```

3.87.15.7. 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;

    ...
}
```

3.88. XQUERY

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

3.88.1. What's inside

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

Refer to the above links for usage and configuration details.

3.88.2. 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>
```

3.88.3. 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
```

3.89. SCHEDULER

Generate messages in specified intervals using java.util.concurrent.ScheduledExecutorService.

3.89.1. What's inside

- Scheduler component, URI syntax: **scheduler:name**

Refer to the above link for usage and configuration details.

3.89.2. 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-scheduler</artifactId>
</dependency>
```

3.90. SEDA

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

3.90.1. What's inside

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

Refer to the above link for usage and configuration details.

3.90.2. 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>
```

3.91. SERVLET

Serve HTTP requests by a Servlet.

3.91.1. What's inside

- Servlet component, URI syntax: **servlet:contextPath**

Refer to the above link for usage and configuration details.

3.91.2. 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-servlet</artifactId>
</dependency>
```

3.91.3. 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)
```

3.91.4. Additional Camel Quarkus configuration

Configuration property	Type	Default
quarkus.camel.servlet.url-patterns A comma separated list of path patterns under which the CamelServlet should be accessible. Example path patterns: /*, /services/*	string	
quarkus.camel.servlet.servlet-class A fully qualified name of a servlet class to serve paths that match url-patterns	string	org.apache.camel.component.servlet.CamelHttpTransportServlet
quarkus.camel.servlet.servlet-name A servletName as it would be defined in a web.xml file or in the jakarta.servlet.annotation.WebServlet#name() annotation.	string	CamelServlet
quarkus.camel.servlet."named-servlets".url-patterns A comma separated list of path patterns under which the CamelServlet should be accessible. Example path patterns: /*, /services/*	string	
quarkus.camel.servlet."named-servlets".servlet-class A fully qualified name of a servlet class to serve paths that match url-patterns	string	org.apache.camel.component.servlet.CamelHttpTransportServlet
quarkus.camel.servlet."named-servlets".servlet-name A servletName as it would be defined in a web.xml file or in the jakarta.servlet.annotation.WebServlet#name() annotation.	string	CamelServlet

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

3.92. SLACK

Send and receive messages to/from Slack.

3.92.1. What's inside

- Slack component, URI syntax: **slack:channel**

Refer to the above link for usage and configuration details.

3.92.2. 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-slack</artifactId>
</dependency>
```

3.92.3. 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.

3.93. SNMP

Receive traps and poll SNMP (Simple Network Management Protocol) capable devices.

3.93.1. What's inside

- SNMP component, URI syntax: **snmp:host:port**

Refer to the above link for usage and configuration details.

3.93.2. Maven coordinates

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

3.94. SOAP DATAFORMAT

Marshal Java objects to SOAP messages and back.

3.94.1. What's inside

- SOAP data format

Refer to the above link for usage and configuration details.

3.94.2. 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>
```

3.95. SPLUNK

Publish or search for events in Splunk.

3.95.1. What's inside

- Splunk component, URI syntax: **splunk:name**

Refer to the above link for usage and configuration details.

3.95.2. 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-splunk</artifactId>
</dependency>
```

3.95.3. 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.

3.96. SPLUNK HEC

The splunk component allows to publish events in Splunk using the HTTP Event Collector.

3.96.1. What's inside

- Splunk HEC component, URI syntax: **splunk-hec:splunkURL/token**

Refer to the above link for usage and configuration details.

3.96.2. Maven coordinates

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

3.97. SQL

Perform SQL queries.

3.97.1. What's inside

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

Refer to the above links for usage and configuration details.

3.97.2. 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>
```

3.97.3. Additional Camel Quarkus configuration

3.97.3.1. 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.

3.97.3.1.1. 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.

3.97.3.2. 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
```

3.97.3.3. SQL aggregation repository in native mode

In order to use SQL aggregation repositories like **JdbcAggregationRepository** in native mode, you must enable native serialization support.

In addition, if your exchange bodies are custom types, they must be registered for serialization by annotating their class declaration with **@RegisterForReflection(serialization = true)**.

3.98. TELEGRAM

Send and receive messages acting as a Telegram Bot Telegram Bot API.

3.98.1. What's inside

- Telegram component, URI syntax: **telegram:type**

Refer to the above link for usage and configuration details.

3.98.2. 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-telegram</artifactId>
</dependency>
```

3.98.3. Usage

3.98.4. Webhook Mode

The Telegram extension supports usage in the webhook mode.

In order to enable webhook mode, you must add a REST implementation to your application. Maven users, for example, can add the **camel-quarkus-rest** extension to their **pom.xml** file:

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

3.98.4.1. Webhook

IMPORTANT

In this release of Red Hat build of Apache Camel for Quarkus, webhook mode is not supported.

3.98.5. 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.

3.99. TIMER

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

3.99.1. What's inside

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

Refer to the above link for usage and configuration details.

3.99.2. 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>
```

3.100. VALIDATOR

Validate the payload using XML Schema and JAXP Validation.

3.100.1. What's inside

- Validator component, URI syntax: **validator:resourceUri**

Refer to the above link for usage and configuration details.

3.100.2. 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-validator</artifactId>
</dependency>
```

3.101. VELOCITY

Transform messages using a Velocity template.

3.101.1. What's inside

- Velocity component, URI syntax: **velocity:resourceUri**

Refer to the above link for usage and configuration details.

3.101.2. 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-velocity</artifactId>
</dependency>
```

3.101.3. Usage

3.101.3.1. Custom body as domain object in the native mode

When using a custom object as message body and referencing its properties in the template in the native mode, all the classes need to be registered for reflection (see the documentation).

Example:

```
@RegisterForReflection
public interface CustomBody {
}
```

3.101.4. allowContextMapAll option in native mode

The **allowContextMapAll** option is not supported in native mode as it requires reflective access to security sensitive camel core classes such as **CamelContext** & **Exchange**. This is considered a security risk and thus access to the feature is not provided by default.

3.101.5. Additional Camel Quarkus configuration

This component typically loads Velocity templates from classpath. To make it work also in native mode, you need to explicitly embed the templates in the native executable by using the **quarkus.native.resources.includes** property.

For instance, the route below would load the Velocity template from a classpath resource named **template/simple.vm**:

```
from("direct:start").to("velocity://template/simple.vm");
```

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

```
quarkus.native.resources.includes = template/*.vm
```

3.102. VERT.X HTTP CLIENT

Camel HTTP client support with Vert.x

3.102.1. What's inside

- Vert.x HTTP Client component, URI syntax: **vertx-http:httpUri**

Refer to the above link for usage and configuration details.

3.102.2. 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-vertx-http</artifactId>
</dependency>
```

3.102.3. 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)
```

3.102.4. Additional Camel Quarkus configuration

3.102.5. allowJavaSerializedObject option in native mode

When using the **allowJavaSerializedObject** option in native mode, the support of serialization might need to be enabled. Please, refer to the native mode user guide for more information.

3.102.5.1. Character encodings

Check the Character encodings section of the Native mode guide if the application is expected to send and receive requests using non-default encodings.

3.103. VERT.X WEBSOCKET

This extension enables you to create WebSocket endpoints to that act as either a WebSocket server, or as a client to connect an existing WebSocket .

It is built on top of the Eclipse Vert.x HTTP server provided by the **quarkus-vertx-http** extension.

3.103.1. What's inside

- Vert.x WebSocket component, URI syntax: **vertx-websocket:host:port/path**

Refer to the above link for usage and configuration details.

3.103.2. 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-vertx-websocket</artifactId>
</dependency>
```

3.103.3. Usage

3.103.3.1. Vert.x WebSocket consumers

When you create a Vert.x WebSocket consumer (E.g with **from("vertx-websocket")**), the host and port configuration in the URI are redundant since the WebSocket will always be hosted on the Quarkus HTTP server.

The configuration of the consumer can be simplified to only include the resource path of the WebSocket. For example.

```
from("vertx-websocket:/my-websocket-path")
  .setBody().constant("Hello World");
```

NOTE

While you do not need to explicitly configure the host/port on the vertx-websocket consumer. If you choose to, the host & port must exactly match the value of the Quarkus HTTP server configuration values for **quarkus.http.host** and **quarkus.http.port**. Otherwise an exception will be thrown at runtime.

3.103.3.2. Vert.x WebSocket producers

Similar to above, if you want to produce messages to the internal Vert.x WebSocket consumer, then you can omit the host and port from the endpoint URI.

```
from("vertx-websocket:/my-websocket-path")
  .log("Got body: ${body}");

from("direct:sendToWebSocket")
  .log("vertx-websocket:/my-websocket-path");
```

Or alternatively, you can refer to the full host & port configuration for the Quarkus HTTP server.

```
from("direct:sendToWebSocket")
  .log("vertx-websocket:{{quarkus.http.host}}:{{quarkus.http.port}}/my-websocket-path");
```

When producing messages to an external WebSocket server, then you must always provide the host name and port (if required).

3.103.4. Additional Camel Quarkus configuration

3.103.4.1. Vert.x WebSocket server configuration

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

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

3.103.4.2. Character encodings

Check the Character encodings section of the Native mode guide if you expect your application to send or receive requests using non-default encodings.

3.104. XJ

Transform JSON and XML message using a XSLT.

3.104.1. What's inside

- XJ component, URI syntax: **xj:resourceUri**

Refer to the above link for usage and configuration details.

3.104.2. 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-xj</artifactId>
</dependency>
```

3.105. XML IO DSL

An XML stack for parsing XML route definitions

3.105.1. What's inside

- XML DSL

Refer to the above link for usage and configuration details.

3.105.2. 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-xml-io-dsl</artifactId>
```

```
</dependency>
```

3.105.3. Additional Camel Quarkus configuration

3.105.3.1. XML file encodings

By default, some XML file encodings may not work out of the box in native mode. Please, check the Character encodings section to learn how to fix.

3.106. XML JAXP

XML JAXP type converters and parsers

3.106.1. 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-xml-jaxp</artifactId>
</dependency>
```

3.107. XPATH

Evaluates an XPath expression against an XML payload

3.107.1. What's inside

- XPath language

Refer to the above link for usage and configuration details.

3.107.2. 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>
```

3.107.3. 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
```

3.108. XSLT SAXON

Transform XML payloads using an XSLT template using Saxon.

3.108.1. What's inside

- XSLT Saxon component, URI syntax: **xslt-saxon:resourceUri**

Refer to the above link for usage and configuration details.

3.108.2. 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-xslt-saxon</artifactId>
</dependency>
```

3.109. XSLT

Transforms XML payload using an XSLT template.

3.109.1. What's inside

- XSLT component, URI syntax: **xslt:resourceUri**

Refer to the above link for usage and configuration details.

3.109.2. 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-xslt</artifactId>
</dependency>
```

3.109.3. Additional Camel Quarkus configuration

To optimize XSLT processing, the extension needs to know the locations of the XSLT templates at build time. The XSLT source URIs have to be passed via the **quarkus.camel.xslt.sources** property. Multiple URIs can be separated by comma.

```
quarkus.camel.xslt.sources = transform.xsl, classpath:path/to/my/file.xsl
```

Scheme-less URIs are interpreted as **classpath:** URIs.

Only **classpath:** URIs are supported on Quarkus native mode. **file:**, **http:** and other kinds of URIs can be used on JVM mode only.

<xsl:include> and **<xsl:messaging>** XSLT elements are also supported in JVM mode only right now.

If **aggregate** DSL is used, **XsltSaxonAggregationStrategy** has to be used such as

```
from("file:src/test/resources?noop=true&sortBy=file:name&antInclude=*.xml")
  .routeId("aggregate").noAutoStartup()
  .aggregate(new XsltSaxonAggregationStrategy("xslt/aggregate.xsl"))
  .constant(true)
  .completionFromBatchConsumer()
  .log("after aggregate body: ${body}")
  .to("mock:transformed");
```

Also, it's only supported on JVM mode.

3.109.3.1. Configuration

TransformerFactory features can be configured using following property:

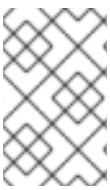
```
quarkus.camel.xslt.features."http\://javax.xml.XMLConstants/feature/secure-processing"=false
```

3.109.3.2. Extension functions support

Xalan's extension functions do work properly only when:

1. Secure-processing is disabled
2. Functions are defined in a separate jar
3. Functions are augmented during native build phase. For example, they can be registered for reflection:

```
@RegisterForReflection(targets = { my.Functions.class })
public class FunctionsConfiguration {
}
```



NOTE

The content of the XSLT source URIs is parsed and compiled into Java classes at build time. These Java classes are the only source of XSLT information at runtime. The XSLT source files may not be included in the application archive at all.

Configuration property	Type	Default
quarkus.camel.xslt.sources A comma separated list of templates to compile.	string	
quarkus.camel.xslt.package-name The package name for the generated classes.	string	org.apache.camel.quarkus.component.xslt.generated
quarkus.camel.xslt.features TransformerFactory features.	Map<String, Boolean>	

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

3.110. YAML DSL

An YAML stack for parsing YAML route definitions

3.110.1. What's inside

- YAML DSL

Refer to the above link for usage and configuration details.

3.110.2. 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-yaml-dsl</artifactId>
</dependency>
```

3.110.3. Usage

3.110.3.1. Native mode

The following constructs when defined within Camel YAML DSL markup, require you to register classes for reflection. Refer to the Native mode guide for details.

3.110.3.1.1. Bean definitions

The YAML DSL provides the capability to define beans as follows.

```
- beans:
    - name: "greetingBean"
      type: "org.acme.GreetingBean"
      properties:
        greeting: "Hello World!"
- route:
    id: "my-yaml-route"
    from:
      uri: "timer:from-yaml?period=1000"
      steps:
        - to: "bean:greetingBean"
```

In this example, the **GreetingBean** class needs to be registered for reflection. This applies to any types that you refer to under the **beans** key in your YAML routes.

```
@RegisterForReflection
public class GreetingBean {
}
```

3.110.3.1.2. Exception handling

Camel provides various methods of handling exceptions. Some of these require that any exception classes referenced in their DSL definitions are registered for reflection.

on-exception

```
- on-exception:
    handled:
      constant: "true"
    exception:
      - "org.acme.MyHandledException"
    steps:
      - transform:
          constant: "Sorry something went wrong"
```

```
@RegisterForReflection
public class MyHandledException {
}
```

throw-exception

```
- route:
    id: "my-yaml-route"
    from:
      uri: "direct:start"
      steps:
        - choice:
            when:
              - simple: "${body} == 'bad value'"
                steps:
                  - throw-exception:
                      exception-type: "org.acme.ForcedException"
                      message: "Forced exception"
            otherwise:
              steps:
                - to: "log:end"
```

```
@RegisterForReflection
public class ForcedException {
}
```

do-catch

```
- route:
    id: "my-yaml-route2"
    from:
      uri: "direct:tryCatch"
      steps:
        - do-try:
            steps:
              - to: "direct:readFile"
            do-catch:
              - exception:
                  - "java.io.FileNotFoundException"
                steps:
                  - transform:
                      constant: "do-catch caught an exception"
```

```
@RegisterForReflection(targets = FileNotFoundException.class)
public class MyClass {
}
```

3.111. ZIP DEFLATE COMPRESSION

Compress and decompress streams using java.util.zip.Deflater, java.util.zip.Inflater or java.util.zip.GZIPStream.

3.111.1. What's inside

- GZip Deflater data format
- Zip Deflater data format

Refer to the above links for usage and configuration details.

3.111.2. 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-zip-deflater</artifactId>
</dependency>
```

3.112. ZIP FILE

Compression and decompress streams using java.util.zip.ZipStream.

3.112.1. What's inside

- Zip File data format

Refer to the above link for usage and configuration details.

3.112.2. 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-zipfile</artifactId>
</dependency>
```

CHAPTER 4. QUARKUS CXF OVERVIEW

This chapter provides information about Quarkus CXF extensions, CXF modules and CXF annotations supported by Quarkus CXF.

4.1. CXF EXTENSIONS

This chapter provides an overview of the CXF extensions. The Support Matrix shows information about the extensions, their support levels, and supported standards for each extension.

There are 8 extensions.

Table 4.1. Quarkus CXF Support Matrix

Quarkus CXF extension	Support level	Since	Supported standards
Quarkus CXF **quarkus-cxf**	Production support	0.1.0	JAX-WS, JAXB, WS-Addressing, WS-Policy, MTOM
Quarkus CXF Metrics Feature **quarkus-cxf-rt-features-metrics**	Production support	0.14.4	
Quarkus CXF OpenTelemetry **quarkus-cxf-integration-tracing-opentelemetry**	Production support	2.7.0	
Quarkus CXF WS-Security **quarkus-cxf-rt-ws-security**	Production support	0.14.4	WS-Security, WS-SecurityPolicy
Quarkus CXF WS-ReliableMessaging **quarkus-cxf-rt-ws-rm**	Production support	1.5.3	WS-ReliableMessaging
Quarkus CXF Security Token Service (STS) **quarkus-cxf-services-sts**	Production support	1.5.3	WS-Trust
Quarkus CXF HTTP Async Transport **quarkus-cxf-rt-transports-http-hc5**	Production support	1.1.0	

Quarkus CXF extension	Support level	Since	Supported standards
Quarkus CXF XJC Plugins **quarkus-cxf-xjc-plugins**	Production support	1.5.11	

4.2. SUPPORTED CXF MODULES

The following is a list of CXF modules supported by Quarkus CXF.

NOTE

Do not use these as direct dependcies. Instead, use one of the Section 4.1, "CXF extensions" that provide the CXF module you want as a transitive dependency.

4.2.1. Front ends

Out of CXF front ends only the JAX-WS front end is fully supported by **quarkus-cxf**.

4.2.2. Data Bindings

Out of CXF Data Bindings only the following ones are supported:

- JAXB
- MTOM Attachments with JAXB

4.2.3. Transports

Out of CXF Transports only the following ones are supported:

- **quarkus-cxf** implements its own custom transport based on Quarkus and Vert.x for serving SOAP endpoints
- HTTP client via **quarkus-cxf**, including
  - Basic Authentication
- Asynchronous Client HTTP Transport via **quarkus-cxf-rt-transports-http-hc5**

4.2.4. Tools

- **wsdl2Java** - see the Generate the Model classes from WSDL section of User guide
- **java2ws** - see the Generate WSDL from Java section of User guide

4.2.5. Supported SOAP Bindings

All CXF WSDL Bindings are supported. In order to switch to SOAP 1.2 or to add MTOM, set **quarkus.cxf.[client|endpoint]."name".soap-binding** to one of the following values:

Binding	Property Value
SOAP 1.1 (default)	**http://schemas.xmlsoap.org/wsdl/soap/http**
SOAP 1.2	**http://www.w3.org/2003/05/soap/bindings/HTTP/**
SOAP 1.1 with MTOM	**http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true**
SOAP 1.2 with MTOM	**http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true**

4.3. UNSUPPORTED CXF MODULES

The following is a list of CXF modules currently not supported by Quarkus CXF, with suggested alternatives.

CXF module	Alternative
JAX-RS **cxf-rt-frontend-jaxrs** **cxf-rt-rs-client**	Use Quarkus RESTEasy
Fediz	Use Quarkus OpenID Connect
Aegis	Use JAXB and JAX-WS
DOSGI Karaf	
JiBX	Use JAXB and JAX-WS
Local transport **cxf-rt-transports-local**	Use HTTP transport
JMS transport **cxf-rt-transports-jms**	Use HTTP transport
JBI **cxf-rt-transports-jbi** **cxf-rt-bindings-jbi**	Deprecated in CXF use HTTP transport
UDP transport **cxf-rt-transports-udp**	Use HTTP transport

CXF module	Alternative
Coloc transport	Use HTTP transport
WebSocket transport **cxf-rt-transports-websocket**	Use HTTP transport
Clustering **cxf-rt-features-clustering**	Planned
CORBA **cxf-rt-bindings-corba**	Use JAX-WS
SDO databinding **cxf-rt-databinding-sdo**	
XMLBeans	Deprecated in CXF
Javascript frontend	Use JAX-WS
JCA transport	Use HTTP transport
WS-Transfer runtime **cxf-rt-ws-transfer**	
Throttling **cxf-rt-features-throttling**	Use load balancer

4.4. SUPPORTED CXF ANNOTATIONS

The following list shows the status for CXF annotations on Quarkus. Unless stated otherwise, the support is available via **io.quarkiverse.cxf:quarkus-cxf**.

Annotation	Status
@org.apache.cxf.feature.Features	Supported
@org.apache.cxf.interceptor.InInterceptors	Supported
@org.apache.cxf.interceptor.OutInterceptors	Supported
@org.apache.cxf.interceptor.OutFaultInterceptors	Supported
@org.apache.cxf.interceptor.InFaultInterceptors	Supported

Annotation	Status
@org.apache.cxf.annotations.WSDLDocumentation	Supported
@org.apache.cxf.annotations.WSDLDocumentationCollection	Supported
@org.apache.cxf.annotations.SchemaValidation	Supported
@org.apache.cxf.annotations.DataBinding	Only the default value **org.apache.cxf.jaxb.JAXBDataBinding** is supported
@org.apache.cxf.ext.logging.Logging	Supported
@org.apache.cxf.annotations.GZIP	Supported
@org.apache.cxf.annotations.FastInfoset	Via **com.sun.xml.fastinfoset:FastInfoset** dependency. Allows CXF endpoints and clients to exchange binary messages encoded in FastInfoset format. This use case only has community support.
@org.apache.cxf.annotations.EndpointProperty	Supported
@org.apache.cxf.annotations.EndpointProperties	Supported
@org.apache.cxf.annotations.Policy	Supported
@org.apache.cxf.annotations.Policies	Supported
@org.apache.cxf.annotations.UseAsyncMethod	Supported

CHAPTER 5. QUARKUS CXF EXTENSIONS REFERENCE

This chapter provides reference information about Quarkus CXF extensions.

5.1. QUARKUS CXF

Core capabilities for implementing SOAP clients and JAX-WS services.

