



Red Hat build of Apache Camel 4.0

Red Hat build of Apache Camel for Quarkus Reference

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

Red Hat build of Apache Camel 4.0 Red Hat build of Apache Camel for Quarkus Reference

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

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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	14
MAKING OPEN SOURCE MORE INCLUSIVE	14
CHAPTER 1. EXTENSIONS OVERVIEW	15
1.1. SUPPORT LEVEL DEFINITIONS	15
1.2. SUPPORTED EXTENSIONS	15
1.3. SUPPORTED LANGUAGES	26
1.4. SUPPORTED DATA FORMATS	27
CHAPTER 2. EXTENSIONS REFERENCE	30
2.1. AMQP	30
2.1.1. What's inside	30
2.1.2. Maven coordinates	30
2.1.3. Usage	30
2.1.3.1. Message mapping with org.w3c.dom.Node	30
2.1.3.2. Native mode support for jakarta.jms.ObjectMessage	30
2.1.3.3. Connection Pooling	30
2.1.4. transferException option in native mode	31
2.1.5. Additional Camel Quarkus configuration	31
2.2. ATTACHMENTS	31
2.2.1. What's inside	31
2.2.2. Maven coordinates	31
2.3. AVRO	31
2.3.1. What's inside	31
2.3.2. Maven coordinates	32
2.3.3. Additional Camel Quarkus configuration	32
2.4. AWS 2 CLOUDWATCH	32
2.4.1. What's inside	32
2.4.2. Maven coordinates	32
2.4.3. SSL in native mode	33
2.5. AWS 2 DYNAMODB	33
2.5.1. What's inside	33
2.5.2. Maven coordinates	33
2.5.3. SSL in native mode	33
2.5.4. Additional Camel Quarkus configuration	33
2.5.4.1. Optional integration with Quarkus Amazon DynamoDB	33
2.6. AWS 2 KINESIS	34
2.6.1. What's inside	34
2.6.2. Maven coordinates	34
2.6.3. SSL in native mode	34
2.7. AWS 2 LAMBDA	34
2.7.1. What's inside	34
2.7.2. Maven coordinates	35
2.7.3. SSL in native mode	35
2.7.4. Additional Camel Quarkus configuration	35
2.7.4.1. Not possible to leverage quarkus-amazon-lambda by Camel aws2-lambda extension	35
2.8. AWS 2 S3 STORAGE SERVICE	35
2.8.1. What's inside	35
2.8.2. Maven coordinates	35
2.8.3. SSL in native mode	35
2.8.4. Additional Camel Quarkus configuration	35

2.8.4.1. Optional integration with Quarkus Amazon S3	36
2.9. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)	36
2.9.1. What's inside	36
2.9.2. Maven coordinates	36
2.9.3. SSL in native mode	36
2.9.4. Additional Camel Quarkus configuration	36
2.9.4.1. Optional integration with Quarkus Amazon SNS	37
2.10. AWS 2 SIMPLE QUEUE SERVICE (SQS)	37
2.10.1. What's inside	37
2.10.2. Maven coordinates	37
2.10.3. SSL in native mode	37
2.10.4. Additional Camel Quarkus configuration	37
2.10.4.1. Optional integration with Quarkus Amazon SQS	38
2.11. AZURE SERVICEBUS	38
2.11.1. What's inside	38
2.11.2. Maven coordinates	38
2.12. AZURE STORAGE BLOB SERVICE	38
2.12.1. What's inside	38
2.12.2. Maven coordinates	39
2.12.3. Usage	39
2.12.3.1. Micrometer metrics support	39
2.12.4. SSL in native mode	39
2.13. AZURE STORAGE QUEUE SERVICE	39
2.13.1. What's inside	39
2.13.2. Maven coordinates	39
2.13.3. Usage	40
2.13.3.1. Micrometer metrics support	40
2.13.4. SSL in native mode	40
2.14. BEAN VALIDATOR	40
2.14.1. What's inside	40
2.14.2. Maven coordinates	40
2.14.3. Usage	40
2.14.3.1. Configuring the ValidatorFactory	40
2.14.3.2. Custom validation groups in native mode	41
2.14.4. Camel Quarkus limitations	41
2.15. BEAN	41
2.15.1. What's inside	41
2.15.2. Maven coordinates	41
2.15.3. Usage	41
2.16. BINDY	41
2.16.1. What's inside	42
2.16.2. Maven coordinates	42
2.16.3. Camel Quarkus limitations	42
2.17. BROWSE	42
2.17.1. What's inside	42
2.17.2. Maven coordinates	42
2.18. CASSANDRA CQL	43
2.18.1. What's inside	43
2.18.2. Maven coordinates	43
2.18.3. Additional Camel Quarkus configuration	43
2.18.3.1. Cassandra aggregation repository in native mode	43
2.19. CLI CONNECTOR	43
2.19.1. What's inside	43

2.19.2. Maven coordinates	43
2.20. CONTROL BUS	44
2.20.1. What's inside	44
2.20.2. Maven coordinates	44
2.20.3. Usage	44
2.20.3.1. Languages	44
2.20.3.1.1. Bean	44
2.20.3.1.2. Simple	44
2.20.4. Camel Quarkus limitations	45
2.20.4.1. Statistics	45
2.21. CORE	45
2.21.1. What's inside	45
2.21.2. Maven coordinates	45
2.21.3. Additional Camel Quarkus configuration	45
2.21.3.1. Simple language	46
2.21.3.1.1. Using the OGNL notation	46
2.21.3.1.2. Using dynamic type resolution in native mode	46
2.21.3.1.3. Using the simple language with classpath resources in native mode	46
2.21.3.1.4. Configuring a custom bean via properties in native mode	46
2.22. CRON	55
2.22.1. What's inside	55
2.22.2. Maven coordinates	55
2.22.3. Additional Camel Quarkus configuration	55
2.23. CRYPTO (JCE)	55
2.23.1. What's inside	55
2.23.2. Maven coordinates	55
2.23.3. SSL in native mode	56