5.1.1. Maven coordinates

Create a new project using **quarkus-cxf** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf</artifactId>
</dependency>
```

5.1.2. Supported standards

- JAX-WS
- JAXB
- WS-Addressing
- WS-Policy
- MTOM

5.1.3. Usage

There are several chapters in the User guide covering the usage of this extension.

5.1.4. Configuration

The padlock icon indicates a configuration property that is fixed at build time. All other configuration properties are overridable at runtime.

Configuration property	Type	Default
quarkus.cxf.path	string	/services

Configuration property	Type	Default
The default path for CXF resources.  **EARLIER VERSIONS** The default value before Quarkus CXF version 2.0.0 was /. Environment variable: **QUARKUS_CXF_PATH**		
quarkus.cxf.min-chunk-size	int	128
The size in bytes of the chunks of memory allocated when writing data. This is a very advanced setting that should only be set if you understand exactly how it affects the output IO operations of the application. Environment variable: **QUARKUS_CXF_MIN_CHUNK_SIZE**		
quarkus.cxf.output-buffer-size	int	8191
The size of the output stream response buffer in bytes. If a response is larger than this and no content-length is provided then the response will be chunked. Larger values may give slight performance increases for large responses, at the expense of more memory usage. Environment variable: **QUARKUS_CXF_OUTPUT_BUFFER_SIZE**		
quarkus.cxf.decoupled-endpoint-base	string	

An URI base to use as a prefix of **quarkus.cxf.client.myClient.decoupled-endpoint**. You will typically want to set this to something like the following:
 	
`quarkus.cxf.decoupled-endpoint-base = https://api.example.com:${quarkus.http.ssl-port}${quarkus.cxf.path}`
 	
or, for plain HTTP:
 	
`quarkus.cxf.decoupled-endpoint-base = http://api.example.com:${quarkus.http.port}${quarkus.cxf.path}`
 	
If you invoke your WS client from within a HTTP handler, you can leave this option unspecified and rather set it dynamically on the request context of your WS client using the **org.apache.cxf.ws.addressing.decoupled.endpoint.base** key. Here is an example how to do that from a RESTeasy handler method:
 	
`import java.util.Map;`
`import jakarta.inject.Inject;`
`import jakarta.ws.rs.POST;`		

```
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.UriInfo;
import jakarta.xml.ws.BindingProvider;
import io.quarkiverse.cxf.annotation.CXFClient;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Path("/my-rest")
public class MyRestEasyResource {

    @Inject
    @CXFClient("hello")
    HelloService helloService;

    @ConfigProperty(name = "quarkus.cxf.path")
                String quarkusCxfPath;

    @POST
    @Path("/hello")
    @Produces(MediaType.TEXT_PLAIN)
       public String hello(String body, @Context UriInfo uriInfo) throws IOException {

       // You may consider doing this only once if you are sure that your service is accessed
       // through a single hostname
       String decoupledEndpointBase = uriInfo.getBaseUriBuilder().path(quarkusCxfPath);
       Map>String, Object< requestContext = ((BindingProvider)
       helloService).getRequestContext();
       requestContext.put("org.apache.cxf.ws.addressing.decoupled.endpoint.base",
       decoupledEndpointBase);

       return wsrmHelloService.hello(body);
    }
}
```

Configuration property	Type	Default
Environment variable: **QUARKUS_CXF_DECOUPLED_ENDPOINT_BASE** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.logging.enabled-for	**clients**, **services**, **both**, **none**	**none**
Specifies whether the message logging will be enabled for clients, services, both or none. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.enabled** or **quarkus.cxf.client."clients".logging.enabled** respectively. **Environment variable**: **QUARKUS_CXF_LOGGING_ENABLED_FOR** **Since Quarkus CXF**: 2.6.0		

Configuration property	Type	Default
quarkus.cxf.logging.pretty	boolean	false
If **true**, the XML elements will be indented in the log; otherwise they will appear unindented. This setting can be overridden per client or service endpoint using quarkus.cxf.endpoint."endpoints".logging.pretty or quarkus.cxf.client."clients".logging.pretty respectively. Environment variable: **QUARKUS_CXF_LOGGING_PRETTY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.limit	int	49152
A message length in bytes at which it is truncated in the log. This setting can be overridden per client or service endpoint using quarkus.cxf.endpoint."endpoints".logging.limit or quarkus.cxf.client."clients".logging.limit respectively. Environment variable: **QUARKUS_CXF_LOGGING_LIMIT** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.in-mem-threshold	long	-1
A message length in bytes at which it will be written to disk. **-1** is unlimited. This setting can be overridden per client or service endpoint using quarkus.cxf.endpoint."endpoints".logging.in-mem-threshold or quarkus.cxf.client."clients".logging.in-mem-threshold respectively. Environment variable: **QUARKUS_CXF_LOGGING_IN_MEM_THRESHOLD** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.log-binary	boolean	false
If **true**, binary payloads will be logged; otherwise they won't be logged. This setting can be overridden per client or service endpoint using quarkus.cxf.endpoint."endpoints".logging.log-binary or quarkus.cxf.client."clients".logging.log-binary respectively. Environment variable: **QUARKUS_CXF_LOGGING_LOG_BINARY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.log-multipart	boolean	true
If **true**, multipart payloads will be logged; otherwise they won't be logged. This setting can be overridden per client or service endpoint using quarkus.cxf.endpoint."endpoints".logging.log-multipart or quarkus.cxf.client."clients".logging.log-multipart respectively. Environment variable: **QUARKUS_CXF_LOGGING_LOG_MULTIPART** **Since Quarkus CXF**: 2.6.0		

Configuration property	Type	Default
quarkus.cxf.logging.verbose	boolean	true
If **true**, verbose logging will be enabled; otherwise it won't be enabled. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.verbose** or **quarkus.cxf.client."clients".logging.verbose** respectively. Environment variable: **QUARKUS_CXF_LOGGING_VERBOSE** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.in-binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.in-binary-content-media-types** or **quarkus.cxf.client."clients".logging.in-binary-content-media-types** respectively. Environment variable: **QUARKUS_CXF_LOGGING_IN_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.out-binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** whose content will not be logged unless**log-binary** is **true**. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.out-binary-content-media-types** or **quarkus.cxf.client."clients".logging.out-binary-content-media-types** respectively. Environment variable: **QUARKUS_CXF_LOGGING_OUT_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** and **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.binary-content-media-types** or **quarkus.cxf.client."clients".logging.binary-content-media-types** respectively. Environment variable: **QUARKUS_CXF_LOGGING_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.sensitive-element-names	List of **string**	

Configuration property	Type	Default
A comma separated list of XML elements containing sensitive information to be masked in the log. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.sensitive-element-names** or **quarkus.cxf.client."clients".logging.sensitive-element-names** respectively. Environment variable: **QUARKUS_CXF_LOGGING_SENSITIVE_ELEMENT_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.logging.sensitive-protocol-header-names	List of **string**	
A comma separated list of protocol headers containing sensitive information to be masked in the log. This setting can be overridden per client or service endpoint using **quarkus.cxf.endpoint."endpoints".logging.sensitive-protocol-header-names** or **quarkus.cxf.client."clients".logging.sensitive-protocol-header-names** respectively. Environment variable: **QUARKUS_CXF_LOGGING_SENSITIVE_PROTOCOL_HEADER_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".service-interface	**string**	
The client service interface class name Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SERVICE_INTERFACE**		
quarkus.cxf.client."clients".alternative	**boolean**	**false**
Indicates whether this is an alternative proxy client configuration. If true, then this configuration is ignored when configuring a client without annotation **@CXFClient**. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ALTERNATIVE**		
quarkus.cxf.client."clients".native.runtime-initialized	**boolean**	**false**
If **true**, the client dynamic proxy class generated by native compiler will be initialized at runtime; otherwise the proxy class will be initialized at build time. Setting this to **true** makes sense if your service endpoint interface references some class initialized at runtime in its method signatures. E.g. Say, your service interface has method **int add(Operands o)** and the **Operands** class was requested to be initialized at runtime. Then, without setting this configuration parameter to **true**, the native compiler will throw an exception saying something like**Classes that should be initialized at run time got initialized during image building: org.acme.Operands … jdk.proxy<some-number>.$Proxy<some-number> caused initialization of this class**. **jdk.proxy<some-number>.$Proxy<some-number>** is the proxy class generated by the native compiler. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__NATIVE_RUNTIME_INITIALIZED**		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".implementor	string	
The service endpoint implementation class Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__IMPLEMENTOR**		
quarkus.cxf.endpoint."endpoints".wsdl	string	
The service endpoint WSDL path Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__WSDL**		
quarkus.cxf.endpoint."endpoints".soap-binding	string	
The URL of the SOAP Binding, should be one of four values: • **http://schemas.xmlsoap.org/wsdl/soap/http** for SOAP11HTTP_BINDING • **http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true** for SOAP11HTTP_MTOM_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/** for SOAP12HTTP_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true** for SOAP12HTTP_MTOM_BINDING Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SOAP_BINDING**		
quarkus.cxf.endpoint."endpoints".published-endpoint-url	string	
The published service endpoint URL Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__PUBLISHED_ENDPOINT_URL**		
quarkus.cxf.endpoint."endpoints".logging.enabled	**true**, **false**, **pretty**	
If **true** or **pretty**, the message logging will be enabled; otherwise it will not be enabled. If the value is **pretty** (since 2.7.0), the **pretty** attribute will effectively be set to **true**. The default is given by quarkus.cxf.logging.enabled-for. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_ENABLED** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.pretty	**boolean**	

Configuration property	Type	Default
If **true**, the XML elements will be indented in the log; otherwise they will appear unindented. The default is given by **quarkus.cxf.logging.pretty** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_PRETTY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.limit	**int**	
A message length in bytes at which it is truncated in the log. The default is given by **quarkus.cxf.logging.limit** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_LIMIT** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.in-mem-threshold	**long**	
A message length in bytes at which it will be written to disk. **-1** is unlimited. The default is given by **quarkus.cxf.logging.in-mem-threshold** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_IN_MEM_THRESHOLD** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.log-binary	**boolean**	
If **true**, binary payloads will be logged; otherwise they won't be logged. The default is given by **quarkus.cxf.logging.log-binary** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_LOG_BINARY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.log-multipart	**boolean**	
If **true**, multipart payloads will be logged; otherwise they won't be logged. The default is given by **quarkus.cxf.logging.log-multipart** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_LOG_MULTIPART** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.verbose	**boolean**	

Configuration property	Type	Default
If **true**, verbose logging will be enabled; otherwise it won't be enabled. The default is given by **quarkus.cxf.logging.verbose** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_VERBOSE** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.in-binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by **quarkus.cxf.logging.in-binary-content-media-types** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_IN_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.out-binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by **quarkus.cxf.logging.out-binary-content-media-types** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_OUT_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** and **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by**quarkus.cxf.logging.binary-content-media-types** Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.sensitive-element-names	List of **string**	

Configuration property	Type	Default
A comma separated list of XML elements containing sensitive information to be masked in the log. The default is given by **quarkus.cxf.logging.sensitive-element-names** **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_SENSITIVE_ELEMENT_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".logging.sensitive-protocol-header-names	List of **string**	
A comma separated list of protocol headers containing sensitive information to be masked in the log. The default is given by **quarkus.cxf.logging.sensitive-protocol-header-names** **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__LOGGING_SENSITIVE_PROTOCOL_HEADER_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.endpoint."endpoints".features	List of **string**	

A comma-separated list of fully qualified CXF Feature class names or named CDI beans.
 	
Examples:
 	
`quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature`
`quarkus.cxf.endpoint."/fruit".features = #myCustomLoggingFeature`
 	
In the second case, the **#myCustomLoggingFeature** bean can be produced as follows:
 	
`import org.apache.cxf.ext.logging.LoggingFeature;`
`import javax.enterprise.context.ApplicationScoped;`
`import javax.enterprise.inject.Produces;`
 	
`class Producers {`
 	
`@Produces`
`@ApplicationScoped`
`LoggingFeature myCustomLoggingFeature() {`
`LoggingFeature loggingFeature = new LoggingFeature();`
`loggingFeature.setPrettyLogging(true);`
`return loggingFeature;`
`}`
`}`
 	
Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__FEATURES**		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".handlers	List of **string**	
The comma-separated list of Handler classes Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__HANDLERS**		
quarkus.cxf.endpoint."endpoints".in-interceptors	List of **string**	
The comma-separated list of InInterceptor classes Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__IN_INTERCEPTORS**		
quarkus.cxf.endpoint."endpoints".out-interceptors	List of **string**	
The comma-separated list of OutInterceptor classes Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__OUT_INTERCEPTORS**		
quarkus.cxf.endpoint."endpoints".out-fault-interceptors	List of **string**	
The comma-separated list of OutFaultInterceptor classes Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__OUT_FAULT_INTERCEPTORS**		
quarkus.cxf.endpoint."endpoints".in-fault-interceptors	List of **string**	
The comma-separated list of InFaultInterceptor classes Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__IN_FAULT_INTERCEPTORS**		
quarkus.cxf.endpoint."endpoints".schema-validation.enabled-for	**in**, **request**, **out**, **response**, **both**, **none**	

Configuration property	Type	Default
Select for which messages XML Schema validation should be enabled. If not specified, no XML Schema validation will be enforced unless it is enabled by other means, such as **@org.apache.cxf.annotations.SchemaValidation** or **@org.apache.cxf.annotations.EndpointProperty(key = "schema-validation-enabled", value = "true")** annotations. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SCHEMA_VALIDATION_ENABLED_FOR** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".wsdl	**string**	
A URL, resource path or local filesystem path pointing to a WSDL document to use when generating the service proxy of this client. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__WSDL**		
quarkus.cxf.client."clients".soap-binding	**string**	
The URL of the SOAP Binding, should be one of four values: • **http://schemas.xmlsoap.org/wsdl/soap/http** for SOAP11HTTP_BINDING • **http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true** for SOAP11HTTP_MTOM_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/** for SOAP12HTTP_BINDING • **http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true** for SOAP12HTTP_MTOM_BINDING Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SOAP_BINDING**		
quarkus.cxf.client."clients".client-endpoint-url	**string**	
The client endpoint URL Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CLIENT_ENDPOINT_URL**		
quarkus.cxf.client."clients".endpoint-namespace	**string**	
The client endpoint namespace Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ENDPOINT_NAMESPACE**		
quarkus.cxf.client."clients".endpoint-name	**string**	
The client endpoint name Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ENDPOINT_NAME**		

Configuration property	Type	Default
quarkus.cxf.client."clients".username	string	
The username for HTTP Basic authentication Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__USERNAME**		
quarkus.cxf.client."clients".password	string	
The password for HTTP Basic authentication Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PASSWORD**		
quarkus.cxf.client."clients".secure-wsdl-access	boolean	false
If **true**, then the **Authentication** header will be sent preemptively when requesting the WSDL, as long as the **username** is set; otherwise the WSDL will be requested anonymously. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURE_WSDL_ACCESS** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".logging.enabled	true, false, pretty	
If **true** or **pretty**, the message logging will be enabled; otherwise it will not be enabled. If the value is **pretty** (since 2.7.0), the **pretty** attribute will effectively be set to **true**. The default is given by quarkus.cxf.logging.enabled-for. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_ENABLED** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.pretty	boolean	
If **true**, the XML elements will be indented in the log; otherwise they will appear unindented. The default is given by quarkus.cxf.logging.pretty Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_PRETTY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.limit	int	
A message length in bytes at which it is truncated in the log. The default is given by quarkus.cxf.logging.limit Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_LIMIT** **Since Quarkus CXF**: 2.6.0		

Configuration property	Type	Default
quarkus.cxf.client."clients".logging.in-mem-threshold	**long**	
A message length in bytes at which it will be written to disk. **-1** is unlimited. The default is given by **quarkus.cxf.logging.in-mem-threshold** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_IN_MEM_THRESHOLD** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.log-binary	**boolean**	
If **true**, binary payloads will be logged; otherwise they won't be logged. The default is given by **quarkus.cxf.logging.log-binary** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_LOG_BINARY** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.log-multipart	**boolean**	
If **true**, multipart payloads will be logged; otherwise they won't be logged. The default is given by **quarkus.cxf.logging.log-multipart** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_LOG_MULTIPART** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.verbose	**boolean**	
If **true**, verbose logging will be enabled; otherwise it won't be enabled. The default is given by **quarkus.cxf.logging.verbose** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_VERBOSE** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.in-binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by **quarkus.cxf.logging.in-binary-content-media-types** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_IN_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.out-binary-content-media-types	List of **string**	

Configuration property	Type	Default
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by **quarkus.cxf.logging.out-binary-content-media-types** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_OUT_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.binary-content-media-types	List of **string**	
A comma separated list of additional binary media types to add to the default values in the **LoggingOutInterceptor** and **LoggingInInterceptor** whose content will not be logged unless**log-binary** is **true**. The default is given by**quarkus.cxf.logging.binary-content-media-types** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_BINARY_CONTENT_MEDIA_TYPES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.sensitive-element-names	List of **string**	
A comma separated list of XML elements containing sensitive information to be masked in the log. The default is given by **quarkus.cxf.logging.sensitive-element-names** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_SENSITIVE_ELEMENT_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".logging.sensitive-protocol-header-names	List of **string**	
A comma separated list of protocol headers containing sensitive information to be masked in the log. The default is given by **quarkus.cxf.logging.sensitive-protocol-header-names** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__LOGGING_SENSITIVE_PROTOCOL_HEADER_NAMES** **Since Quarkus CXF**: 2.6.0		
quarkus.cxf.client."clients".features	List of **string**	

Configuration property	Type	Default
A comma-separated list of fully qualified CXF Feature class names. Example: `quarkus.cxf.endpoint.myClient.features = org.apache.cxf.ext.logging.LoggingFeature` Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__FEATURES**		
quarkus.cxf.client."clients".handlers	List of **string**	
The comma-separated list of Handler classes Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__HANDLERS**		
quarkus.cxf.client."clients".in-interceptors	List of **string**	
The comma-separated list of InInterceptor classes Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__IN_INTERCEPTORS**		
quarkus.cxf.client."clients".out-interceptors	List of **string**	
The comma-separated list of OutInterceptor classes Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__OUT_INTERCEPTORS**		
quarkus.cxf.client."clients".out-fault-interceptors	List of **string**	
The comma-separated list of OutFaultInterceptor classes Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__OUT_FAULT_INTERCEPTORS**		
quarkus.cxf.client."clients".in-fault-interceptors	List of **string**	
The comma-separated list of InFaultInterceptor classes Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__IN_FAULT_INTERCEPTORS**		
quarkus.cxf.client."clients".connection-timeout	**long**	**30000**

Configuration property	Type	Default
Specifies the amount of time, in milliseconds, that the consumer will attempt to establish a connection before it times out. 0 is infinite. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CONNECTION_TIMEOUT** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".receive-timeout	**long**	**60000**
Specifies the amount of time, in milliseconds, that the consumer will wait for a response before it times out. 0 is infinite. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__RECEIVE_TIMEOUT** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".connection-request-timeout	**long**	**60000**
Specifies the amount of time, in milliseconds, used when requesting a connection from the connection manager(if appliable). 0 is infinite. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CONNECTION_REQUEST_TIMEOUT** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".auto-redirect	**boolean**	**false**
Specifies if the consumer will automatically follow a server issued redirection. (name is not part of standard) Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__AUTO_REDIRECT** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".max-retransmits	**int**	**-1**
Specifies the maximum amount of retransmits that are allowed for redirects. Retransmits for authorization is included in the retransmit count. Each redirect may cause another retransmit for a UNAUTHORIZED response code, ie. 401. Any negative number indicates unlimited retransmits, although, loop protection is provided. The default is unlimited. (name is not part of standard) Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__MAX_RETRANSMITS** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".allow-chunking	**boolean**	**true**
If true, the client is free to use chunking streams if it wants, but it is not required to use chunking streams. If false, the client must use regular, non-chunked requests in all cases. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ALLOW_CHUNKING** **Since Quarkus CXF**: 2.2.3		

Configuration property	Type	Default
quarkus.cxf.client."clients".chunking-threshold	**int**	**4096**
If AllowChunking is true, this sets the threshold at which messages start getting chunked. Messages under this limit do not get chunked. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CHUNKING_THRESHOLD** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".chunk-length	**int**	**-1**
Specifies the chunk length for a HttpURLConnection. This value is used in java.net.HttpURLConnection.setChunkedStreamingMode(int chunklen). chunklen indicates the number of bytes to write in each chunk. If chunklen is less than or equal to zero, a default value will be used. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CHUNK_LENGTH** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".accept	**string**	
Specifies the MIME types the client is prepared to handle (e.g., HTML, JPEG, GIF, etc.) Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".accept-language	**string**	
Specifies the language the client desires (e.g., English, French, etc.) Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT_LANGUAGE**		
quarkus.cxf.client."clients".accept-encoding	**string**	
Specifies the encoding the client is prepared to handle (e.g., gzip) Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__ACCEPT_ENCODING** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".content-type	**string**	
Specifies the content type of the stream being sent in a post request. (this should be text/xml for web services, or can be set to application/x-www-form-urlencoded if the client is sending form data). Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CONTENT_TYPE** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".host	**string**	

Configuration property	Type	Default
Specifies the Internet host and port number of the resource on which the request is being invoked. This is sent by default based upon the URL. Certain DNS scenarios or application designs may request you to set this, but typically it is not required. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__HOST** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".connection	**close**, **keep-alive**	**keep-alive**
The connection disposition. If close the connection to the server is closed after each request/response dialog. If Keep-Alive the client requests the server to keep the connection open, and if the server honors the keep alive request, the connection is reused. Many servers and proxies do not honor keep-alive requests. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CONNECTION** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".cache-control	**string**	
Most commonly used to specify no-cache, however the standard supports a dozen or so caching related directives for requests Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__CACHE_CONTROL** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".version	**string**	**auto**
HTTP Version used for the connection. The default value **auto** will use whatever the default is for the **HTTPConduit** implementation defined via**quarkus.cxf.client.myClient.http-conduit-factory**. Other possible values: **1.1**, **2**. Some of these values might be unsupported by some **HTTPConduit** implementations. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__VERSION** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".browser-type	**string**	
The value of the **User-Agent** HTTP header. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__BROWSER_TYPE** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".decoupled-endpoint	**string**	

Configuration property	Type	Default
An URI path (starting with /) or a full URI for the receipt of responses over a separate provider → consumer connection. If the value starts with /, then it is prefixed with the base URI configured via **quarkus.cxf.client.myClient.decoupled-endpoint-base** before being used as a value for the WS-Addressing **ReplyTo** message header. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__DECOUPLED_ENDPOINT** Since Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-server	**string**	
Specifies the address of proxy server if one is used. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER** Since Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-server-port	**int**	
Specifies the port number used by the proxy server. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER_PORT** Since Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".non-proxy-hosts	**string**	
Specifies the list of hostnames that will not use the proxy configuration. Examples: • **localhost** - a single hostname • **localhost|www.google.com** - two hostnames that will not use the proxy configuration • **localhost|www.google.*|*.apache.org** - hostname patterns Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__NON_PROXY_HOSTS** Since Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-server-type	**http**, **socks**	**http**
Specifies the type of the proxy server. Can be either HTTP or SOCKS. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PROXY_SERVER_TYPE** Since Quarkus CXF: 2.2.3		
quarkus.cxf.client."clients".proxy-username	**string**	
Username for the proxy authentication Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PROXY_USERNAME** Since Quarkus CXF: 2.2.3		

Configuration property	Type	Default
quarkus.cxf.client."clients".proxy-password	string	
Password for the proxy authentication Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__PROXY_PASSWORD** **Since Quarkus CXF**: 2.2.3		
quarkus.cxf.client."clients".http-conduit-factory	**QuarkusCXFDefault**, **CXFDefault**, **HttpClientHTTPConduitFactory**, **URLConnectionHTTPConduitFactory**	
Select the **HTTPConduitFactory** implementation for this client. • **QuarkusCXFDefault** (default): if **io.quarkiverse.cxf:quarkus-cxf-rt-transports-http-hc5** is present in class path, then its **HTTPConduitFactory** implementation will be used; otherwise this value is equivalent with **URLConnectionHTTPConduitFactory** (this may change, once issue #992 gets resolved in CXF) • **CXFDefault**: the selection of **HTTPConduitFactory** implementation is left to CXF • **HttpClientHTTPConduitFactory**: the **HTTPConduitFactory** for this client will be set to an implementation always returning **org.apache.cxf.transport.http.HttpClientHTTPConduit**. This will use **java.net.http.HttpClient** as the underlying HTTP client. • **URLConnectionHTTPConduitFactory**: the **HTTPConduitFactory** for this client will be set to an implementation always returning **org.apache.cxf.transport.http.URLConnectionHTTPConduit**. This will use **java.net.HttpURLConnection** as the underlying HTTP client. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__HTTP_CONDUIT_FACTORY**		
quarkus.cxf.client."clients".trust-store	string	
The trust store location for this client. The resource is first looked up in the classpath, then in the file system. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE**		

Configuration property	Type	Default
quarkus.cxf.client."clients".trust-store-password	**string**	
The trust store password Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE_PASSWORD**		
quarkus.cxf.client."clients".trust-store-type	**string**	**JKS**
The type of the trust store. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__TRUST_STORE_TYPE**		
quarkus.cxf.client."clients".hostname-verifier	**string**	
Can be one of the following: • One of the well known values: **AllowAllHostnameVerifier**, **HttpsURLConnectionDefaultHostnameVerifier** • A fully qualified class name implementing **javax.net.ssl.HostnameVerifier** to look up in the CDI container. • A bean name prefixed with **#** that will be looked up in the CDI container; example: **#myHostnameVerifier** If not specified, then the creation of the**HostnameVerifier** is delegated to CXF, which boils down to **org.apache.cxf.transport.https.httpclient.DefaultHostnameVerifier** with the default **org.apache.cxf.transport.https.httpclient.PublicSuffixMatcherLoader** as returned from **PublicSuffixMatcherLoader.getDefault()**. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__HOSTNAME_VERIFIER**		
quarkus.cxf.client."clients".schema-validation.enabled-for	**in**, **request**, **out**, **response**, **both**, **none**	
Select for which messages XML Schema validation should be enabled. If not specified, no XML Schema validation will be enforced unless it is enabled by other means, such as **@org.apache.cxf.annotations.SchemaValidation** or **@org.apache.cxf.annotations.EndpointProperty(key = "schema-validation-enabled", value = "true")** annotations. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SCHEMA_VALIDATION_ENABLED_FOR** **Since Quarkus CXF**: 2.7.0		

5.2. METRICS FEATURE

Collect metrics using Micrometer.

IMPORTANT

Unlike CXF Metrics feature, this Quarkus CXF extension does not support Dropwizard Metrics. Only Micrometer is supported.

5.2.1. Maven coordinates

Create a new project using **quarkus-cxf-rt-features-metrics** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
  <groupId>io.quarkiverse.cxf</groupId>
  <artifactId>quarkus-cxf-rt-features-metrics</artifactId>
</dependency>
```

5.2.2. Usage

The integration of CXF into the Quarkus Micrometer ecosystem is implemented using **io.quarkiverse.cxf.metrics.QuarkusCxfMetricsFeature**. As long as your application depends on **quarkus-cxf-rt-features-metrics**, an instance of **QuarkusCxfMetricsFeature** is created internally and enabled by default for all clients and service endpoints created by Quarkus CXF. You can disable it via **quarkus.cxf.metrics.enabled-for**, **quarkus.cxf.client."clients".metrics.enabled** and **quarkus.cxf.endpoint."endpoints".metrics.enabled** properties documented below.

5.2.2.1. Runnable example

There is an integration test covering Micrometer Metrics in the Quarkus CXF source tree.

Unsurprisingly, it depends on **quarkus-cxf-rt-features-metrics**

pom.xml

```
<dependency>
  <groupId>io.quarkiverse.cxf</groupId>
  <artifactId>quarkus-cxf-rt-features-metrics</artifactId>
</dependency>
```

It is using **quarkus-micrometer-registry-prometheus** extension to export the metrics in JSON format and for Prometheus:

pom.xml

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-micrometer-registry-prometheus</artifactId>
</dependency>
```

The following configuration is needed to be able to inspect the collected metrics over a REST endpoint:

application.properties

```
quarkus.micrometer.export.json.enabled = true
quarkus.micrometer.export.json.path = metrics/json
quarkus.micrometer.export.prometheus.path = metrics/prometheus
```

Having all the above in place, you can start the application in Dev mode:

```
$ mvn quarkus:dev
```

Now send a request to the **HelloService**:

```
$ curl \
  -d '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns2:helloResponse xmlns:ns2="http://it.server.metrics.cxf.quarkiverse.io/"><return>Hello Joe!
</return></ns2:helloResponse></soap:Body></soap:Envelope>' \
  -H 'Content-Type: text/xml' \
  -X POST \
  http://localhost:8080/metrics/client/hello
```

After that, metrics are shown under **cxf.server.requests** in the output of the endpoint you configured above:

```
$ curl http://localhost:8080/q/metrics/json
metrics: {
    ...
    "cxf.server.requests": {

"count;exception=None;faultCode=None;method=POST;operation=hello;outcome=SUCCESS;status=20
0;uri=/soap/hello": 2,

"elapsedTime;exception=None;faultCode=None;method=POST;operation=hello;outcome=SUCCESS;st
atus=200;uri=/soap/hello": 64.4
    },
    ...
}
```

5.2.3. Configuration

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

Configuration property	Type	Default
quarkus.cxf.metrics.enabled-for	**clients**, **services**, **both**, **none**	**both**

Configuration property	Type	Default
Specifies whether the metrics collection will be enabled for clients, services, both or none. This global setting can be overridden per client or service endpoint using the **quarkus.cxf.client."clients".metrics.enabled** or **quarkus.cxf.endpoint."endpoints".metrics.enabled** option respectively. Environment variable: **QUARKUS_CXF_METRICS_ENABLED_FOR** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".metrics.enabled	**boolean**	**true**
If **true** and if **quarkus.cxf.metrics.enabled-for** is set to **both** or **clients** then the **MetricsFeature** will be added to this client; otherwise the feature will not be added to this client. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__METRICS_ENABLED** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.endpoint."endpoints".metrics.enabled	**boolean**	**true**
If **true** and if **quarkus.cxf.metrics.enabled-for** is set to **both** or **services** then the **MetricsFeature** will be added to this service endpoint; otherwise the feature will not be added to this service endpoint. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__METRICS_ENABLED** **Since Quarkus CXF**: 2.7.0		

5.3. OPENTELEMETRY

Generate OpenTelemetry traces.

IMPORTANT

OpenTelemetry Metrics and Logging are not supported yet on neither Quarkus nor CXF side, hence Quarkus CXF cannot support them either. Therefore, tracing is the only OpenTelemetry feature supported by this extension.

5.3.1. Maven coordinates

Create a new project using **quarkus-cxf-integration-tracing-opentelemetry** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-integration-tracing-opentelemetry</artifactId>
</dependency>
```

5.3.2. Usage

This extension builds on top of **org.apache.cxf.tracing.opentelemetry.OpenTelemetryFeature** (for service endpoints) and **org.apache.cxf.tracing.opentelemetry.OpenTelemetryClientFeature** (for clients). Instances of these are created and configured internally using the instance of **io.opentelemetry.api.OpenTelemetry** provided by Quarkus OpenTelemetry.

The tracing is enabled by default for all clients and service endpoints created by Quarkus CXF, unless you disable it explicitly via **quarkus.cxf.otel.enabled-for**, **quarkus.cxf.client."clients".otel.enabled** or **quarkus.cxf.endpoint."endpoints".otel.enabled**.

5.3.2.1. Runnable example

There is an integration test covering OpenTelemetry in the Quarkus CXF source tree. It is using **InMemorySpanExporter** from **io.opentelemetry:opentelemetry-sdk-testing**, so that the spans can be inspected from tests easily. Refer to Quarkus OpenTelemetry guide for information about other supported span exporters and collectors.

5.3.3. Configuration

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

Configuration property	Type	Default
quarkus.cxf.otel.enabled-for	**clients**, **services**, **both**, **none**	**both**
Specifies whether the OpenTelemetry tracing will be enabled for clients, services, both or none. This global setting can be overridden per client or service endpoint using the **quarkus.cxf.client."clients".otel.enabled** or **quarkus.cxf.endpoint."endpoints".otel.enabled** option respectively. Environment variable: **QUARKUS_CXF_OTEL_ENABLED_FOR** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".otel.enabled	**boolean**	**true**
If **true** and if **quarkus.cxf.otel.enabled-for** is set to **both** or **clients** then the **OpenTelemetryClientFeature** will be added to this client; otherwise the feature will not be added to this client. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__OTEL_ENABLED** **Since Quarkus CXF**: 2.7.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".otel.enabled	boolean	true
If **true** and if **quarkus.cxf.otel.enabled-for** is set to **both** or **services** then the **OpenTelemetryFeature** will be added to this service endpoint; otherwise the feature will not be added to this service endpoint. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__OTEL_ENABLED** **Since Quarkus CXF**: 2.7.0		

5.4. WS-SECURITY

Provides CXF framework's WS-Security implementation allowing you to:

- Pass authentication tokens between services
- Encrypt messages or parts of messages
- Sign messages
- Timestamp messages

5.4.1. Maven coordinates

Create a new project using **quarkus-cxf-rt-ws-security** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-ws-security</artifactId>
</dependency>
```

5.4.2. Supported standards

- WS-Security
- WS-SecurityPolicy

5.4.3. Usage

The CXF framework's WS-Security (WSS) implementation is based on WSS4J. It can be activated in two ways:

- By using WS-SecurityPolicy
- By adding WSS4J interceptors to your clients and service endpoints.

WS-SecurityPolicy is preferable because in that way, the security requirements become a part of the WSDL contract. That in turn greatly simplifies not only the implementation of clients and service endpoints but also the interoperability between vendors.

Nevertheless, if you leverage WS-SecurityPolicy, CXF sets up the WSS4J interceptors under the hood for you.

We won't explain the manual approach with WSS4J interceptors in detail here, but you can still refer to our WS-Security integration test as an example.

5.4.3.1. WS-Security via WS-SecurityPolicy

TIP

The sample code snippets used in this section come from the WS-SecurityPolicy integration test in the source tree of Quarkus CXF

Let's say our aim is to ensure that the communication between the client and service is confidential (through encryption) and that the message has not been tampered with (through digital signatures). We also want to assure that the clients are who they claim to be by authenticating themselves by X.509 certificates.

We can express all these requirements in a single WS-SecurityPolicy document:

encrypt-sign-policy.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsp:Policy wsu:Id="SecurityServiceEncryptThenSignPolicy"
   xmlns:wsp="http://www.w3.org/ns/ws-policy"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
   <wsp:ExactlyOne>
      <wsp:All>
         (1)
         <sp:AsymmetricBinding xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
            <wsp:Policy>
               (2)
               <sp:InitiatorToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                        <wsp:Policy>
                           <sp:WssX509V3Token11/>
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:InitiatorToken>
               <sp:RecipientToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/Never">
                        <wsp:Policy>
                           <sp:WssX509V3Token11/>
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:RecipientToken>
```

```
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic256/>
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict/>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp/>
            <sp:ProtectTokens/>
            <sp:OnlySignEntireHeadersAndBody/>
            <sp:EncryptBeforeSigning/>
          </wsp:Policy>
        </sp:AsymmetricBinding>
        3
        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
        </sp:SignedParts>
        4
        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <sp:Body/>
        </sp:EncryptedParts>
        <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:MustSupportRefIssuerSerial/>
          </wsp:Policy>
        </sp:Wss10>
      </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

1 **AsymmetricBinding** specifies the use of asymmetric (public/private key) cryptography for securing the communication between two parties

2 **InitiatorToken** indicates that the initiator (sender) of the message will use an X.509 certificate token that must always be provided to the recipient.

3 **SignedParts** specifies which parts of the SOAP message must be signed to ensure their integrity.

4 **EncryptedParts** specifies the parts of the SOAP message that must be encrypted to ensure their confidentiality.

We set this policy on the Service Endpoint Interface (SEI) **EncryptSignPolicyHelloService** using **@org.apache.cxf.annotations.Policy** annotation:

EncryptSignPolicyHelloService.java

```
@WebService(serviceName = "EncryptSignPolicyHelloService")
@Policy(placement = Policy.Placement.BINDING, uri = "encrypt-sign-policy.xml")
public interface EncryptSignPolicyHelloService extends AbstractHelloService {
...
}
```

On the first sight, setting the policy on the SEI should suffice to enforce it on both the service and all clients generated from the SEI or from the WSDL served by the service. However, that's not all. Security keys, usernames, passwords and other kinds of confidental information cannot be exposed in a public policy.

Those have to be set in the configuration.

NOTE

The server and client references **helloEncryptSign** differently, since there are differences in what the reference represents:

The server reference is a path to the service endpoint, accessible via HTTP, which means it has to start with /.

The client reference is a label. It is used to group the client options and to specify the client for injection with **@CXFClient("helloEncryptSign")**.

The label for the client can be completely different (for example, e.g. **foo**), or start with /. To associate with the service, the important thing is that you provide a correct **client-endpoint-url** value.

Let's do it for the service first:

application.properties

```
# A service with encrypt-sign-policy.xml set
quarkus.cxf.endpoint."/helloEncryptSign".implementor =
io.quarkiverse.cxf.it.security.policy.EncryptSignPolicyHelloServiceImpl
# can be jks or pkcs12 - set from Maven profiles in this test
keystore.type = ${keystore.type}
# Signature settings
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.username = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.password = password
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.security.crypto.pr
ovider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.security.crypto.m
erlin.keystore.type" = ${keystore.type}
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.security.crypto.m
erlin.keystore.password" = password
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.security.crypto.m
erlin.keystore.alias" = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.signature.properties."org.apache.ws.security.crypto.m
erlin.file" = bob.${keystore.type}
# Encryption settings
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.username = alice
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.security.crypto.p
rovider" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.security.crypto.n
erlin.keystore.type" = ${keystore.type}
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.security.crypto.n
erlin.keystore.password" = password
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.security.crypto.n
```

```
erlin.keystore.alias" = bob
quarkus.cxf.endpoint."/helloEncryptSign".security.encryption.properties."org.apache.ws.security.crypto.m
erlin.file" = bob.${keystore.type}
```

Similar setup is necessary on the client side:

application.properties

```
# A client with encrypt-sign-policy.xml set
quarkus.cxf.client.helloEncryptSign.client-endpoint-url = https://localhost:${quarkus.http.test-ssl-
port}/services/helloEncryptSign
quarkus.cxf.client.helloEncryptSign.service-interface =
io.quarkiverse.cxf.it.security.policy.EncryptSignPolicyHelloService
quarkus.cxf.client.helloEncryptSign.features = #messageCollector
# The client-endpoint-url above is HTTPS, so we have to setup the server's SSL certificates
quarkus.cxf.client.helloEncryptSign.trust-store = client-truststore.${keystore.type}
quarkus.cxf.client.helloEncryptSign.trust-store-password = password
# Signature settings
quarkus.cxf.client.helloEncryptSign.security.signature.username = alice
quarkus.cxf.client.helloEncryptSign.security.signature.password = password
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.crypto.provide
" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.crypto.merlin.k
eystore.type" = pkcs12
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.crypto.merlin.k
eystore.password" = password
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.crypto.merlin.k
eystore.alias" = alice
quarkus.cxf.client.helloEncryptSign.security.signature.properties."org.apache.ws.security.crypto.merlin.fi
e" = alice.${keystore.type}
# Encryption settings
quarkus.cxf.client.helloEncryptSign.security.encryption.username = bob
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.crypto.provide
r" = org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.crypto.merlin.
eystore.type" = pkcs12
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.crypto.merlin.
eystore.password" = password
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.crypto.merlin.
eystore.alias" = alice
quarkus.cxf.client.helloEncryptSign.security.encryption.properties."org.apache.ws.security.crypto.merlin.
le" = alice.${keystore.type}
```

To inspect the flow of the messages, you can execute the **EncryptSignPolicyTest** as follows:

```
# Clone the repository
$ git clone https://github.com/quarkiverse/quarkus-cxf.git -o upstream
$ cd quarkus-cxf
# Build the whole source tree
$ mvn clean install -DskipTests -Dquarkus.build.skip
# Run the test
$ cd integration-tests/ws-security-policy
$ mvn clean test -Dtest=EncryptSignPolicyTest
```

Some messages containing **Signature** elements and encrypted bodies are shown in the console output.

5.4.4. Configuration

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

Configuration property	Type	Default
quarkus.cxf.client."clients".security.username	**string**	
The user's name. It is used as follows: • As the name in the UsernameToken for WS-Security • As the alias name in the keystore to get the user's cert and private key for signature if **signature.username** is not set • As the alias name in the keystore to get the user's public key for encryption if **encryption.username** is not set Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.password	**string**	
The user's password when a **callback-handler** is not defined. This is only used for the password in a WS-Security UsernameToken. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_PASSWORD** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.username	**string**	
The user's name for signature. It is used as the alias name in the keystore to get the user's cert and private key for signature. If this is not defined, then **username** is used instead. If that is also not specified, it uses the the default alias set in the properties file referenced by **signature.properties**. If that's also not set, and the keystore only contains a single key, that key will be used. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_USERNAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.password	**string**	
The user's password for signature when a **callback-handler** is not defined. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_PASSWORD** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.encryption.username	**string**	

Configuration property	Type	Default
The user's name for encryption. It is used as the alias name in the keystore to get the user's public key for encryption. If this is not defined, then **username** is used instead. If that is also not specified, it uses the default alias set in the properties file referenced by **encrypt.properties**. If that's also not set, and the keystore only contains a single key, that key will be used. For the WS-Security web service provider, the **useReqSigCert** value can be used to accept (encrypt to) any client whose public key is in the service's truststore (defined in **encrypt.properties**). **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_USERNAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.callback-handler	**string**	
A reference to a **javax.security.auth.callback.CallbackHandler** bean used to obtain passwords, for both outbound and inbound requests. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CALLBACK_HANDLER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.saml-callback-handler	**string**	
A reference to a **javax.security.auth.callback.CallbackHandler** implementation used to construct SAML Assertions. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_CALLBACK_HANDLER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.signature.properties	**Map<String,String>**	
The Crypto property configuration to use for signing, if **signature.crypto** is not set. Example [prefix].signature.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].signature.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_PROPERTIES** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.client."clients".security.encryption.properties	**Map<String,String>**	
The Crypto property configuration to use for encryption, if **encryption.crypto** is not set. Example [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_PROPERTIES** Since Quarkus CXF: 2.5.0		
quarkus.cxf.client."clients".security.signature.crypto	**string**	
A reference to a **org.apache.wss4j.common.crypto.Crypto** bean to be used for signature. If not set, **signature.properties** will be used to configure a **Crypto** instance. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SIGNATURE_CRYPTO** Since Quarkus CXF: 2.5.0		
quarkus.cxf.client."clients".security.encryption.crypto	**string**	
A reference to a **org.apache.wss4j.common.crypto.Crypto** to be used for encryption. If not set, **encryption.properties** will be used to configure a **Crypto** instance. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_CRYPTO** Since Quarkus CXF: 2.5.0		
quarkus.cxf.client."clients".security.encryption.certificate	**string**	
A message property for prepared X509 certificate to be used for encryption. If this is not defined, then the certificate will be either loaded from the keystore **encryption.properties** or extracted from request (when WS-Security is used and if **encryption.username** has value **useReqSigCert**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENCRYPTION_CERTIFICATE** Since Quarkus CXF: 2.5.0		
quarkus.cxf.client."clients".security.enable-revocation	**boolean**	**false**

Configuration property	Type	Default
If **true**, Certificate Revocation List (CRL) checking is enabled when verifying trust in a certificate; otherwise it is not enabled. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_REVOCATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable-unsigned-saml-assertion-principal	**boolean**	**false**
If **true**, unsigned SAML assertions will be allowed as SecurityContext Principals; otherwise they won't be allowed as SecurityContext Principals.  SIGNATURE The label "unsigned" refers to an internal signature. Even if the token is signed by an external signature (as per the "sender-vouches" requirement), this boolean must still be configured if you want to use the token to set up the security context. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_UNSIGNED_SAML_ASSERTION_PRINCIPAL** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.validate-saml-subject-confirmation	**boolean**	**true**
If **true**, the **SubjectConfirmation** requirements of a received SAML Token (sender-vouches or holder-of-key) will be validated; otherwise they won't be validated. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_VALIDATE_SAML_SUBJECT_CONFIRMATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.sc-from-jaas-subject	**boolean**	**true**

Configuration property	Type	Default
If **true**, security context can be created from JAAS Subject; otherwise it must not be created from JAAS Subject. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SC_FROM_JAAS_SUBJECT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.audience-restriction-validation	**boolean**	**true**
If **true**, then if the SAML Token contains Audience Restriction URIs, one of them must match one of the values in **audience.restrictions**; otherwise the SAML AudienceRestriction validation is disabled. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_AUDIENCE_RESTRICTION_VALIDATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.saml-role-attributename	**string**	**http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role**
The attribute URI of the SAML **AttributeStatement** where the role information is stored. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_ROLE_ATTRIBUTENAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.subject-cert-constraints	**string**	

Configuration property	Type	Default
A String of regular expressions (separated by the value specified in **security.cert.constraints.separator**) which will be applied to the subject DN of the certificate used for signature validation, after trust verification of the certificate chain associated with the certificate. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SUBJECT_CERT_CONSTRAINTS** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.cert-constraints-separator	**string**	**,**
The separator that is used to parse certificate constraints configured in **security.subject.cert.constraints** This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CERT_CONSTRAINTS_SEPARATOR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.actor	**string**	
The actor or role name of the **wsse:Security** header. If this parameter is omitted, the actor name is not set. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ACTOR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.validate.token	**boolean**	**true**
If **true**, the password of a received **UsernameToken** will be validated; otherwise it won't be validated. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_VALIDATE_TOKEN** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.username-token.always.encrypted	**boolean**	**true**

Configuration property	Type	Default
Whether to always encrypt **UsernameTokens** that are defined as a**SupportingToken**. This should not be set to **false** in a production environment, as it exposes the password (or the digest of the password) on the wire. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAME_TOKEN_ALWAYS_ENCRYPTED** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.is-bsp-compliant	**boolean**	**true**
If **true**, the compliance with the Basic Security Profile (BSP) 1.1 will be ensured; otherwise it will not be ensured. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_IS_BSP_COMPLIANT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable.nonce.cache	**boolean**	
If **true**, the **UsernameToken** nonces will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators.  CACHING Caching only applies when either a **UsernameToken** WS-SecurityPolicy is in effect, or the **UsernameToken** action has been configured for the non-security-policy case. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_NONCE_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable.timestamp.cache	**boolean**	

Configuration property	Type	Default
If **true**, the **Timestamp Created** Strings (these are only cached in conjunction with a message Signature) will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators.  **CACHING** Caching only applies when either a **IncludeTimestamp** policy is in effect, or the **Timestamp** action has been configured for the non-security-policy case. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_TIMESTAMP_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable.streaming	**boolean**	**false**
If **true**, the new streaming (StAX) implementation of WS-Security is used; otherwise the old DOM implementation is used. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_STREAMING** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.return.security.error	**boolean**	**false**
If **true**, detailed security error messages are sent to clients; otherwise the details are omitted and only a generic error message is sent. The "real" security errors should not be returned to the client in production, as they may leak information about the deployment, or otherwise provide an "oracle" for attacks. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_RETURN_SECURITY_ERROR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.must-understand	**boolean**	**true**
If **true**, the SOAP **mustUnderstand** header is included in security headers based on a WS-SecurityPolicy; otherwise the header is always omitted. Works only with **enable.streaming = true** - see CXF-8940 Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_MUST_UNDERSTAND** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.enable.saml.cache	**boolean**	

Configuration property	Type	Default
If **true** and in case the token contains a **OneTimeUse** Condition, the SAML2 Token Identifiers will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators. Caching only applies when either a **SamlToken** policy is in effect, or a SAML action has been configured for the non-security-policy case. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ENABLE_SAML_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.store.bytes.in.attachment	**boolean**	
Whether to store bytes (CipherData or BinarySecurityToken) in an attachment. The default is true if MTOM is enabled. Set it to false to BASE-64 encode the bytes and "inlined" them in the message instead. Setting this to true is more efficient, as it means that the BASE-64 encoding step can be skipped. This only applies to the DOM WS-Security stack. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STORE_BYTES_IN_ATTACHMENT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.swa.encryption.attachment.transform.content	**boolean**	**false**
If **true**, **Attachment-Content-Only** transform will be used when an Attachment is encrypted via a WS-SecurityPolicy expression; otherwise **Attachment-Complete** transform will be used when an Attachment is encrypted via a WS-SecurityPolicy expression. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SWA_ENCRYPTION_ATTACHMENT_TRANSFORM_CONTENT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.use.str.transform	**boolean**	**true**

Configuration property	Type	Default
If **true**, the STR (Security Token Reference) Transform will be used when (externally) signing a SAML Token; otherwise the STR (Security Token Reference) Transform will not be used. Some frameworks cannot process the **SecurityTokenReference**. You may set this **false** in such cases. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USE_STR_TRANSFORM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.add.inclusive.prefixes	**boolean**	**true**
If **true**, an **InclusiveNamespaces PrefixList** will be added as a **CanonicalizationMethod** child when generating Signatures using **WSConstants.C14N_EXCL_OMIT_COMMENTS**; otherwise the **PrefixList** will not be added. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ADD_INCLUSIVE_PREFIXES** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.disable.require.client.cert.check	**boolean**	**false**
If **true**, the enforcement of the WS-SecurityPolicy **RequireClientCertificate** policy will be disabled; otherwise the enforcement of the WS-SecurityPolicy **RequireClientCertificate** policy is enabled. Some servers may not do client certificate verification at the start of the SSL handshake, and therefore the client certificates may not be available to the WS-Security layer for policy verification. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_DISABLE_REQUIRE_CLIENT_CERT_CHECK** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.expand.xop.include	**boolean**	

Configuration property	Type	Default
If **true**, the **xop:Include** elements will be searched for encryption and signature (on the outbound side) or for signature verification (on the inbound side); otherwise the search won't happen. This ensures that the actual bytes are signed, and not just the reference. The default is **true** if MTOM is enabled, otherwise the default is **false**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_EXPAND_XOP_INCLUDE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.timestamp.timeToLive	**string**	**300**
The time in seconds to add to the Creation value of an incoming **Timestamp** to determine whether to accept it as valid or not. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TIMESTAMP_TIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.timestamp.futureTimeToLive	**string**	**60**
The time in seconds in the future within which the **Created** time of an incoming **Timestamp** is valid. The default is greater than zero to avoid problems where clocks are slightly askew. Set this to **0** to reject all future-created `Timestamp`s. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TIMESTAMP_FUTURETIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.usernametoken.timeToLive	**string**	**300**
The time in seconds to append to the Creation value of an incoming **UsernameToken** to determine whether to accept it as valid or not. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAMETOKEN_TIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.usernametoken.futureTimeToLive	**string**	**60**

Configuration property	Type	Default
The time in seconds in the future within which the **Created** time of an incoming **UsernameToken** is valid. The default is greater than zero to avoid problems where clocks are slightly askew. Set this to **0** to reject all future-created `UsernameToken`s. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_USERNAMETOKEN_FUTURETIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.spnego.client.action	**string**	
A reference to a **org.apache.wss4j.common.spnego.SpnegoClientAction** bean to use for SPNEGO. This allows the user to plug in a different implementation to obtain a service ticket. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SPNEGO_CLIENT_ACTION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.nonce.cache.instance	**string**	
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache **UsernameToken** nonces. A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_NONCE_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.timestamp.cache.instance	**string**	
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache **Timestamp Created** Strings. A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TIMESTAMP_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.saml.cache.instance	**string**	

Configuration property	Type	Default
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache SAML2 Token Identifier Strings (if the token contains a **OneTimeUse** condition). A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SAML_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.cache.config.file	**string**	
Set this property to point to a configuration file for the underlying caching implementation for the **TokenStore**. The default configuration file that is used is **cxf-ehcache.xml** in **org.apache.cxf:cxf-rt-security** JAR. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CACHE_CONFIG_FILE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.token-store-cache-instance	**string**	
A reference to a **org.apache.cxf.ws.security.tokenstore.TokenStore** bean to use for caching security tokens. By default this uses a instance. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_TOKEN_STORE_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.cache.identifier	**string**	
The Cache Identifier to use with the TokenStore. CXF uses the following key to retrieve a token store: **org.apache.cxf.ws.security.tokenstore.TokenStore-<identifier>**. This key can be used to configure service-specific cache configuration. If the identifier does not match, then it falls back to a cache configuration with key **org.apache.cxf.ws.security.tokenstore.TokenStore**. The default **<identifier>** is the QName of the service in question. However to pick up a custom cache configuration (for example, if you want to specify a TokenStore per-client proxy), it can be configured with this identifier instead. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_CACHE_IDENTIFIER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.role.classifier	**string**	

Configuration property	Type	Default
The Subject Role Classifier to use. If one of the WSS4J Validators returns a JAAS Subject from Validation, then the **WSS4JInInterceptor** will attempt to create a **SecurityContext** based on this Subject. If this value is not specified, then it tries to get roles using the **DefaultSecurityContext** in **org.apache.cxf:cxf-core**. Otherwise it uses this value in combination with the **role.classifier.type** to get the roles from the **Subject**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ROLE_CLASSIFIER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.role.classifier.type	string	prefix
The Subject Role Classifier Type to use. If one of the WSS4J Validators returns a JAAS Subject from Validation, then the **WSS4JInInterceptor** will attempt to create a **SecurityContext** based on this Subject. Currently accepted values are **prefix** or **classname**. Must be used in conjunction with the **role.classifier**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ROLE_CLASSIFIER_TYPE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.asymmetric.signature.algorithm	string	
This configuration tag allows the user to override the default Asymmetric Signature algorithm (RSA-SHA1) for use in WS-SecurityPolicy, as the WS-SecurityPolicy specification does not allow the use of other algorithms at present. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_ASYMMETRIC_SIGNATURE_ALGORITHM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.symmetric.signature.algorithm	string	
This configuration tag allows the user to override the default Symmetric Signature algorithm (HMAC-SHA1) for use in WS-SecurityPolicy, as the WS-SecurityPolicy specification does not allow the use of other algorithms at present. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SYMMETRIC_SIGNATURE_ALGORITHM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.password.encryptor.instance	string	

Configuration property	Type	Default
A reference to a **org.apache.wss4j.common.crypto.PasswordEncryptor** bean, which is used to encrypt or decrypt passwords in the Merlin Crypto implementation (or any custom Crypto implementations). By default, WSS4J uses the **org.apache.wss4j.common.crypto.JasyptPasswordEncryptor** which must be instantiated with a password to use to decrypt keystore passwords in the Merlin Crypto definition. This password is obtained via the CallbackHandler defined via **callback-handler** The encrypted passwords must be stored in the format "ENC(encoded encrypted password)". This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_PASSWORD_ENCRYPTOR_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.delegated.credential	**string**	
A reference to a Kerberos **org.ietf.jgss.GSSCredential** bean to use for WS-Security. This is used to retrieve a service ticket instead of using the client credentials. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_DELEGATED_CREDENTIAL** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.security.context.creator	**string**	
A reference to a **org.apache.cxf.ws.security.wss4j.WSS4JSecurityContextCreator** bean that is used to create a CXF SecurityContext from the set of WSS4J processing results. The default implementation is **org.apache.cxf.ws.security.wss4j.DefaultWSS4JSecurityContextCreator**. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SECURITY_CONTEXT_CREATOR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.security.token.lifetime	**long**	**300000**
The security token lifetime value (in milliseconds). This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_SECURITY_TOKEN_LIFETIME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.request.credential.delegation	**boolean**	**false**

Configuration property	Type	Default
If **true**, credential delegation is requested in the KerberosClient; otherwise the credential delegation is not in the KerberosClient. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_REQUEST_CREDENTIAL_DELEGATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.use.credential.delegation	**boolean**	**false**
If **true**, GSSCredential bean is retrieved from the Message Context using the **delegated.credential** property and then it is used to obtain a service ticket. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_USE_CREDENTIAL_DELEGATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.is.username.in.servicename.form	**boolean**	**false**
If **true**, the Kerberos username is in servicename form; otherwise the Kerberos username is not in servicename form. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_IS_USERNAME_IN_SERVICENAME_FORM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.jaas.context	**string**	
The JAAS Context name to use for Kerberos. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_JAAS_CONTEXT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.spn	**string**	

Configuration property	Type	Default
The Kerberos Service Provider Name (spn) to use. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_SPN** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.kerberos.client	**string**	
A reference to a **org.apache.cxf.ws.security.kerberos.KerberosClient** bean used to obtain a service ticket. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_KERBEROS_CLIENT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.client."clients".security.sts.client	**string**	
A reference to a fully configured **org.apache.cxf.ws.security.trust.STSClient** bean to communicate with the STS. If not set, the STS client will be created and configured based on other **[prefix].security.sts.client.*** properties as long as they are available. To work around the fact that **org.apache.cxf.ws.security.trust.STSClient** does not have a no-args constructor and cannot thus be used as a CDI bean type, you can use the wrapper class **io.quarkiverse.cxf.ws.security.sts.client.STSClientBean** instead. **TIP** Check the Security Token Service (STS) extension page for more information about WS-Trust. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.wsdl	**string**	
A URL, resource path or local filesystem path pointing to a WSDL document to use when generating the service proxy of the STS client. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_WSDL** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.service-name	**string**	

Configuration property	Type	Default
A fully qualified name of the STS service. Common values include: • WS-Trust 1.0: **{http://schemas.xmlsoap.org/ws/2005/02/trust/}SecurityTokenService** • WS-Trust 1.3: **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}SecurityTokenService** • WS-Trust 1.4: **{http://docs.oasis-open.org/ws-sx/ws-trust/200802/}SecurityTokenService** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_SERVICE_NAME** Since Quarkus CXF: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.endpoint-name	**string**	
A fully qualified name of the STS endpoint name. Common values include: • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}X509_Port** • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}Transport_Port** • **{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}UT_Port** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENDPOINT_NAME** Since Quarkus CXF: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.username	**string**	
The user name to use when authenticating against the STS. It is used as follows: • As the name in the UsernameToken for WS-Security • As the alias name in the keystore to get the user's cert and private key for signature if **signature.username** is not set • As the alias name in the keystore to get the user's public key for encryption if **encryption.username** is not set Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_USERNAME** Since Quarkus CXF: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.password	**string**	
The password associated with the **username**. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_PASSWORD** Since Quarkus CXF: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.username	**string**	

Configuration property	Type	Default
The user's name for encryption. It is used as the alias name in the keystore to get the user's public key for encryption. If this is not defined, then **username** is used instead. If that is also not specified, it uses the default alias set in the properties file referenced by **encrypt.properties**. If that's also not set, and the keystore only contains a single key, that key will be used. For the WS-Security web service provider, the **useReqSigCert** value can be used to accept (encrypt to) any client whose public key is in the service's truststore (defined in **encrypt.properties**). **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_USERNAME** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.properties	**Map<String,String>**	
The Crypto property configuration to use for encryption, if **encryption.crypto** is not set. Example [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_PROPERTIES** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.encryption.crypto	**string**	
A reference to a **org.apache.wss4j.common.crypto.Crypto** to be used for encryption. If not set, **encryption.properties** will be used to configure a**Crypto** instance. **Environment variable**: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_ENCRYPTION_CRYPTO** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.crypto	**string**	

Configuration property	Type	Default
A reference to a **org.apache.wss4j.common.crypto.Crypto** to be used for the STS. If not set, **token.properties** will be used to configure a**Crypto** instance. WCF's trust server sometimes will encrypt the token in the response IN ADDITION TO the full security on the message. These properties control the way the STS client will decrypt the EncryptedData elements in the response. These are also used by the **token.properties** to send/process any RSA/DSAKeyValue tokens used if the KeyType is **PublicKey** Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_CRYPTO** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.properties	**Map<String,String>**	
The Crypto property configuration to use for encryption, if **encryption.crypto** is not set. Example [prefix].token.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].token.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].token.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_PROPERTIES** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.username	**string**	
The alias name in the keystore to get the user's public key to send to the STS for the PublicKey KeyType case. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_USERNAME** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.token.usecert	**boolean**	**false**

Configuration property	Type	Default
Whether to write out an X509Certificate structure in UseKey/KeyInfo, or whether to write out a KeyValue structure. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_TOKEN_USECERT** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.client."clients".security.sts.client.soap12-binding	**boolean**	**false**
If **true** the STS client will be set to send Soap 1.2 messages; otherwise it will send SOAP 1.1 messages. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__SECURITY_STS_CLIENT_SOAP12_BINDING** **Since Quarkus CXF**: 3.8.0		
quarkus.cxf.endpoint."endpoints".security.username	**string**	
The user's name. It is used as follows: • As the name in the UsernameToken for WS-Security • As the alias name in the keystore to get the user's cert and private key for signature if **signature.username** is not set • As the alias name in the keystore to get the user's public key for encryption if **encryption.username** is not set Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.password	**string**	
The user's password when a **callback-handler** is not defined. This is only used for the password in a WS-Security UsernameToken. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_PASSWORD** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.username	**string**	
The user's name for signature. It is used as the alias name in the keystore to get the user's cert and private key for signature. If this is not defined, then **username** is used instead. If that is also not specified, it uses the the default alias set in the properties file referenced by **signature.properties**. If that's also not set, and the keystore only contains a single key, that key will be used. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_USERNAME** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.signature.password	string	
The user's password for signature when a **callback-handler** is not defined. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_PASSWORD** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.encryption.username	string	
The user's name for encryption. It is used as the alias name in the keystore to get the user's public key for encryption. If this is not defined, then **username** is used instead. If that is also not specified, it uses the default alias set in the properties file referenced by **encrypt.properties**. If that's also not set, and the keystore only contains a single key, that key will be used. For the WS-Security web service provider, the **useReqSigCert** value can be used to accept (encrypt to) any client whose public key is in the service's truststore (defined in **encrypt.properties**). Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_USERNAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.callback-handler	string	
A reference to a **javax.security.auth.callback.CallbackHandler** bean used to obtain passwords, for both outbound and inbound requests. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CALLBACK_HANDLER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.saml-callback-handler	string	
A reference to a **javax.security.auth.callback.CallbackHandler** implementation used to construct SAML Assertions. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_CALLBACK_HANDLER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.properties	Map<String,String>	

Configuration property	Type	Default
The Crypto property configuration to use for signing, if **signature.crypto** is not set. Example [prefix].signature.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].signature.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_PROPERTIES** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.encryption.properties	**Map<String,String>**	
The Crypto property configuration to use for encryption, if **encryption.crypto** is not set. Example [prefix].encryption.properties."org.apache.ws.security.crypto.provider" = org.apache.ws.security.components.crypto.Merlin [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" = password [prefix].encryption.properties."org.apache.ws.security.crypto.merlin.file" = certs/alice.jks **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_PROPERTIES** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.signature.crypto	**string**	
A reference to a **org.apache.wss4j.common.crypto.Crypto** bean to be used for signature. If not set, **signature.properties** will be used to configure a**Crypto** instance. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SIGNATURE_CRYPTO** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.encryption.crypto	**string**	

Configuration property	Type	Default
A reference to a **org.apache.wss4j.common.crypto.Crypto** to be used for encryption. If not set, **encryption.properties** will be used to configure a **Crypto** instance. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_CRYPTO** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.encryption.certificate	**string**	
A message property for prepared X509 certificate to be used for encryption. If this is not defined, then the certificate will be either loaded from the keystore **encryption.properties** or extracted from request (when WS-Security is used and if **encryption.username** has value **useReqSigCert**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENCRYPTION_CERTIFICATE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable-revocation	**boolean**	**false**
If **true**, Certificate Revocation List (CRL) checking is enabled when verifying trust in a certificate; otherwise it is not enabled. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_REVOCATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable-unsigned-saml-assertion-principal	**boolean**	**false**
If **true**, unsigned SAML assertions will be allowed as SecurityContext Principals; otherwise they won't be allowed as SecurityContext Principals.  SIGNATURE The label "unsigned" refers to an internal signature. Even if the token is signed by an external signature (as per the "sender-vouches" requirement), this boolean must still be configured if you want to use the token to set up the security context. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_UNSIGNED_SAML_ASSERTION_PRINCIPAL** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.validate-saml-subject-confirmation	boolean	true
If **true**, the **SubjectConfirmation** requirements of a received SAML Token (sender-vouches or holder-of-key) will be validated; otherwise they won't be validated. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_VALIDATE_SAML_SUBJECT_CONFIRMATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.sc-from-jaas-subject	boolean	true
If **true**, security context can be created from JAAS Subject; otherwise it must not be created from JAAS Subject. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SC_FROM_JAAS_SUBJECT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.audience-restriction-validation	boolean	true
If **true**, then if the SAML Token contains Audience Restriction URIs, one of them must match one of the values in **audience.restrictions**; otherwise the SAML AudienceRestriction validation is disabled. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_AUDIENCE_RESTRICTION_VALIDATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.saml-role-attributename	string	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role

Configuration property	Type	Default
The attribute URI of the SAML **AttributeStatement** where the role information is stored. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_ROLE_ATTRIBUTENAME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.subject-cert-constraints	**string**	
A String of regular expressions (separated by the value specified in **security.cert.constraints.separator**) which will be applied to the subject DN of the certificate used for signature validation, after trust verification of the certificate chain associated with the certificate. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SUBJECT_CERT_CONSTRAINTS** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.cert-constraints-separator	**string**	**,**
The separator that is used to parse certificate constraints configured in **security.subject.cert.constraints** This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CERT_CONSTRAINTS_SEPARATOR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.actor	**string**	
The actor or role name of the **wsse:Security** header. If this parameter is omitted, the actor name is not set. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ACTOR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.validate.token	**boolean**	**true**
If **true**, the password of a received **UsernameToken** will be validated; otherwise it won't be validated. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_VALIDATE_TOKEN** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.username-token.always.encrypted	**boolean**	**true**

Configuration property	Type	Default
Whether to always encrypt **UsernameTokens** that are defined as a**SupportingToken**. This should not be set to **false** in a production environment, as it exposes the password (or the digest of the password) on the wire. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAME_TOKEN_ALWAYS_ENCRYPTED** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.is-bsp-compliant	**boolean**	**true**
If **true**, the compliance with the Basic Security Profile (BSP) 1.1 will be ensured; otherwise it will not be ensured. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_IS_BSP_COMPLIANT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable.nonce.cache	**boolean**	
If **true**, the **UsernameToken** nonces will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators.  CACHING Caching only applies when either a **UsernameToken** WS-SecurityPolicy is in effect, or the **UsernameToken** action has been configured for the non-security-policy case. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_NONCE_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable.timestamp.cache	**boolean**	

Configuration property	Type	Default
If **true**, the **Timestamp Created** Strings (these are only cached in conjunction with a message Signature) will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators.  **CACHING** Caching only applies when either a **IncludeTimestamp** policy is in effect, or the **Timestamp** action has been configured for the non-security-policy case. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_TIMESTAMP_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.enable.streaming	**boolean**	**false**
If **true**, the new streaming (StAX) implementation of WS-Security is used; otherwise the old DOM implementation is used. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_STREAMING** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.return.security.error	**boolean**	**false**
If **true**, detailed security error messages are sent to clients; otherwise the details are omitted and only a generic error message is sent. The "real" security errors should not be returned to the client in production, as they may leak information about the deployment, or otherwise provide an "oracle" for attacks. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_RETURN_SECURITY_ERROR** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.must-understand	**boolean**	**true**
If **true**, the SOAP **mustUnderstand** header is included in security headers based on a WS-SecurityPolicy; otherwise the header is always omitted. Works only with **enable.streaming = true** - see CXF-8940 **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_MUST_UNDERSTAND** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.enable.saml.cache	**boolean**	
If **true** and in case the token contains a **OneTimeUse** Condition, the SAML2 Token Identifiers will be cached for both message initiators and recipients; otherwise they won't be cached for neither message initiators nor recipients. The default is **true** for message recipients, and **false** for message initiators. Caching only applies when either a **SamlToken** policy is in effect, or a SAML action has been configured for the non-security-policy case. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ENABLE_SAML_CACHE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.store.bytes.in.attachment	**boolean**	
Whether to store bytes (CipherData or BinarySecurityToken) in an attachment. The default is true if MTOM is enabled. Set it to false to BASE-64 encode the bytes and "inlined" them in the message instead. Setting this to true is more efficient, as it means that the BASE-64 encoding step can be skipped. This only applies to the DOM WS-Security stack. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_STORE_BYTES_IN_ATTACHMENT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.swa.encryption.attachment.transform.content	**boolean**	**false**
If **true**, **Attachment-Content-Only** transform will be used when an Attachment is encrypted via a WS-SecurityPolicy expression; otherwise **Attachment-Complete** transform will be used when an Attachment is encrypted via a WS-SecurityPolicy expression. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SWA_ENCRYPTION_ATTACHMENT_TRANSFORM_CONTENT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.use.str.transform	**boolean**	**true**

Configuration property	Type	Default
If **true**, the STR (Security Token Reference) Transform will be used when (externally) signing a SAML Token; otherwise the STR (Security Token Reference) Transform will not be used. Some frameworks cannot process the **SecurityTokenReference**. You may set this **false** in such cases. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USE_STR_TRANSFORM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.add.inclusive.prefixes	**boolean**	**true**
If **true**, an **InclusiveNamespaces PrefixList** will be added as a **CanonicalizationMethod** child when generating Signatures using **WSConstants.C14N_EXCL_OMIT_COMMENTS**; otherwise the **PrefixList** will not be added. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ADD_INCLUSIVE_PREFIXES** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.disable.require.client.cert.check	**boolean**	**false**
If **true**, the enforcement of the WS-SecurityPolicy **RequireClientCertificate** policy will be disabled; otherwise the enforcement of the WS-SecurityPolicy **RequireClientCertificate** policy is enabled. Some servers may not do client certificate verification at the start of the SSL handshake, and therefore the client certificates may not be available to the WS-Security layer for policy verification. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_DISABLE_REQUIRE_CLIENT_CERT_CHECK** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.expand.xop.include	**boolean**	

Configuration property	Type	Default
If **true**, the **xop:Include** elements will be searched for encryption and signature (on the outbound side) or for signature verification (on the inbound side); otherwise the search won't happen. This ensures that the actual bytes are signed, and not just the reference. The default is **true** if MTOM is enabled, otherwise the default is **false**. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_EXPAND_XOP_INCLUDE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.timestamp.timeToLive	**string**	**300**
The time in seconds to add to the Creation value of an incoming **Timestamp** to determine whether to accept it as valid or not. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_TIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.timestamp.futureTimeToLive	**string**	**60**
The time in seconds in the future within which the **Created** time of an incoming **Timestamp** is valid. The default is greater than zero to avoid problems where clocks are slightly askew. Set this to **0** to reject all future-created `Timestamp`s. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_FUTURETIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.usernametoken.timeToLive	**string**	**300**
The time in seconds to append to the Creation value of an incoming **UsernameToken** to determine whether to accept it as valid or not. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAMETOKEN_TIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.usernametoken.futureTimeToLive	**string**	**60**

Configuration property	Type	Default
The time in seconds in the future within which the **Created** time of an incoming **UsernameToken** is valid. The default is greater than zero to avoid problems where clocks are slightly askew. Set this to **0** to reject all future-created `UsernameToken`s. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_USERNAMETOKEN_FUTURETIMETOLIVE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.spnego.client.action	**string**	
A reference to a **org.apache.wss4j.common.spnego.SpnegoClientAction** bean to use for SPNEGO. This allows the user to plug in a different implementation to obtain a service ticket. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SPNEGO_CLIENT_ACTION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.nonce.cache.instance	**string**	
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache **UsernameToken** nonces. A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_NONCE_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.timestamp.cache.instance	**string**	
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache **Timestamp Created** Strings. A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TIMESTAMP_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.saml.cache.instance	**string**	

Configuration property	Type	Default
A reference to a **org.apache.wss4j.common.cache.ReplayCache** bean used to cache SAML2 Token Identifier Strings (if the token contains a **OneTimeUse** condition). A **org.apache.wss4j.common.cache.EHCacheReplayCache** instance is used by default. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SAML_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.cache.config.file	**string**	
Set this property to point to a configuration file for the underlying caching implementation for the **TokenStore**. The default configuration file that is used is **cxf-ehcache.xml** in **org.apache.cxf:cxf-rt-security** JAR. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CACHE_CONFIG_FILE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.token-store-cache-instance	**string**	
A reference to a **org.apache.cxf.ws.security.tokenstore.TokenStore** bean to use for caching security tokens. By default this uses a instance. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_TOKEN_STORE_CACHE_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.cache.identifier	**string**	
The Cache Identifier to use with the TokenStore. CXF uses the following key to retrieve a token store: **org.apache.cxf.ws.security.tokenstore.TokenStore-<identifier>**. This key can be used to configure service-specific cache configuration. If the identifier does not match, then it falls back to a cache configuration with key **org.apache.cxf.ws.security.tokenstore.TokenStore**. The default **<identifier>** is the QName of the service in question. However to pick up a custom cache configuration (for example, if you want to specify a TokenStore per-client proxy), it can be configured with this identifier instead. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_CACHE_IDENTIFIER** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.role.classifier	**string**	
The Subject Role Classifier to use. If one of the WSS4J Validators returns a JAAS Subject from Validation, then the **WSS4JInInterceptor** will attempt to create a **SecurityContext** based on this Subject. If this value is not specified, then it tries to get roles using the **DefaultSecurityContext** in **org.apache.cxf:cxf-core**. Otherwise it uses this value in combination with the **role.classifier.type** to get the roles from the **Subject**. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ROLE_CLASSIFIER** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.role.classifier.type	**string**	**prefix**
The Subject Role Classifier Type to use. If one of the WSS4J Validators returns a JAAS Subject from Validation, then the **WSS4JInInterceptor** will attempt to create a **SecurityContext** based on this Subject. Currently accepted values are **prefix** or **classname**. Must be used in conjunction with the **role.classifier**. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ROLE_CLASSIFIER_TYPE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.asymmetric.signature.algorithm	**string**	
This configuration tag allows the user to override the default Asymmetric Signature algorithm (RSA-SHA1) for use in WS-SecurityPolicy, as the WS-SecurityPolicy specification does not allow the use of other algorithms at present. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_ASYMMETRIC_SIGNATURE_ALGORITHM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.symmetric.signature.algorithm	**string**	

Configuration property	Type	Default
This configuration tag allows the user to override the default Symmetric Signature algorithm (HMAC-SHA1) for use in WS-SecurityPolicy, as the WS-SecurityPolicy specification does not allow the use of other algorithms at present. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SYMMETRIC_SIGNATURE_ALGORITHM** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.password.encryptor.instance	**string**	
A reference to a **org.apache.wss4j.common.crypto.PasswordEncryptor** bean, which is used to encrypt or decrypt passwords in the Merlin Crypto implementation (or any custom Crypto implementations). By default, WSS4J uses the **org.apache.wss4j.common.crypto.JasyptPasswordEncryptor** which must be instantiated with a password to use to decrypt keystore passwords in the Merlin Crypto definition. This password is obtained via the CallbackHandler defined via **callback-handler** The encrypted passwords must be stored in the format "ENC(encoded encrypted password)". This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_PASSWORD_ENCRYPTOR_INSTANCE** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.delegated.credential	**string**	
A reference to a Kerberos **org.ietf.jgss.GSSCredential** bean to use for WS-Security. This is used to retrieve a service ticket instead of using the client credentials. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_DELEGATED_CREDENTIAL** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.security.context.creator	**string**	
A reference to a **org.apache.cxf.ws.security.wss4j.WSS4JSecurityContextCreator** bean that is used to create a CXF SecurityContext from the set of WSS4J processing results. The default implementation is **org.apache.cxf.ws.security.wss4j.DefaultWSS4JSecurityContextCreator**. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SECURITY_CONTEXT_CREATOR** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.security.token.lifetime	**long**	**300000**
The security token lifetime value (in milliseconds). This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_SECURITY_TOKEN_LIFETIME** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.request.credential.delegation	**boolean**	**false**
If **true**, credential delegation is requested in the KerberosClient; otherwise the credential delegation is not in the KerberosClient. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_REQUEST_CREDENTIAL_DELEGATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.use.credential.delegation	**boolean**	**false**
If **true**, GSSCredential bean is retrieved from the Message Context using the **delegated.credential** property and then it is used to obtain a service ticket. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_USE_CREDENTIAL_DELEGATION** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.is.username.in.servicename.form	**boolean**	**false**
If **true**, the Kerberos username is in servicename form; otherwise the Kerberos username is not in servicename form. This option is experimental, because it is not covered by tests yet. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_IS_USERNAME_IN_SERVICENAME_FORM** **Since Quarkus CXF**: 2.5.0		

Configuration property	Type	Default
quarkus.cxf.endpoint."endpoints".security.kerberos.jaas.context	**string**	
The JAAS Context name to use for Kerberos. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_JAAS_CONTEXT** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.spn	**string**	
The Kerberos Service Provider Name (spn) to use. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_SPN** **Since Quarkus CXF**: 2.5.0		
quarkus.cxf.endpoint."endpoints".security.kerberos.client	**string**	
A reference to a **org.apache.cxf.ws.security.kerberos.KerberosClient** bean used to obtain a service ticket. This option is experimental, because it is not covered by tests yet. **Environment variable**: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__SECURITY_KERBEROS_CLIENT** **Since Quarkus CXF**: 2.5.0		

5.5. WS-RELIABLEMESSAGING

WS-ReliableMessaging (WS-RM) is a protocol ensuring a reliable delivery of messages in a distributed environment even in presence of software, system, or network failures.

This extension provides CXF framework's WS-ReliableMessaging implementation.

5.5.1. Maven coordinates

Create a new project using **quarkus-cxf-rt-ws-rm** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-ws-rm</artifactId>
</dependency>
```

5.5.2. Supported standards

- WS-ReliableMessaging

5.5.3. Usage

Once your application depends on **quarkus-cxf-rt-ws-rm**, WS-RM is enabled for all clients and service endpoints defined in **application.properties**. This is due to the fact that the **quarkus.cxf.client.myClient.rm.enabled** and **quarkus.cxf.endpoint."/my-endpoint".rm.enabled** properties are **true** by default.

Enabling WS-RM for a client or service endpoints means that WS-RM interceptors will be added to the given client or endpoint.

In addition to that you may want to set some of the configuration options or the following WS-Addressing options:

- **quarkus.cxf.client.myClient.decoupled-endpoint**
- **quarkus.cxf.decoupled-endpoint-base**

5.5.3.1. Runnable example

There is an integration test covering WS-RM with a decoupled endpoint in the Quarkus CXF source tree.

It is split into two separate applications that communicate with each other:

- Server
- Client

To run it, you need to install the server into your local Maven repository first

```
$ cd test-util-parent/test-ws-rm-server-jvm
$ mvn clean install
```

And then you can run the test scenario implemented in the client module:

```
$ cd ../../integration-tests/ws-rm-client
$ mvn clean test
```

The exchange of SOAP messages between the client, the server and the decoupled endpoint are shown in the console.

5.5.4. Configuration

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

Configuration property	Type	Default

Configuration property	Type	Default
quarkus.cxf.rm.namespace	string	http://schemas.xmlsoap.org/ws/2005/02/rm
WS-RM version namespace: http://schemas.xmlsoap.org/ws/2005/02/rm/ or http://docs.oasis-open.org/ws-rx/wsrm/200702 Environment variable: QUARKUS_CXF_RM_NAMESPACE Since Quarkus CXF: 2.7.0		
quarkus.cxf.rm.wsa-namespace	string	http://schemas.xmlsoap.org/ws/2004/08/addressing
WS-Addressing version namespace: http://schemas.xmlsoap.org/ws/2004/08/addressing or http://www.w3.org/2005/08/addressing. Note that this property is ignored unless you are using the http://schemas.xmlsoap.org/ws/2005/02/rm/ RM namespace. Environment variable: QUARKUS_CXF_RM_WSA_NAMESPACE Since Quarkus CXF: 2.7.0		
quarkus.cxf.rm.inactivity-timeout	long	
A time duration in milliseconds after which the associated sequence will be closed if no messages (including acknowledgments and other control messages) were exchanged between the sender and receiver during that period of time. If not set, the associated sequence will never be closed due to inactivity. Environment variable: QUARKUS_CXF_RM_INACTIVITY_TIMEOUT Since Quarkus CXF: 2.7.0		
quarkus.cxf.rm.retransmission-interval	long	3000

Configuration property	Type	Default
A time duration in milliseconds between successive attempts to resend a message that has not been acknowledged by the receiver. Environment variable: **QUARKUS_CXF_RM_RETRANSMISSION_INTERVAL** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.exponential-backoff	**boolean**	**false**
If **true** the retransmission interval will be doubled on every transmission attempt; otherwise the retransmission interval stays equal to **quarkus.cxf.rm.retransmission-interval** for every retransmission attempt. Environment variable: **QUARKUS_CXF_RM_EXPONENTIAL_BACKOFF** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.acknowledgement-interval	**long**	
A time duration in milliseconds within which an acknowledgement for a received message is expected to be sent by a RM destination. If not specified, the acknowledgements will be sent immediately. Environment variable: **QUARKUS_CXF_RM_ACKNOWLEDGEMENT_INTERVAL** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.store	**string**	
A reference to a **org.apache.cxf.ws.rm.persistence.RMStore** bean used to store source and destination sequences and message references. Environment variable: **QUARKUS_CXF_RM_STORE** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.rm.feature-ref	**string**	**#defaultRmFeature**
A reference to a **org.apache.cxf.ws.rm.feature.RMFeature** bean to set on clients and service endpoint which have **quarkus.cxf.[client|service]."name".rm.enabled = true**. If the value is **#defaultRmFeature** then Quarkus CXF creates and configures the bean for you. Environment variable: **QUARKUS_CXF_RM_FEATURE_REF** **Since Quarkus CXF**: 2.7.0		
quarkus.cxf.client."clients".rm.enabled	**boolean**	**true**

Configuration property	Type	Default
If **true** then the WS-ReliableMessaging interceptors will be added to this client or service endpoint. Environment variable: **QUARKUS_CXF_CLIENT__CLIENTS__RM_ENABLED**		
quarkus.cxf.endpoint."endpoints".rm.enabled	**boolean**	**true**
If **true** then the WS-ReliableMessaging interceptors will be added to this client or service endpoint. Environment variable: **QUARKUS_CXF_ENDPOINT__ENDPOINTS__RM_ENABLED**		

5.6. SECURITY TOKEN SERVICE (STS)

Issue, renew and validate security tokens in context of WS-Trust.

5.6.1. Maven coordinates

Create a new project using **quarkus-cxf-services-sts** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-services-sts</artifactId>
</dependency>
```

5.6.2. Supported standards

- WS-Trust

5.6.3. Usage

Here are the key parts of a basic WS-Trust scenario:

- **WS-SecurityPolicy** - except for defining security requirements, such as transport protocols, encryption and signing, it can also contain an **<IssuedToken>** assertion. It specifies the requirements and constraints for these security tokens that the client must adhere to when accessing the service.
- **Security Token Service (STS)** - issues, validates, and renews security tokens upon request. It acts as a trusted authority that authenticates clients and issues tokens that assert the client's identity and permissions.
- **Client** - requests a token from the STS to access a web service. It must authenticate itself to the STS and provide details about the kind of token required.
- **Service** - relies on the STS to authenticate clients and validate their tokens.

5.6.3.1. Runnable example

There is an integration test covering WS-Trust in the Quarkus CXF source tree. Let's walk through it and see how the individual parts are set to work together.

5.6.3.1.1. WS-SecurityPolicy

The policy is located in **asymmetric-saml2-policy.xml** file. Its key part is the **<IssuedToken>** assertion requiring a SAML 2.0 token:

asymmetric-saml2-policy.xml

```
                    <sp:IssuedToken
                       sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                       <sp:RequestSecurityTokenTemplate>
                          <t:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0</t:TokenType>
                          <t:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/PublicKey</t:KeyType>
                       </sp:RequestSecurityTokenTemplate>
                       <wsp:Policy>
                          <sp:RequireInternalReference />
                       </wsp:Policy>
                       <sp:Issuer>
                          <wsaws:Address>http://localhost:8081/services/sts</wsaws:Address>
                          <wsaws:Metadata xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
                                          wsdli:wsdlLocation="http://localhost:8081/services/sts?wsdl">
                             <wsaw:ServiceName
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                                               xmlns:stsns="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"

EndpointName="UT_Port">stsns:SecurityTokenService</wsaw:ServiceName>
                          </wsaws:Metadata>
                       </sp:Issuer>
                    </sp:IssuedToken>
```

5.6.3.1.2. Security Token Service (STS)

The STS is implemented in **Sts.java**:

Sts.java

```
@WebServiceProvider(serviceName = "SecurityTokenService", portName = "UT_Port",
targetNamespace = "http://docs.oasis-open.org/ws-sx/ws-trust/200512/", wsdlLocation = "ws-trust-
1.4-service.wsdl")
public class Sts extends SecurityTokenServiceProvider {

   public Sts() throws Exception {
      super();

      StaticSTSProperties props = new StaticSTSProperties();
      props.setSignatureCryptoProperties("stsKeystore.properties");
      props.setSignatureUsername("sts");
      props.setCallbackHandlerClass(StsCallbackHandler.class.getName());
      props.setIssuer("SampleSTSIssuer");
```

```
        List<ServiceMBean> services = new LinkedList<ServiceMBean>();
        StaticService service = new StaticService();
        final Config config = ConfigProvider.getConfig();
        final int port = LaunchMode.current().equals(LaunchMode.TEST) ?
config.getValue("quarkus.http.test-port", Integer.class)
                : config.getValue("quarkus.http.port", Integer.class);
        service.setEndpoints(Arrays.asList(
                "http://localhost:" + port + "/services/hello-ws-trust",
                "http://localhost:" + port + "/services/hello-ws-trust-actas",
                "http://localhost:" + port + "/services/hello-ws-trust-onbehalfof"));
        services.add(service);

        TokenIssueOperation issueOperation = new TokenIssueOperation();
        issueOperation.setServices(services);
        issueOperation.getTokenProviders().add(new SAMLTokenProvider());
        // required for OnBehalfOf
        issueOperation.getTokenValidators().add(new UsernameTokenValidator());
        // added for OnBehalfOf and ActAs
        issueOperation.getDelegationHandlers().add(new UsernameTokenDelegationHandler());
        issueOperation.setStsProperties(props);

        TokenValidateOperation validateOperation = new TokenValidateOperation();
        validateOperation.getTokenValidators().add(new SAMLTokenValidator());
        validateOperation.setStsProperties(props);

        this.setIssueOperation(issueOperation);
        this.setValidateOperation(validateOperation);
    }
}
```

and configured in **application.properties**:

application.properties

```
quarkus.cxf.endpoint."/sts".implementor = io.quarkiverse.cxf.it.ws.trust.sts.Sts
quarkus.cxf.endpoint."/sts".logging.enabled = pretty

quarkus.cxf.endpoint."/sts".security.signature.username = sts
quarkus.cxf.endpoint."/sts".security.signature.password = password
quarkus.cxf.endpoint."/sts".security.validate.token = false

quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.provider" =
org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.t
ype" = pkcs12
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.
password" = password
quarkus.cxf.endpoint."/sts".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.f
le" = sts.pkcs12
```

5.6.3.1.3. Service

The service is implemented in **TrustHelloServiceImpl.java**:

TrustHelloServiceImpl.java

```
@WebService(portName = "TrustHelloServicePort", serviceName = "TrustHelloService",
targetNamespace = "https://quarkiverse.github.io/quarkiverse-docs/quarkus-cxf/test/ws-trust",
endpointInterface = "io.quarkiverse.cxf.it.ws.trust.server.TrustHelloService")
public class TrustHelloServiceImpl implements TrustHelloService {
    @WebMethod
    @Override
    public String hello(String person) {
        return "Hello " + person + "!";
    }
}
```

The **asymmetric-saml2-policy.xml** mentioned above is set in the Service Endpoint Interface **TrustHelloService.java**:

TrustHelloServiceImpl.java

```
@WebService(targetNamespace = "https://quarkiverse.github.io/quarkiverse-docs/quarkus-
cxf/test/ws-trust")
@Policy(placement = Policy.Placement.BINDING, uri = "classpath:/asymmetric-saml2-policy.xml")
public interface TrustHelloService {
    @WebMethod
    @Policies({
            @Policy(placement = Policy.Placement.BINDING_OPERATION_INPUT, uri = "classpath:/io-
policy.xml"),
            @Policy(placement = Policy.Placement.BINDING_OPERATION_OUTPUT, uri =
"classpath:/io-policy.xml")
    })
    String hello(String person);
}
```

The service endpoint is configured in **application.properties**:

application.properties

```
quarkus.cxf.endpoint."/hello-ws-trust".implementor =
io.quarkiverse.cxf.it.ws.trust.server.TrustHelloServiceImpl
quarkus.cxf.endpoint."/hello-ws-trust".logging.enabled = pretty

quarkus.cxf.endpoint."/hello-ws-trust".security.signature.username = service
quarkus.cxf.endpoint."/hello-ws-trust".security.signature.password = password
quarkus.cxf.endpoint."/hello-ws-
trust".security.signature.properties."org.apache.ws.security.crypto.provider" =
org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/hello-ws-
trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.type" = pkcs12
quarkus.cxf.endpoint."/hello-ws-
trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.password" =
password
quarkus.cxf.endpoint."/hello-ws-
trust".security.signature.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = service
quarkus.cxf.endpoint."/hello-ws-
trust".security.signature.properties."org.apache.ws.security.crypto.merlin.file" = service.pkcs12

quarkus.cxf.endpoint."/hello-ws-
trust".security.encryption.properties."org.apache.ws.security.crypto.provider" =
```

```
org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.endpoint."/hello-ws-
trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.type" = pkcs12
quarkus.cxf.endpoint."/hello-ws-
trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password" =
password
quarkus.cxf.endpoint."/hello-ws-
trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = service
quarkus.cxf.endpoint."/hello-ws-
trust".security.encryption.properties."org.apache.ws.security.crypto.merlin.file" = service.pkcs12
```

5.6.3.1.4. Client

Finally, for the SOAP client to be able to communicate with the service, its **STSClient** needs to be configured. It can be done in **application.properties**:

application.properties

```
quarkus.cxf.client.hello-ws-trust.security.sts.client.wsdl = http://localhost:${quarkus.http.test-
port}/services/sts?wsdl
quarkus.cxf.client.hello-ws-trust.security.sts.client.service-name = {http://docs.oasis-open.org/ws-
sx/ws-trust/200512/}SecurityTokenService
quarkus.cxf.client.hello-ws-trust.security.sts.client.endpoint-name = {http://docs.oasis-open.org/ws-
sx/ws-trust/200512/}UT_Port
quarkus.cxf.client.hello-ws-trust.security.sts.client.username = client
quarkus.cxf.client.hello-ws-trust.security.sts.client.password = password
quarkus.cxf.client.hello-ws-trust.security.sts.client.encryption.username = sts
quarkus.cxf.client.hello-ws-
trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.provider" =
org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.hello-ws-
trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.type" =
pkcs12
quarkus.cxf.client.hello-ws-
trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.password"
= password
quarkus.cxf.client.hello-ws-
trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.alias" =
client
quarkus.cxf.client.hello-ws-
trust.security.sts.client.encryption.properties."org.apache.ws.security.crypto.merlin.keystore.file" =
client.pkcs12
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.username = client
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.provider" =
org.apache.ws.security.components.crypto.Merlin
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.type" =
pkcs12
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.password" =
password
quarkus.cxf.client.hello-ws-
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.alias" = client
quarkus.cxf.client.hello-ws-
```

```
trust.security.sts.client.token.properties."org.apache.ws.security.crypto.merlin.keystore.file" =
client.pkcs12
quarkus.cxf.client.hello-ws-trust.security.sts.client.token.usecert = true
```

TIP

The properties for configuring the STS client are provided by the **io.quarkiverse.cxf:quarkus-cxf-rt-ws-security** extension and documented on its reference page.

Alternatively, the client can be set as a bean reference:

application.properties

```
quarkus.cxf.client.hello-ws-trust-bean.security.sts.client = #stsClientBean
```

In that case, the **@Named** bean needs to be produced programmatically, e.g. using **@jakarta.enterprise.inject.Produces**:

BeanProducers.java

```
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Named;

import org.apache.cxf.ws.security.SecurityConstants;

import io.quarkiverse.cxf.ws.security.sts.client.STSClientBean;

public class BeanProducers {

    /**
     * Create and configure an STSClient for use by the TrustHelloService client.
     */
    @Produces
    @ApplicationScoped
    @Named("stsClientBean")
    STSClientBean createSTSClient() {
        /*
         * We cannot use org.apache.cxf.ws.security.trust.STSClient as a return type of this bean
producer method
         * because it does not have a no-args constructor. STSClientBean is a subclass of STSClient
having one.
         */
        STSClientBean stsClient = STSClientBean.create();
        stsClient.setWsdlLocation("http://localhost:8081/services/sts?wsdl");
        stsClient.setServiceQName(new QName("http://docs.oasis-open.org/ws-sx/ws-trust/200512/",
"SecurityTokenService"));
        stsClient.setEndpointQName(new QName("http://docs.oasis-open.org/ws-sx/ws-trust/200512/",
"UT_Port"));
        Map<String, Object> props = stsClient.getProperties();
        props.put(SecurityConstants.USERNAME, "client");
        props.put(SecurityConstants.PASSWORD, "password");
        props.put(SecurityConstants.ENCRYPT_PROPERTIES,
```

```
Thread.currentThread().getContextClassLoader().getResource("clientKeystore.properties"));
        props.put(SecurityConstants.ENCRYPT_USERNAME, "sts");
        props.put(SecurityConstants.STS_TOKEN_USERNAME, "client");
        props.put(SecurityConstants.STS_TOKEN_PROPERTIES,

Thread.currentThread().getContextClassLoader().getResource("clientKeystore.properties"));
        props.put(SecurityConstants.STS_TOKEN_USE_CERT_FOR_KEYINFO, "true");
        return stsClient;
    }
}
```

5.7. HTTP ASYNC TRANSPORT

Implement async SOAP Clients using Apache HttpComponents HttpClient 5.

5.7.1. Maven coordinates

Create a new project using **quarkus-cxf-rt-transports-http-hc5** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-transports-http-hc5</artifactId>
</dependency>
```

5.7.2. Usage

Once the **quarkus-cxf-rt-transports-http-hc5** dependency is available in the classpath, CXF will use **HttpAsyncClient** for asynchronous calls and will continue using **HttpURLConnection** for synchronous calls.

5.7.2.1. Generate async methods

Asynchronous client invocations require some additional methods in the service endpoint interface. That code is not generated by default.

To enable it, you need to create a JAX-WS binding file with **enableAsyncMapping** set to **true**:

TIP

The sample code snippets used in this section come from the HC5 integration test in the source tree of Quarkus CXF

src/main/resources/wsdl/async-binding.xml

```
<?xml version="1.0"?>
<bindings
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns="https://jakarta.ee/xml/ns/jaxws"
        wsdlLocation="CalculatorService.wsdl">
    <bindings node="wsdl:definitions">
```

```
        <enableAsyncMapping>true</enableAsyncMapping>
    </bindings>
</bindings>
```

This file should then be passed to wsdl2java through its additional-params property:

application.properties

```
quarkus.cxf.codegen.wsdl2java.includes = wsdl/*.wsdl
quarkus.cxf.codegen.wsdl2java.additional-params = -b,src/main/resources/wsdl/async-binding.xml
```

5.7.2.2. Asynchronous Clients and Mutiny

Once the asynchronous stubs are available, it is possible to wrap a client call in **io.smallrye.mutiny.Uni** as shown below:

```
package io.quarkiverse.cxf.hc5.it;

import java.util.concurrent.Future;

import jakarta.inject.Inject;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;

import org.jboss.eap.quickstarts.wscalculator.calculator.AddResponse;
import org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService;

import io.quarkiverse.cxf.annotation.CXFClient;
import io.smallrye.mutiny.Uni;

@Path("/hc5")
public class Hc5Resource {

    @Inject
    @CXFClient("myCalculator") // name used in application.properties
    CalculatorService myCalculator;

    @SuppressWarnings("unchecked")
    @Path("/add-async")
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Uni<Integer> addAsync(@QueryParam("a") int a, @QueryParam("b") int b) {
        return Uni.createFrom()
                .future(
                        (Future<AddResponse>) myCalculator
                                .addAsync(a, b, res -> {
                                }))
                .map(addResponse -> addResponse.getReturn());
    }

}
```

5.7.2.3. Thread pool

Asynchronous clients delivered by this extension leverage **ManagedExecutor** with a thread pool provided by Quarkus. The thread pool can be configured using the **quarkus.thread-pool.*** family of options. As a consequence of this, the executor and thread pool related attributes of **org.apache.cxf.transports.http.configuration.HTTPClientPolicy** are not honored for async clients on Quarkus.

TIP

You can see more details about the CXF asynchronous client and how to tune it further in CXF documentation.

5.8. XJC PLUGINS

XJC plugins for wsdl2java code generation. You'll need to add this extension if you want to use any of the following in quarkus.cxf.codegen.wsdl2java.additional-params:

- **-xjc-Xbg** - generate **getFoo()** instead of **isFoo()** accessor methods for boolean fields.
- **-xjc-Xdv** - let the generated getter methods return the default value defined in the schema unless the field is set explicitly.
- **-xjc-Xjavadoc** - generate JavaDoc based on **xs:documentation** present in the schema.
- **-xjc-Xproperty-listener** - add **PropertyChangeListener** support to the generated beans.
- **-xjc-Xts** - generate **toString()** methods in model classes.
- **-xjc-Xwsdlextension** - generate beans that can be used directly with WSDL4J as extensors in the WSDL.

TIP

Check the wsdl2java section of User guide for more details about **wsdl2java**.

5.8.1. Maven coordinates

Create a new project using **quarkus-cxf-xjc-plugins** on code.quarkus.redhat.com or add these coordinates to your existing project:

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-xjc-plugins</artifactId>
</dependency>
```

CHAPTER 6. QUARKUS CXF USER GUIDE

This chapter provides information about Quarkus CXF usage and configuration.

6.1. USER GUIDE

This User guide explains typical use cases of Quarkus CXF.

6.2. CREATE A NEW PROJECT

This guide explains how to set up a new project for a Quarkus application hosting a CXF client or server or both.

6.2.1. Prerequisites

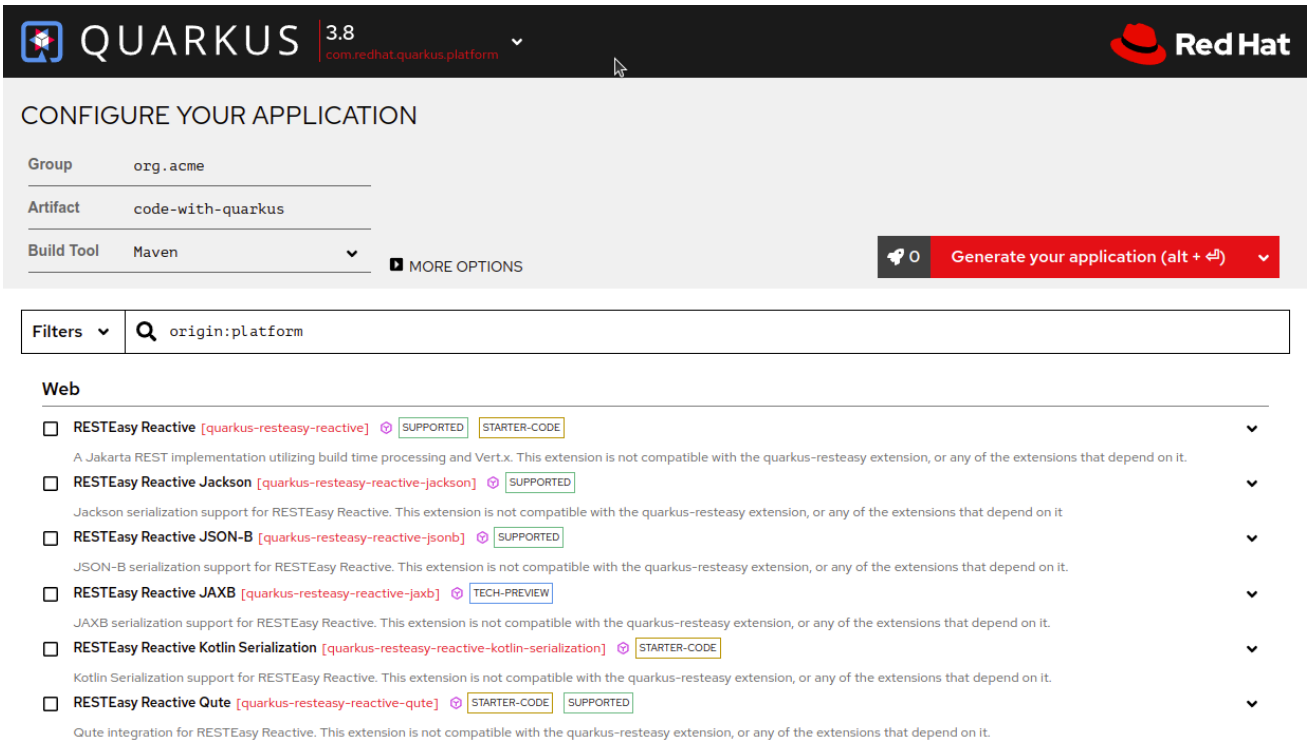
Read the Project prerequisites section of Quarkus getting started guide.

In addition to that, you may need

- GraalVM with the **native-image** command installed and the **GRAALVM_HOME** environment variable set. See Building a native executable section of the Quarkus documentation.
- If you are on Linux, a container runtime like **docker** is sufficient for the native mode too. Use **-Pnative -Dquarkus.native.container-build=true** instead of **-Pnative** if you choose this option.

6.2.2. Creating a project

New project skeletons can be generated using code.quarkus.redhat.com.



- Here you can select the extensions that you want to work with.
- For a simple Hello world Web service or client the **quarkus-cxf** extension is enough.

- Click the blue **Generate your application** button to download a basic skeleton project.
- Unpack the zip file and import the project the into your favorite IDE.

6.2.3. Dependency management

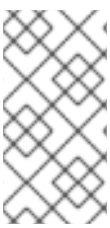
Quarkus CXF is a part of Quarkus Platform since Quarkus Platform version 3.1.0.Final. Among other things, this means that code.quarkus.redhat.com and other Quarkus development tools generate projects with proper dependency management:

NOTE

For **quarkus-cxf** projects you can use either **com.redhat.quarkus.platform:quarkus-camel-bom** or **com.redhat.quarkus.platform:quarkus-cxf-bom** (since **quarkus-camel-bom** is a superset of **quarkus-cxf-bom**).

For **camel-quarkus-cxf-soap** projects you can use **quarkus-camel-bom**.

```
<project ...>
  ...
  <properties>
    ...
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.version><!-- Check the latest
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-cxf-bom/ --
></quarkus.platform.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>quarkus-cxf-bom</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
...
```



NOTE

Always take care to import the same version of **com.redhat.quarkus.platform:quarkus-bom** and **com.redhat.quarkus.platform:quarkus-cxf-bom** into your project. That's the most reliable way to get compatible versions of Quarkus, CXF, Quarkus CXF and all their transitive dependencies.

6.2.4. Where to go next

We recommend to proceed with any of the following chapters:

- Your first SOAP Web service
- Your first SOAP Client

6.3. YOUR FIRST SOAP WEB SERVICE ON QUARKUS

In this guide we explain how to create a Quarkus application exposing a simple SOAP Web service.

CREATE PROJECT FIRST

Follow the Section 6.2, "Create a new project" guide before proceeding here.

6.3.1. Hello world! Web service

Having the **pom.xml** in place, you can add a simple Hello world! Web service in **src/main/java**.

CODE EXAMPLES

The sample code snippets used in this section come from the server integration test in the source tree of Quarkus CXF

First add the service interface:

HelloService.java

```
package io.quarkiverse.cxf.it.server;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

/**
 * The simplest Hello service.
 */
@WebService(name = "HelloService", serviceName = "HelloService")
public interface HelloService {

    @WebMethod
    String hello(String text);

}
```

and then the implementation:

HelloServiceImpl.java

```
package io.quarkiverse.cxf.it.server;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;
```

```
/**
 * The simplest Hello service implementation.
 */
@WebService(serviceName = "HelloService")
public class HelloServiceImpl implements HelloService {

    @WebMethod
    @Override
    public String hello(String text) {
        return "Hello " + text + "!";
    }

}
```

For the implementation to get exposed under a certain path, you need to add the following configuration to **application.properties**:

```
# The context path under which all services will be available
quarkus.cxf.path = /soap

# Publish "HelloService" under the context path /${quarkus.cxf.path}/hello
quarkus.cxf.endpoint."/hello".implementor = io.quarkiverse.cxf.it.server.HelloServiceImpl
quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature
```

TIP

All configuration properties are documented in the Configuration properties reference.

With these files in place, you can start Quarkus in **dev** mode:

```
$ mvn quarkus:dev
```

This will compile the project and start the application on the background.

You can test the service using **curl** or some other SOAP client.

First let's have a look at the auto-generated WSDL under http://localhost:8080/soap/hello?wsdl:

```
$ curl http://localhost:8080/soap/hello?wsdl
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://server.it.cxf.quarkiverse.io/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
    name="HelloService" targetNamespace="http://server.it.cxf.quarkiverse.io/">
  <wsdl:types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://server.it.cxf.quarkiverse.io/" attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://server.it.cxf.quarkiverse.io/">
  <xsd:element name="hello" type="tns:hello"/>
  <xsd:complexType name="hello">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="arg0" type="xsd:string"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="helloResponse" type="tns:helloResponse"/>
  <xsd:complexType name="helloResponse">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="return" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
  </wsdl:types>
  <wsdl:message name="helloResponse">
    <wsdl:part element="tns:helloResponse" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="hello">
    <wsdl:part element="tns:hello" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="HelloService">
    <wsdl:operation name="hello">
      <wsdl:input message="tns:hello" name="hello">
    </wsdl:input>
      <wsdl:output message="tns:helloResponse" name="helloResponse">
    </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloServiceSoapBinding" type="tns:HelloService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="hello">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="hello">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="helloResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloService">
    <wsdl:port binding="tns:HelloServiceSoapBinding" name="HelloServicePort">
      <soap:address location="http://localhost:8080/soap/hello"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Second, let's send a SOAP request to the service:

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
      '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body><ns2:hello xmlns:ns2="http://server.it.cxf.quarkiverse.io/"><arg0>World</arg0>
</ns2:hello></soap:Body>
       </soap:Envelope>' \
    http://localhost:8080/soap/hello
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
```

```
    <ns1:helloResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <return>Hello World!</return>
    </ns1:helloResponse>
  </soap:Body>
</soap:Envelope>
```

You can see the expected **<return>Hello World!</return>** in the SOAP response.

6.3.2. Add the logging feature while dev mode is running

Sometimes it may come in handy to be able to inspect the SOAP messages received or sent by the server or client. This is easily doable by adding the **quarkus-cxf-rt-features-logging** extension to **pom.xml**.

TIP

Do this while Quarkus dev mode is running. When you save your changes in the source tree, the application is recompiled and redeployed.

Add this to pom.xml

```
<dependency>
    <groupId>io.quarkiverse.cxf</groupId>
    <artifactId>quarkus-cxf-rt-features-logging</artifactId>
</dependency>
```

Enable SOAP payload logging in application.properties

```
quarkus.cxf.endpoint."/hello".features=org.apache.cxf.ext.logging.LoggingFeature
```

After that you can send a new SOAP request and see some SOAP payloads in the application console:

```
2023-01-11 22:12:21,315 INFO  [org.apa.cxf.ser.Hel.REQ_IN] (vert.x-worker-thread-0) REQ_IN
    Address: http://localhost:8080/soap/hello
    HttpMethod: POST
    Content-Type: text/xml;charset=UTF-8
    ExchangeId: af10747a-8477-4c17-bf5f-2a4a3a95d61c
    ServiceName: HelloService
    PortName: HelloServicePort
    PortTypeName: HelloService
    Headers: {Accept=*/*, User-Agent=curl/7.79.1, content-type=text/xml;charset=UTF-8,
Host=localhost:8080, Content-Length=203, x-quarkus-hot-deployment-done=true}
    Payload: <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body><ns2:hello xmlns:ns2="http://server.it.cxf.quarkiverse.io/"><arg0>World</arg0>
</ns2:hello></soap:Body>
</soap:Envelope>


2023-01-11 22:12:21,327 INFO  [org.apa.cxf.ser.Hel.RESP_OUT] (vert.x-worker-thread-0)
RESP_OUT
    Address: http://localhost:8080/soap/hello
    Content-Type: text/xml
    ResponseCode: 200
```

```
    ExchangeId: af10747a-8477-4c17-bf5f-2a4a3a95d61c
    ServiceName: HelloService
    PortName: HelloServicePort
    PortTypeName: HelloService
    Headers: {}
    Payload: <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<ns1:helloResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/"><return>Hello World!</return>
</ns1:helloResponse></soap:Body></soap:Envelope>
```

6.3.3. Further steps

You may want to proceed with packaging your application for running on a JVM or natively .

6.4. CREATING YOUR FIRST SOAP CLIENT ON QUARKUS

In this guide we explain how to create a simple Quarkus application acting as a client of a remote Web service.

CREATE PROJECT FIRST

Follow the Section 6.2, "Create a new project" guide before proceeding here.

6.4.1. Remote Web service for testing

First, we need some remote Web service to connect to. We can use a simple Calculator Web service running in a container for that purpose.

```
$ docker run -p 8082:8080 quay.io/l2x6/calculator-ws:1.0
```

Once the container is up and running, we can inspect its WSDL

```
$ curl -s http://localhost:8082/calculator-ws/CalculatorService?wsdl
<?xml version="1.0" ?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="CalculatorService"
targetNamespace="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">

  ...

  <wsdl:binding name="CalculatorServiceSoapBinding" type="tns:CalculatorService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http">
</soap:binding>
    <wsdl:operation name="add">
      <soap:operation soapAction="" style="document"></soap:operation>
      <wsdl:input name="add">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="addResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
```

```
    </wsdl:operation>
    <wsdl:operation name="subtract">
      <soap:operation soapAction="" style="document"></soap:operation>
      <wsdl:input name="subtract">
        <soap:body use="literal"></soap:body>
      </wsdl:input>
      <wsdl:output name="subtractResponse">
        <soap:body use="literal"></soap:body>
      </wsdl:output>
    </wsdl:operation>

    ...

  </wsdl:binding>

  ...

</wsdl:definitions>
```

As you can see in the WSDL, the service offers some basic arithmetic operations, such as **add**, **subtract**, etc.

Let's test it with **curl**:

```
$ curl -s \
    -X POST \
    -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
       '<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
          <Body>
             <add xmlns="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">
                <arg0 xmlns="">7</arg0> (1)
                <arg1 xmlns="">4</arg1>
             </add>
          </Body>
       </Envelope>' \
    http://localhost:8082/calculator-ws/CalculatorService
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:addResponse xmlns:ns2="http://www.jboss.org/eap/quickstarts/wscalculator/Calculator">
      <return>11</return> (2)
    </ns2:addResponse>
  </soap:Body>
</soap:Envelope>
```

(1) The request to add **7** and **4**

(2) **11** - return value of the operation

6.4.2. SOAP client

Now let's have a look how we can get the client inside a Quarkus application.

First, we need the Service Endpoint Interface (SEI) and all other model classes it requires.

There are several ways to get them:

- Write by hand
- Copy from the Web Sevice project, if it is written in Java
- Have a Maven artifact containing the model classes, perhaps it is offered by the Service project
- Generate the model classes from WSDL

The last option tends to be the easiest and most flexible for client applications.

TIP

If you want to use this approach, first follow the Generate Java from WSDL section and then continue with the next steps.

6.4.3. Using SEI as a client

In our case, the Service Endpoint Interface (SEI) is **org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService**.

As usual on Quarkus, we can obtain an instance of it via CDI.

To make it testable easily, we'll wrap it in a REST service:

CxfClientResource.java

```
package io.quarkiverse.cxf.client.it;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;

import org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService;

import io.quarkiverse.cxf.annotation.CXFClient;

@Path("/cxf/calculator-client")
public class CxfClientRestResource {

    @CXFClient("myCalculator") (1)
    CalculatorService myCalculator;

    @GET
    @Path("/add")
    @Produces(MediaType.TEXT_PLAIN)
    public int add(@QueryParam("a") int a, @QueryParam("b") int b) {
        return myCalculator.add(a, b); (2)
    }

}
```

1. Let the CDI container inject an instance of the client. **@CXFClient("myCalculator")** is actually equivalent to **@Inject @CXFClient("myCalculator")**
2. Invoke the **add** operation thus calling the remote Web service

Is this all we need for the client to work? - No, in addition to the above, we need to tell a few other things to the CXF Quarkus extension in **application.properties**:

application.properties

```
cxf.it.calculator.baseUri=http://localhost:8082
quarkus.cxf.client.myCalculator.wsdl = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService?wsdl
quarkus.cxf.client.myCalculator.client-endpoint-url = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService
quarkus.cxf.client.myCalculator.service-interface = org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService
```

TIP

All client configuration properties are documented in the Configuration properties reference.

With all the above files in place, we should be able to start the application in Quarkus **dev** mode

```
$ mvn quarkus:dev
...
INFO  [io.quarkus] (Quarkus Main Thread) ... Listening on: http://localhost:8080
```

and test it by sending some requests to it:

```
$ curl -s 'http://localhost:8080/cxf/calculator-client/add?a=5&b=6'
11
```

where **11** is the correct result of adding **5** and **6**.

6.4.4. Further steps

You may want to proceed with

- Package for JVM and native.
- Advanced SOAP client topics

6.5. PROJECT CONFIGURATION OPTIONS

Quarkus CXF exposes a large number of configuration options.

The configuration options can be set in **application.properties** file or via environment variables - see Quarkus configuration reference for details.

6.5.1. Bean references

Several configuration options of Quarkus CXF allow referring to beans present in Quarkus CDI container. Features and interceptors are typical examples of those.

There are two ways how to set a bean reference in the configuration: by type or by bean name.

6.5.1.1. Bean reference by type

Here is an example:

application.properties

```
# bean reference by type
quarkus.cxf.endpoint."/hello".features = org.apache.cxf.ext.logging.LoggingFeature
```

When using a reference by type name, the resolution proceeds as follows:

- Fist the bean is looked up in Quarkus CDI container by type.
- If the bean is available, it is used.
- If multiple beans assignable to the given type, then an exception is thrown.
- If no matching bean is available, then the class is loaded and an attempt is performed to instantiate it using its default constructor.

6.5.1.2. Bean reference by bean name

Here is an example:

application.properties

```
# bean reference by bean name
quarkus.cxf.endpoint."/fruit".features = #myCustomLoggingFeature
```

When using a reference by bean name, then unsurprisingly, the bean is looked up in Quarkus CDI container by name. A named bean called **myCustomLoggingFeature** can be defined as follows:

```
import org.apache.cxf.ext.logging.LoggingFeature;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Produces;

class Producers {

    @Produces
    @ApplicationScoped
    @Named("myCustomLoggingFeature")
    LoggingFeature myCustomLoggingFeature() {
        LoggingFeature loggingFeature = new LoggingFeature();
        loggingFeature.setPrettyLogging(true);
        return loggingFeature;
    }
}
```

6.6. PACKAGE FOR RUNNING ON A JVM OR NATIVELY

In this chapter, we explain how to package a Quarkus CXF application for running on a JVM or for running it natively.

6.6.1. JVM mode

In the introductory guides for SOAP client and SOAP service, we worked only in Quarkus **dev** mode: Quarkus tooling was running on the background, watching for changes in our workspace, recompiling and reloading the application as needed.

How do we run the application on a JVM once we are done with the development?

First we need to package it with Maven:

```
$ mvn package
```

The libraries needed to run the application on a JVM can be found in **target/quarkus-app** directory:

```
$ ls -lh target/quarkus-app
drwxr-xr-x. 2 ppalaga ppalaga 4.4K Jan 12 22:29 app
drwxr-xr-x. 4 ppalaga ppalaga 4.4K Jan 12 22:29 lib
drwxr-xr-x. 2 ppalaga ppalaga 4.4K Jan 12 22:29 quarkus
-rw-r-r--. 1 ppalaga ppalaga 6.1K Jan 12 22:29 quarkus-app-dependencies.txt
-rw-r-r--. 1 ppalaga ppalaga  678 Jan 12 22:29 quarkus-run.jar
```

We can start the application as follows:

```
$ java -jar target/quarkus-app/quarkus-run.jar
```

You can send some SOAP requests using **curl** to make sure that the application works.

6.6.2. Native mode

Quarkus offers first class support for building GraalVM native images and Quarkus CXF fully honors that promise too.

IMAGES

GraalVM native images are platform specific executable files that you can run directly without a JVM. They boot faster and spend less memory compared to running the same application in JVM mode.

The **pom.xml** file generated by code.quarkus.redhat.com contains the **native** profile needed for building the native image:

```
<profile>
  <id>native</id>
  <activation>
    <property>
      <name>native</name>
    </property>
  </activation>
```

```
  <properties>
    <skipITs>false</skipITs>
    <quarkus.package.type>native</quarkus.package.type>
  </properties>
</profile>
```

Further, as mentioned in the creating a project section, you need the GraalVM **native-image** tool.

Either install it locally or set your **GRAALVM_HOME** environment appropriately, or – if you only need to produce a Linux native executable – you can use **docker**.

With local installation of GraalVM

```
# Make sure $GRAALVM_HOME is set properly
$ echo $GRAALVM_HOME
/home/{user}/.sdkman/candidates/java/{major}.{minor}.r{java-version}-grl

# Produce the native executable
mvn package -Pnative
```

TIP

Quarkus is quite picky about the GraalVM version. When using the local installation, always make sure that you use the version preferred by Quarkus. You can do that by opening **quarkus-bom** imported in your **pom.xml** and searching for **graalvm** there. If you use Docker, Quarkus takes care for pulling the right version for you.

With docker

```
# Produce the native executable
mvn package -Pnative -Dquarkus.native.container-build=true
```

This can take a minute or so for a simple application.

When the build is done, the native executable should be available in **target** directory:

```
$ ls -l target
...
-rwxr-xr-x. 1 ppalaga ppalaga  71M Jan 11 22:42 quarkus-cxf-integration-test-server-1.8.0-
SNAPSHOT-runner
...
```

As you can see, it has a size of only 71 MB, and is executable.

You can run it as follows:

```
$ target/*-runner
...
INFO  [io.quarkus] (main) quarkus-cxf-integration-test-server 1.8.0-SNAPSHOT native (powered by
Quarkus
2.15.2.Final) started in 0.042s. Listening on: http://0.0.0.0:8080
...
```

Again, you can send some SOAP requests using **curl** to make sure that the native executable works.

Do not forget to compare the memory usage, time to first request and other performance metrics with the stack you used before and share your results!

6.6.3. Native Image: Additional Resources

You may also refer to the links below which contain tips on how to work with native images.

- Quarkus: Building a Native Executable
- Quarkus: Tips for Writing Native Applications
- Quarkus: Native Reference Guide
- GraalVM: Accessing Resources in Native Images
- GraalVM: Reflection Use in Native Images
- GraalVM: Tracing Agent

6.6.4. Create container image

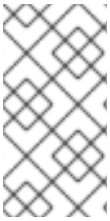
Refer to Quarkus Container image guide.

6.7. LOGGING

Refer to Quarkus Logging guide for basic information about logging on Quarkus, such as

- Getting a logger in your application code
- Log levels
- Categories
- Format
- JSON format

6.7.1. Payload logging



HISTORY

Since Quarkus CXF 2.6.0, the payload logging functionality is available via **io.quarkiverse.cxf:quarkus-cxf** extension. Before 2.6.0, it was available through a separate extension **io.quarkiverse.cxf:quarkus-cxf-rt-features-logging** which is now deprecated and will be removed in the future.

The payload logging functionality is implemented primarily through the **org.apache.cxf.ext.logging.LoggingFeature** class.

There are several ways how you can set the feature on a client or service endpoint.

6.7.2. Configuring payload logging through configuration properties

6.7.2.1. Global settings

The global logging options exist since Quarkus CXF 2.6.0. If enabled via **quarkus.cxf.logging.enabled = true**, Quarkus CXF creates a global **LoggingFeature** instance and sets it on every client and service endpoint in the application. The global settings can be overriden on the client or service endpoint level.

application.properties

```
# Global settings
quarkus.cxf.logging.enabled = true
quarkus.cxf.logging.pretty = true
```

All logging configuration options are listed on **quarkus-cxf** reference page.

TIP

All logging properties mentioned on this page are **runtime** configuration options. Hence you can pass them when starting the application without having to rebuild it. It can be done either by passing a system property on the command line (e.g. **-Dquarkus.cxf.logging.enabled=true**) or by setting an environment variable (e.g. **export QUARKUS_CXF_LOGGING_ENABLED=true**).

6.7.2.2. Per client and per service endpoint settings

Since Quarkus CXF 2.5.0, the **LoggingFeature** can be configured and attached to a client or a service endpoint declaratively by setting the appropriate options in **application.properties**:

application.properties

```
# For a service:
quarkus.cxf.endpoint."/hello".logging.enabled = true
quarkus.cxf.endpoint."/hello".logging.pretty = true
# For a client:
quarkus.cxf.client.hello.logging.enabled = true
quarkus.cxf.client.hello.logging.pretty = true
```

All logging configuration options are documented on **quarkus-cxf** reference page:

- For clients
- For service endpoints

6.7.3. Alternative ways of adding a LoggingFeature to a client or service

To attach an instance with default settings, you can do one of the following:

1. In **application.properties**:

```
# For a service:
quarkus.cxf.endpoint."/hello".features=org.apache.cxf.ext.logging.LoggingFeature
# For a client:
quarkus.cxf.client."myClient".features=org.apache.cxf.ext.logging.LoggingFeature
```

TIP

There is an example in Your first SOAP Web service chapter of the User guide.

or alternatively

2. Use the **@Features** annotation of CXF:

```
@org.apache.cxf.feature.Features (features = {"org.apache.cxf.ext.logging.LoggingFeature"})
@WebService(endpointInterface = "org.acme.SayHi", targetNamespace = "uri:org.acme")
public class SayHiImplementation implements SayHi {
  public long sayHi(long arg) {
     return arg;
  }
  //...
}
```

6.7.3.1. Producing a custom **LoggingFeature** bean

If you need some custom logic to setup the **LoggingFeature**, you can produce a named **LoggingFeature** bean:

```
import org.apache.cxf.ext.logging.LoggingFeature;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;

class Producers {

   @Produces
   @ApplicationScoped
   @Named("limitedLoggingFeature") // "limitedLoggingFeature" is redundant if the name of the
method is the same
   LoggingFeature limitedLoggingFeature() {
      LoggingFeature loggingFeature = new LoggingFeature();
      loggingFeature.setPrettyLogging(true);
      loggingFeature.setLimit(1024);
      return loggingFeature;
   }
}
```

Then you can refer to it by its name prefixed with **#** in **application.properties**:

```
# For a service:
quarkus.cxf.endpoint."/hello".features = #limitedLoggingFeature
# For a client:
quarkus.cxf.client.hello.features = #limitedLoggingFeature
```

6.8. COMPLEX SOAP PAYLOADS WITH JAXB

Our introductory guides for Quarkus SOAP client and Your first SOAP Web service dealt with services having only primitive parameters and return values such as integers and strings. Let's have a look at passing and receiving more complex objects.

As an example, let's create an application for managing fruits.

NOTE

The sample code snippets used in this section come from the server integration test in the source tree of Quarkus CXF

Because our representation of fruit is supposed to be a complex, let's model it as a Java bean with a couple of attributes:

```
package io.quarkiverse.cxf.it.server;

import java.util.Objects;

import jakarta.xml.bind.annotation.XmlElement;
import jakarta.xml.bind.annotation.XmlRootElement;
import jakarta.xml.bind.annotation.XmlType;

@XmlType(name = "Fruit")
@XmlRootElement
public class Fruit {

    private String name;

    private String description;

    public Fruit() {
    }

    public Fruit(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getName() {
        return name;
    }

    @XmlElement
    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    @XmlElement
    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Fruit)) {
```

```
            return false;
        }

        Fruit other = (Fruit) obj;

        return Objects.equals(other.getName(), this.getName());
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.getName());
    }
}
```

As you may have noticed, we have used some JAXB annotations, such as **@XmlElement**, **@XmlRootElement** and **@XmlType**. This is to control the serialization and deserialization of the bean from and to XML.

6.8.1. Automatic registration for reflection

JAXB is a reflection based serialization framework. When learning about GraalVM native images, one of the first things you typically hear is that you have to register classes, fields and methods for reflection at build time. With plain GraalVM you'd have to do this through **reflection-config.json** manually. Well, at least for the classes you have written yourself. Not so with Quarkus. **quarkus-jaxb** extension (which **quarkus-cxf** depends on) is able to scan your application's class path for classes annotated with JAXB annotations and register them for reflection automatically.

Hence working with complex payloads on Quarkus is not different from stock CXF. The JAXB serialization and deserialization will work out of the box without any additional configuration.

6.8.2. SEI and implementation

The Service Endpoint Interface (SEI) for managing fruits might look like the following:

```
package io.quarkiverse.cxf.it.server;

import java.util.Set;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService
public interface FruitService {

    @WebMethod
    Set<Fruit> list();

    @WebMethod
    Set<Fruit> add(Fruit fruit);

    @WebMethod
    Set<Fruit> delete(Fruit fruit);
}
```

We can implement the SEI as simply as possible:

```
package io.quarkiverse.cxf.it.server;

import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.Set;

import jakarta.jws.WebService;

@WebService(serviceName = "FruitService")
public class FruitServiceImpl implements FruitService {

    private Set<Fruit> fruits = Collections.synchronizedSet(new LinkedHashSet<>());

    public FruitServiceImpl() {
        fruits.add(new Fruit("Apple", "Winter fruit"));
        fruits.add(new Fruit("Pineapple", "Tropical fruit"));
    }

    @Override
    public Set<Fruit> list() {
        return fruits;
    }

    @Override
    public Set<Fruit> add(Fruit fruit) {
        fruits.add(fruit);
        return fruits;
    }

    @Override
    public Set<Fruit> delete(Fruit fruit) {
        fruits.remove(fruit);
        return fruits;
    }
}
```

6.8.3. application.properties

The implementation is pretty straightforward and you just need to define your endpoints using the **application.properties**.

```
quarkus.cxf.endpoint."/fruits".implementor = io.quarkiverse.cxf.it.server.FruitServiceImpl
quarkus.cxf.endpoint."/fruits".features = org.apache.cxf.ext.logging.LoggingFeature
```

6.8.4. Test with Quarkus dev mode and curl

Having the above files in place, you can start Quarkus tooling in **dev** mode:

```
$ mvn quarkus:dev
...
INFO  [io.quarkus] (Quarkus Main Thread) ... Listening on: http://localhost:8080
```

and then check whether the service is working by invoking its **list** operation:

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
     '<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
       <soapenv:Body>
          <ns1:list/>
       </soapenv:Body>
    </soapenv:Envelope>' \
    http://localhost:8080/soap/fruits
...
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:listResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Winter fruit</description>
        <name>Apple</name>
      </return>
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Tropical fruit</description>
        <name>Pineapple</name>
      </return>
    </ns1:listResponse>
  </soap:Body>
</soap:Envelope>
```

As you can see, the endpoint has returned the two fruits **Apple** and **Pineapple** available by default.

Now let's add another fruit, say an **Orange**:

```
$ curl -v -X POST -H "Content-Type: text/xml;charset=UTF-8" \
    -d \
     '<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
       <soap:Body>
         <ns2:add xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
           <arg0>
             <description>Mediterranean fruit</description>
             <name>Orange</name>
           </arg0>
         </ns2:add>
       </soap:Body></soap:Envelope>' \
    http://localhost:8080/soap/fruits
...
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:addResponse xmlns:ns1="http://server.it.cxf.quarkiverse.io/">
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Winter fruit</description>
        <name>Apple</name>
      </return>
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
        <description>Tropical fruit</description>
        <name>Pineapple</name>
      </return>
      <return xmlns:ns2="http://server.it.cxf.quarkiverse.io/">
```

```
      <description>Mediterranean fruit</description>
      <name>Orange</name>
    </return>
  </ns1:addResponse>
 </soap:Body>
</soap:Envelope>
```

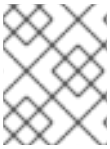
We can see **Orange** having been added in the returned list as expected.

6.8.5. Further steps

You may want to proceed with packaging your application for running on a JVM or natively .

6.9. SSL

This chapter documents various use cases related to SSL, TLS and HTTPS.



NOTE

The sample code snippets used in this section come from the WS-SecurityPolicy integration test in the source tree of Quarkus CXF

6.9.1. Client SSL configuration

If your client is going to communicate with a server whose SSL certificate is not trusted by the client's operating system, then you need to set up a custom trust store for your client. Tools like **openssl** or Java **keytool** are commonly used for doing that. You may want to check someexamples in Quarkus CXF source tree.

Once you have prepared the trust store, you need to configure your client to use it.

6.9.1.1. Set the client trust store in application.properties

This is the easiest way to set the client trust store. The key role is played by the following properties:

- **quarkus.cxf.client."clients".trust-store**
- **quarkus.cxf.client."clients".trust-store-type**
- **quarkus.cxf.client."clients".trust-store-password**

Here is an example:

application.properties

```
keystore.type = jks (1)
quarkus.cxf.client.hello.client-endpoint-url = https://localhost:${quarkus.http.test-ssl-
port}/services/hello
quarkus.cxf.client.hello.service-interface = io.quarkiverse.cxf.it.security.policy.HelloService
quarkus.cxf.client.hello.trust-store-type = ${keystore.type}
(2)
quarkus.cxf.client.hello.trust-store = client-truststore.${keystore.type}
quarkus.cxf.client.hello.trust-store-password = password
```

1 **pkcs12** trust store type is a common alternative to **jks**.

2 The referenced **client-truststore.jks** file has to be available in **src/main/resources** directory.

6.9.2. Server SSL configuration

The server SSL configuration is driven by Quarkus HTTP layer a.k.a. Vert.x. Quarkus HTTP guide provides the information about the configuration options.

Here is a basic example:

application.properties

```
1
quarkus.http.ssl.certificate.key-store-file = localhost.${keystore.type}
quarkus.http.ssl.certificate.key-store-password = password
quarkus.http.ssl.certificate.key-store-key-alias = localhost
quarkus.http.ssl.certificate.key-store-key-password = password
```

1 The referenced **localhost.jks** file has to be available in **src/main/resources** directory.

6.9.3. Enforce SSL through WS-SecurityPolicy

The requirement for the clients to connect through HTTPS can be defined in a policy.

The functionality is provided by **quarkus-cxf-rt-ws-security** extension.

Here is an example of a policy file:

https-policy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <wsp:ExactlyOne>
        <wsp:All>
            <sp:TransportBinding>
                <wsp:Policy>
                    <sp:TransportToken>
                        <wsp:Policy>
                            <sp:HttpsToken RequireClientCertificate="false" />
                        </wsp:Policy>
                    </sp:TransportToken>
                    <sp:IncludeTimestamp />
                    <sp:AlgorithmSuite>
                        <wsp:Policy>
                            <sp:Basic128 />
                        </wsp:Policy>
                    </sp:AlgorithmSuite>
                </wsp:Policy>
            </sp:TransportBinding>
```

```
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

The policy has to be referenced from a service endpoint interface (SEI):

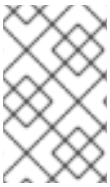
HttpsPolicyHelloService.java

```
package io.quarkiverse.cxf.it.security.policy;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

import org.apache.cxf.annotations.Policy;

/**
 * A service implementation with a transport policy set
 */
@WebService(serviceName = "HttpsPolicyHelloService")
@Policy(placement = Policy.Placement.BINDING, uri = "https-policy.xml")
public interface HttpsPolicyHelloService extends AbstractHelloService {

    @WebMethod
    @Override
    public String hello(String text);

}
```

With this setup in place, any request delivered over HTTP will be rejected by the **PolicyVerificationInInterceptor**:

```
ERROR [org.apa.cxf.ws.pol.PolicyVerificationInInterceptor] Inbound policy verification failed: These policy alternatives can not be satisfied:
 {http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702}TransportBinding: TLS is not enabled
 ...
```

6.10. AUTHENTICATION AND AUTHORIZATION



NOTE

The sample code snippets shown in this section come from the Client and server integration test in the source tree of Quarkus CXF. You may want to use it as a runnable example.

6.10.1. Client HTTP basic authentication

Use the following client configuration options provided by **quarkus-cxf** extension to pass the username and password for HTTP basic authentication:

- **quarkus.cxf.client."clients".username**
- **quarkus.cxf.client."clients".password**

Here is an example:

application.properties

```
quarkus.cxf.client.basicAuth.wsdl = http://localhost:${quarkus.http.test-port}/soap/basicAuth?wsdl
quarkus.cxf.client.basicAuth.client-endpoint-url = http://localhost:${quarkus.http.test-port}/soap/basicAuth
quarkus.cxf.client.basicAuth.username = bob
quarkus.cxf.client.basicAuth.password = bob234
```

6.10.2. Accessing WSDL protected by basic authentication

By default, the clients created by Quarkus CXF do not send the **Authorization** header, unless you set the **quarkus.cxf.client."clients".secure-wsdl-access** to **true**:

application.properties

```
quarkus.cxf.client.basicAuthSecureWsdl.wsdl = http://localhost:${quarkus.http.test-port}/soap/basicAuth?wsdl
quarkus.cxf.client.basicAuthSecureWsdl.client-endpoint-url = http://localhost:${quarkus.http.test-port}/soap/basicAuthSecureWsdl
quarkus.cxf.client.basicAuthSecureWsdl.username = bob
quarkus.cxf.client.basicAuthSecureWsdl.password = ${client-server.bob.password}
quarkus.cxf.client.basicAuthSecureWsdl.secure-wsdl-access = true
```

6.10.3. Securing service endpoints

The server-side authentication and authorization is driven by Quarkus Security, especially when it comes to

- Authentication mechanisms
- Identity providers
- Role-based access control (RBAC)

There is a basic example in our Client and server integration test. Its key parts are:

- **io.quarkus:quarkus-elytron-security-properties-file** dependency as an Identity provider
- Basic authentication enabled and users with their roles configured in **application.properties**:

  application.properties

  ```
  quarkus.http.auth.basic = true
  quarkus.security.users.embedded.enabled = true
  quarkus.security.users.embedded.plain-text = true
  quarkus.security.users.embedded.users.alice = alice123
  quarkus.security.users.embedded.roles.alice = admin
  quarkus.security.users.embedded.users.bob = bob234
  quarkus.security.users.embedded.roles.bob = app-user
  ```

- Role-based access control enfoced via **@RolesAllowed** annotation:

BasicAuthHelloServiceImpl.java

```
package io.quarkiverse.cxf.it.auth.basic;

import jakarta.annotation.security.RolesAllowed;
import jakarta.jws.WebService;

import io.quarkiverse.cxf.it.HelloService;

@WebService(serviceName = "HelloService", targetNamespace = HelloService.NS)
@RolesAllowed("app-user")
public class BasicAuthHelloServiceImpl implements HelloService {
    @Override
    public String hello(String person) {
        return "Hello " + person + "!";
    }
}
```

6.11. ADVANCED SOAP CLIENT TOPICS

6.11.1. client-endpoint-url defaults

If you omit the **client-endpoint-url** property in **application.properties**, the CXF Quarkus extension will assume that the service is published at **http://localhost:8080/{service-path}**, where **{service-path}** is derived from

- Configuration property **quarkus.cxf.path** (if specified); and the
- SEI's class name in lower case

Given **quarkus.cxf.path = /ws**, the default effective **client-endpoint-url** of the **CalculatorService** would be **http://localhost:8080/ws/org.jboss.eap.quickstarts.wscalculator.calculator.calculatorservice**.

If **quarkus.cxf.path** is not specified, the **client-endpoint-url** would be just **http://localhost:8080/org.jboss.eap.quickstarts.wscalculator.calculator.calculatorservice**.

6.11.2. Configure multiple clients

In the example above, we configured just a single client called **myCalculator**. Of course, you can configure multiple clients pointing at different URLs and/or implementing different SEIs using multiple identifiers:

application.properties

```
cxf.it.calculator.baseUri = http://localhost:8082
quarkus.cxf.client.myCalculator.wsdl = ${cxf.it.calculator.baseUri}/calculator-ws/CalculatorService?
wsdl
quarkus.cxf.client.myCalculator.client-endpoint-url = ${cxf.it.calculator.baseUri}/calculator-
ws/CalculatorService
quarkus.cxf.client.myCalculator.service-interface =
org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService

# another client
```

```
quarkus.cxf.client.anotherCalculator.wsdl = https://acme.com/ws/WeatherService?wsdl
quarkus.cxf.client.anotherCalculator.client-endpoint-url = https://acme.com/ws/WeatherService
quarkus.cxf.client.anotherCalculator.service-interface =
org.jboss.eap.quickstarts.wscalculator.calculator.CalculatorService
```

6.11.3. Advanced Client Configurations

To globally configure all clients in your application, you can use the example snippet below to configure the **HttpConduit**. This allows you to set the **HTTPClientPolicy**, **AuthorizationPolicy**, **ProxyAuthorizationPolicy** or even **TLSClientParameters** for your clients.

```
void onStart(@Observes StartupEvent ev) {

    HTTPConduitConfigurer httpConduitConfigurer = new HTTPConduitConfigurer() {
      public void configure(String name, String address, HTTPConduit c) {
        AsyncHTTPConduit conduit = (AsyncHTTPConduit)c;
        // use setter to configure client
        conduit.getHttpAsyncClient().getCredentialsProvider().setCredentials( AuthScope.ANY,
         new NTCredentials( USER,PWD, "", DOM ) );
        conduit.getClient().setAllowChunking( false );
        conduit.getClient().setAutoRedirect( true );
      }
    };

    final Bus bus = BusFactory.getDefaultBus();
    bus.setExtension(httpConduitConfigurer, HTTPConduitConfigurer.class);
}
```

6.11.4. Pure client applications

Quarkus batch (e.g. periodically scheduled), or command line applications, may do without an HTTP server. Use the property below to prevent launching the HTTP server at startup:

```
quarkus.http.host-enabled = false
```

6.11.5. Prevent resource leaks

CXF client proxies implement **java.io.Closeable**. Therefore, it is important to call **((Closeable) proxy).close()** once the client is not needed anymore to free all associated system resources, such as threads.

Quarkus CXF takes care for closing the clients injected via **@io.quarkiverse.cxf.annotation.CXFClient** automatically as soon as they are disposed by the CDI container.

For client proxies created manually, it is up to you to call **((Closeable) proxy).close()**:

```
import java.net.URL;
import javax.xml.namespace.QName;
import jakarta.xml.ws.Service;
import java.io.Closeable;

final URL serviceUrl = new URL("http://localhost/myService?wsdl");
final QName qName = new QName("http://acme.org/myNamespace", "MyService");
```

```
final Service service = jakarta.xml.ws.Service.create(serviceUrl, qName);
final MyService proxy = service.getPort(MyService.class);

try {
    proxy.doSomething();
} finally {
    ((Closeable) proxy).close();
}
```

6.12. RUNNING BEHIND A REVERSE PROXY

SOAP requests aimed towards services running on Quarkus can be routed through proxies that generate additional headers (e.g. **X-Forwarded-Host**) to keep information from the client-facing side of the proxy servers that is altered or lost when they are involved. In those scenarios, Quarkus can be configured to automatically update information like protocol, host, port and URI reflecting the values in these headers.

TIP

Refer to Quarkus HTTP reference for more details.

Quarkus CXF support for various **X-Forwarded** headers works in line with Quarkus configuration.

IMPORTANT

Activating this feature leaves the server exposed to several security issues (i.e. information spoofing). Consider activating it only when running behind a reverse proxy.

These are the relevant Quarkus properties and their effect on Quarkus CXF:

- **quarkus.http.proxy.proxy-address-forwarding** - the main switch to enable the rewriting of the request destination parts.
    - If enabled, the rewriting of the request fields will be effective throughout the whole CXF server stack.
    - If enabled, the values passed via **X-Forwarded-Proto** and **X-Forwarded-Port** headers will be used to set the protocol part and the port part of the URL returned by **jakarta.servlet.http.HttpServletRequest.getRequestURL()** respectively.
    - If enabled, the value passed via **X-Forwarded-For** will be returned by **jakarta.servlet.ServletRequest.getRemoteAddr()**.
- **quarkus.http.proxy.enable-forwarded-host** - enable the rewriting of the host part of URL returned by **jakarta.servlet.http.HttpServletRequest.getRequestURL()**. The actual host name is taken from the header configured via **quarkus.http.proxy.forwarded-host-header** (default is **X-Forwarded-Host**).
- **quarkus.http.proxy.enable-forwarded-prefix** - enable the rewriting of the path part of the URL returned by **jakarta.servlet.http.HttpServletRequest.getRequestURL()** and of the URI returned by **jakarta.servlet.http.HttpServletRequest.getRequestURI()**. The actual path prefix is taken from the header configured via **quarkus.http.proxy.forwarded-prefix-header** (default is **X-Forwarded-Prefix**).

Here is the most common snippet to copy to your **application.properties**:

```
quarkus.http.proxy.proxy-address-forwarding = true
quarkus.http.proxy.enable-forwarded-host = true
quarkus.http.proxy.enable-forwarded-prefix = true
```

One of the observable effects of these settings is the change of the location value in WSDL served on **http://localhost:8080/services/my-service?wsdl**. For example, if the request contains the following headers

```
X-Forwarded-Proto: https
X-Forwarded-Host: api.example.com
X-Forwarded-Port: 443
X-Forwarded-Prefix: /my-prefix
```

then the WSDL served on **http://localhost:8080/services/my-service?wsdl** would contain the following **location**:

```
...
<soap:address location="https://api.example.com:443/my-prefix/services/my-service"/>
...
```

6.13. CONTRACT FIRST AND CODE FIRST APPROACHES

Both contract first and code first development modes are fully supported by Quarkus CXF.

6.13.1. Contract first client

A SOAP service is described by WSDL. It is a contract defining operations, their parameters and return values, etc. WSDL is rich enough to be used for generating the code of a complete client. CXF provides the **wsdl2java** utility for that.

Quarkus CXF wraps **wsdl2java** in the **quarkus-cxf** extension so you do not need to use it directly.

Follow the Generate the Model classes from WSDL section of the user guide for more details about how to use it.

You may also want to check the CXF Developing a Consumer as a general introduction.

6.13.2. Contract first service

When implementing a service the generation of Java code from WSDL may also come in handy. **wsdl2java** can generate the model classes (with JAXB annotations) and service interfaces (with JAX-WS annotations) for you. Your task is then to provide implementations for those interfaces.

You may want to check the WSDL First Service Development section of CXF documentation for a better understanding of the underlying concepts.

6.13.3. Code first service

Another valid option at your disposal is to write your service in Java, using JAX-WS and JAXB. Then you have two options how to obtain the WSDL contract:

1. Start your service and point your clients at http://your-host/your-service?wsdl
2. Generate the WSDL document from Java classes at build time

TIP

Check the Code first development section of CXF documentation for further information.

6.14. GENERATE THE MODEL CLASSES FROM WSDL

quarkus-cxf extension supports generating Java classes from WSDL during Quarkus code generation phase.

CODE EXAMPLES

The code snippets shown in this section come from the client integration test in the source tree of Quarkus CXF. You may want to check it as an executable example.

You need to set up a couple of things for CXF code generation to work:

- Have **io.quarkiverse.cxf:quarkus-cxf** dependency in your project
- For Maven projects, the **generate-code** goal needs to be present in the configuration of **quarkus-maven-plugin**:

  pom.xml

  ```
  <plugin>
    <groupId>com.redhat.quarkus.platform</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <executions>
      <execution>
        <goals>
          <goal>build</goal>
          <goal>generate-code</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  ```

- For Gradle projects no additional configurarion of **io.quarkus** plugin is needed
- Put your WSDL files under **src/main/resources** or **src/test/resources** or any subdirectory thereof.
- Your WSDL file names must end with **.wsdl**
- Set **quarkus.cxf.codegen.wsdl2java.includes**configuration property to a pattern matching the WSDL files you wish to process. If you want to process all WSDL files under **src/main/resources/wsdl** or **src/test/resources/wsdl**, set it as follows:

  application.properties

  ```
  quarkus.cxf.codegen.wsdl2java.includes = wsdl/*.wsdl
  ```

This will generate Java classes in **target/generated-sources/wsdl2java** or **target/generated-test-sources/wsdl2java** directory. They will be automatically picked by the compiler plugin there. Hence we are free to refer to them from our application or test code.

Note that **quarkus-cxf** code generation uses the wsdl2Java utility from CXF under the hood. **wsdl2Java** is called separately for each WSDL file selected by **includes** and **excludes**.

6.14.1. Custom parameters

Passing custom parameters to **wsdl2java** is possible through **quarkus.cxf.codegen.wsdl2java.additional-params** configuration parameter.

6.14.2. Additional parameters

If you need different **additional-params** for each WSDL file, you may want to define a separate named parameter set for each one of them. Here is an example:

application.properties

```
# Parameters for foo.wsdl
quarkus.cxf.codegen.wsdl2java.foo-params.includes = wsdl/foo.wsdl
quarkus.cxf.codegen.wsdl2java.foo-params.wsdl-location = wsdl/foo.wsdl
# Parameters for bar.wsdl
quarkus.cxf.codegen.wsdl2java.bar-params.includes = wsdl/bar.wsdl
quarkus.cxf.codegen.wsdl2java.bar-params.wsdl-location = wsdl/bar.wsdl
quarkus.cxf.codegen.wsdl2java.bar-params.xjc = ts
```

TIP

Add **io.quarkiverse.cxf:quarkus-cxf-xjc-plugins** dependency to your project to be able to use **-xjc-Xbg**, **-xjc-Xdv**, **-xjc-Xjavadoc**, **-xjc-Xproperty-listener**, **-xjc-Xts** and **-xjc-Xwsdlextension** wsdl2java parameters.

6.15. GENERATE WSDL DOCUMENT FROM JAVA

If the WSDL served by your service at http://your-host/your-service?wsdl is not enough because you e.g. want to distribute it as a Maven artifact, then you can use **java2ws** to generate the WSDL document at build time.

You do not need to invoke the **java2ws** tool provided by CXF directly, or use the **cxf-java2ws-plugin**.

quarkus-cxf wraps **java2ws**, so you can configure it in **application.properties** as any other aspect of your application.

Here is an example:

application.properties

```
quarkus.cxf.java2ws.includes =
io.quarkiverse.cxf.it.server.HelloService,io.quarkiverse.cxf.it.server.FaultyHelloService
quarkus.cxf.java2ws.wsdl-name-template =
%TARGET_DIR%/Java2wsTest/%SIMPLE_CLASS_NAME%-from-java2ws.wsdl
```

Here we have instructed **java2ws** to generate WSDLs for two interfaces, namely **HelloService** and **FaultyHelloService**.

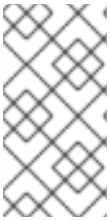
NOTE

The sample code snippets used in this section come from the server integration test in the source tree of Quarkus CXF

ANNOTATIONS

Note that the Service interfaces must be annotated with **jakarta.xml.ws.WebService** to be selected for **java2ws** processing.

The two generated WSDL documents will be stored as **target/Java2wsTest/FaultyHelloService-from-java2ws.wsdl** and **target/Java2wsTest/HelloService-from-java2ws.wsdl** respectively.



NOTE

Unlike **wsdl2java** which is executed within Quarkus source generation phase, **java2ws** is a part Quarkus augmentation that happens after compilation. The input of **java2ws** are, after all, Java class files. Hence you do not need to add **<goal>generate-code</goal>** to **quarkus-maven-plugin** for **java2ws**.

6.15.1. See also

- **quarkus.cxf.java2ws.*** configuration properties of **quarkus-cxf**

6.16. CXF INTERCEPTORS AND FEATURES

CXF interceptors and CXF features can be added to both your client or server using either annotations or **application.properties** configurations.

While CXF provides a number of out of the box embedded interceptors and features, you can also integrate your custom developed implementations.

Annotations can be used on either the service interface or implementor classes.

```
@org.apache.cxf.feature.Features (features = {"org.apache.cxf.ext.logging.LoggingFeature"})
@org.apache.cxf.interceptor.InInterceptors (interceptors = {"org.acme.Test1Interceptor" })
@org.apache.cxf.interceptor.InFaultInterceptors (interceptors = {"org.acme.Test2Interceptor" })
@org.apache.cxf.interceptor.OutInterceptors (interceptors = {"org.acme.Test1Interceptor" })
@org.apache.cxf.interceptor.InFaultInterceptors (interceptors =
{"org.acme.Test2Interceptor","org.acme.Test3Intercetpor" })
@WebService(endpointInterface = "org.acme.SayHi", targetNamespace = "uri:org.acme")
public class SayHiImplementation implements SayHi {
  public long sayHi(long arg) {
     return arg;
  }
  //...
}
```

You may also define your configurations in the **application.properties** file.

```
quarkus.cxf.endpoint."/greeting-service".features=org.apache.cxf.ext.logging.LoggingFeature
quarkus.cxf.endpoint."/greeting-service".in-interceptors=org.acme.Test1Interceptor
quarkus.cxf.endpoint."/greeting-service".out-interceptors=org.acme.Test1Interceptor
quarkus.cxf.endpoint."/greeting-service".in-fault-
interceptors=org.acme.Test2Interceptor,org.acme.Test3Intercetpor
quarkus.cxf.endpoint."/greeting-service".out-fault-interceptors=org.acme.Test1Intercetpor
```

CLASS LOADING

Both feature and interceptor classes are loaded via CDI first..

If no CDI beans are available, the constructor without parameters will be invoked to instantiate each class.

6.17. JAX-WS HANDLERS

As an alternative to the **@HandlerChain** annotation, JAX-WS Handlers can be added to both your client or server via **application.properties**:

application.properties

```
# A web service endpoint with multiple Handler classes
quarkus.cxf.endpoint."/greeting-
service".handlers=org.acme.MySOAPHandler,org.acme.AnotherSOAPHandler

# A web service client with a single Handler
quarkus.cxf.client."greeting-client".handlers=org.acme.MySOAPHandler
```

Where **MySOAPHandler** could look like below:

```
import jakarta.xml.ws.handler.soap.SOAPHandler;
import jakarta.xml.ws.handler.soap.SOAPMessageContext;

public class MySOAPHandler implements SOAPHandler<SOAPMessageContext> {

    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
        return true;
    }
    // other methods
}
```

CLASS LOADING

The **SOAPHandler** classes are loaded via CDI first..

If no CDI beans are available, the constructor without parameters will be invoked to instantiate each class.

6.18. JAX-WS PROVIDERS

JAX-WS Providers are fully supported, and can be implemented as shown below.

Given the following sample **Provider** implementation:

```
import jakarta.xml.transform.stream.StreamSource;
import jakarta.xml.ws.BindingType;
import jakarta.xml.ws.Provider;
import jakarta.xml.ws.Service;
import jakarta.xml.ws.ServiceMode;
import jakarta.xml.ws.WebServiceProvider;
import java.io.StringReader;

@WebServiceProvider
@ServiceMode(value = Service.Mode.PAYLOAD)
public class StreamSourcePayloadProvider implements Provider<StreamSource> {

    public StreamSourcePayloadProvider() {
    }

    public StreamSource invoke(StreamSource request) {
        String payload = StaxUtils.toString(request);

        // Do some interesting things ...

        StreamSource response = new StreamSource(new StringReader(payload));
        return response;
    }
}
```

The **application.properties** can be configured as shown below.

```
# A web service endpoint with the Provider implementation class
quarkus.cxf.endpoint."/stream-source".implementor=org.acme.StreamSourcePayloadProvider
```

CLASS LOADING

Provider classes are loaded via CDI first..

If no CDI beans are available, the constructor without parameters will be invoked to instantiate each class.

6.19. EXAMPLES

The **integration-tests** folder of the codebase provides various examples that demonstrate how to use this extension extensively.

6.20. COMMON PROBLEMS AND TROUBLESHOOTING

Some issues may appear during the development, testing, and native image building process of your **quarkus-cxf** project; below are some common ones and how to address them.

6.20.1. REST and SOAP Endpoints

Sometimes a REST endpoint may be needed in the same project where the Quarkus CXF Extension is used. The REST endpoint path must be different from the SOAP endpoint path (in order to avoid request forwarding conflicts between both protocols).

For example, if a WeatherWebService interface is declared in a WSDL, you can begin by creating the **org.acme.cxf.WeatherWebServiceImpl** class as follows:

```
package org.acme.cxf;

import ...

@Slf4j
@WebService(endpointInterface = "org.acme.cxf.WeatherWebService")
public class WeatherWebServiceImpl implements WeatherWebService {

    @Inject
    BackEndWeatherService backEndWeatherService;

    private Map<String, DailyTemperature> dailyTempByZipCode = Collections.synchronizedMap(new
LinkedHashMap<>());

    public WeatherWebServiceImpl() {
        this.dailyTempByZipCode.addAll(
          this.backEndWeatherService.getDailyForecast(Instant.now()));
    }

    @Override
    public DailyTemperature estimationTemperatures(String zipCode) {
        log.info("Daily estimation temperatures forecast called with '{}' zip code paramter", zipCode);
        return this.dailyTempByZipCode.get(zipCode);
    }
}
```

After that, you would need to specify the root context for your CXF web services, as indicated in the configuration documentation to split the REST (with RESTEasy for example) and SOAP routes based on their root context paths.

CXF's SOAP properties:

```
quarkus.cxf.path=/soap
quarkus.cxf.endpoint."/weather".implementor=org.acme.cxf.WeatherWebServiceImpl
```

Now, imagine the following RESTEasy endpoint:

```
package org.acme.reasteasy;

import ...

@Slf4j
@Path("/healthcheck")
public class HealthCheckResource {

 @Inject
    BackEndWeatherService backEndWeatherService;
```

```
 @GET
 public Response doHealthCheck() {
  if(this.backEndWeatherService.isAvailable()) {
          return Response.ok().build();
  } else {
          return Response.status(Response.Status.SERVICE_UNAVAILABLE);
  }
 }
}
```

You can separate your REST endpoint by configuring the REASTEasy path:

```
quarkus.resteasy.path=/rest
```

You can now send requests to both your REST and SOAP endpoints deployed within a single project, at:

- http://localhost:8080/rest/healthcheck for REST
- http://localhost:8080/soap/weather for SOAP

6.20.2. Non ASCII Characters

Sometimes the wsdl2java autogenerated Java classes may not be fully compatible with GraalVM due to non ASCII characters getting included in the code. Similar exceptions to the below may appear during native image builds.

```
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]      compile: 161 459,15 ms,  8,54 GB
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]        image: 158 272,73 ms,  8,43 GB
[quarkus-dalkia-ticket-loader-1.0.0-SNAPSHOT-runner:26]        write:     205,82 ms,  8,43 GB
Fatal error:com.oracle.svm.core.util.VMError$HostedError: java.lang.RuntimeException: oops :
expected ASCII string!
com.oracle.svm.reflect.OperationOrderStatusType_CRÉÉ_f151156b0d42ecdbdfb919501d8a86dda873
3012_1456.hashCode
 at com.oracle.svm.core.util.VMError.shouldNotReachHere(VMError.java:72)
```

Below is an example of auto-generated non ASCII characters in a Java class:

```
@XmlType(name = "OperationOrderStatusType")
@XmlEnum
public enum OperationOrderStatusType {

    @XmlEnumValue("Cr\u00e9\u00e9")
    CRÉÉ("Cr\u00e9\u00e9"),
    @XmlEnumValue("A communiquer")
    A_COMMUNIQUER("A communiquer"),
    @XmlEnumValue("En attente de r\u00e9ponse")
    EN_ATTENTE_DE_RÉPONSE("En attente de r\u00e9ponse"),
    @XmlEnumValue("Attribu\u00e9")
    ATTRIBUÉ("Attribu\u00e9"),
    @XmlEnumValue("Clotur\u00e9")
    CLOTURÉ("Clotur\u00e9"),
    @XmlEnumValue("Annul\u00e9")
    ANNULÉ("Annul\u00e9");
    private final String value;
```

```
    OperationOrderStatusType(String v) {
        value = v;
    }

    public String value() {
        return value;
    }

    public static OperationOrderStatusType fromValue(String v) {
        for (OperationOrderStatusType c: OperationOrderStatusType.values()) {
            if (c.value.equals(v)) {
                return c;
            }
        }
        throw new IllegalArgumentException(v);
    }
}
```

Anything starting with **u** will be a problem. Consequently the following refactoring is needed:

```
@XmlType(name = "OperationOrderStatusType")
@XmlEnum
public enum OperationOrderStatusType {

    @XmlEnumValue("Créé")
    CREE("Créé"),
    @XmlEnumValue("A communiquer")
    A_COMMUNIQUER("A communiquer"),
    @XmlEnumValue("En attente de réponse")
    EN_ATTENTE_DE_REPONSE("En attente de réponse"),
    @XmlEnumValue("Attribué")
    ATTRIBUE("Attribué"),
    @XmlEnumValue("Cloturé")
    CLOTURE("Cloturé"),
    @XmlEnumValue("Annulé")
    ANNULE("Annulé");
    private final String value;

    OperationOrderStatusType(String v) {
        value = v;
    }

    public String value() {
        return value;
    }

    public static OperationOrderStatusType fromValue(String v) {
        for (OperationOrderStatusType c: OperationOrderStatusType.values()) {
            if (c.value.equals(v)) {
                return c;
            }
        }
        throw new IllegalArgumentException(v);
    }
}
```

6.21. CAMEL INTEGRATION

https//camel.apache.org/camel-quarkus/latest/index.html[Camel Quarkus] supports CXF since its version 2.12.0. Under the hood, the implementation is based on Quarkus CXF. Therefore, all functionality available in Quarkus CXF is also available in Camel Quarkus.

Refer to https//camel.apache.org/camel-quarkus/latest/reference/extensions/cxf-soap.html[Camel Quarkus CXF SOAP] extension documentation for further details.

CHAPTER 7. CAMEL QUARKUS TRANSACTION GUIDE

7.1. ABOUT THE TRANSACTION GUIDE

This guide provides information and instructions for implementing transactional applications with Red Hat build of Apache Camel for Quarkus.

7.2. JTA DEPENDENCIES

In order to use **camel-quarkus-jta**, you need to include the following dependency in your **pom.xml**

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

This leverages **quarkus-narayana-jta** to provide JTA support in Camel.

7.2.1. Important configurations

There are some important **quarkus.transaction-manager** configurations you need to be aware of:

Configuration	Default value	Description	Comment
.node-name	**quarkus**	Identifes the node. It needs to be **unique** and **stable** over both transaction manager and container restarts.	For more information, see Configuring transaction node name identifier in the Quarkus documentation.
.object-store.type	**file-system**	Configures where the transaction logs are stored - either in a directory (**file-system**) or in a database (**jdbc**).	For a cloud environment without access to persistent volumes, consider using the **jdbc** object store. For more information, see Configure storing of Quarkus transaction logs in a database section in the Quarkus documentation
.enable-recovery	**false**	Enables recovery of pending transactions in case of a JVM crash or shutdown.	We recommend that you set it to **true** when you are using XA transactions in your application. Otherwise, pending transactions are lost in case of a JVM crash or shutdown.

Additional resources

For more details, see the Quarkus Transaction Guide.

7.3. CONFIGURING TRANSACTIONAL RESOURCES

7.3.1. JDBC Datasource configuration

To configure the datasource:

1. Include the relevant jdbc extension by following the Configure datasources in Quarkus section in the Quarkus datasource guide.

The extensions are integrated with the Quarkus agroal extension to support pooling and XA transactions.

Optional

2. If you want to use the datasource in a XA transaction, you must enable it:
   quarkus.datasource.jdbc.transactions = xa

   quarkus.datasource.<datasource-name>.jdbc.transactions = xa

   For more information, refer to the Narayana transaction manager integration section in the Quarkus datasource guide.

WARNING

Do not mix using non-XA and XA datasource in a transaction.

It is not a transaction safe.

7.3.2. JMS Configuration

In order to use JMS with distributed transactions, you must do the the following:

1. Use **quarkus-pooled-jms** to support pooling and transaction, by including the following dependency in your **pom.xml**:

```
<dependency>
    <groupId>io.quarkiverse.messaginghub</groupId>
    <artifactId>quarkus-pooled-jms</artifactId>
</dependency>
```

2. Enable XA support by setting **.transaction** to **xa**:
   quarkus.pooled-jms.transaction = xa

For more details, see Support for connection pooling and X/Open XA distributed transactions in the JMS extension documentation.

7.4. EXAMPLES

JPA and JMS

We use Narayana as the standalone JTA Transaction Manager implementation, and Hibernate as the JPA Adapter.
JMS and JPA: A Camel Quarkus example

Message Bridge

A basic REST endpoint is provided for users to dispatch a message to the IBM MQ queue. Messages from the IBM MQ are relayed to an ActiveMQ queue within an XA transaction. This example showcases the transaction functionality with rollback and recovery.
Message Bridge: A Camel Quarkus example

7.5. TRANSACTION POLICIES

There are six transaction policies:

Policy	Description	Comment
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.	Default
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.	

For more details, see Using different transaction propagations in the Quarkus Transactional client documentation.

7.6. KNOWN ISSUES

7.6.1. Non-XA datasource compatibility

Since 3.8.4, there is a compatibility break issue when using non-XA datasource. For more information see the pull request 40365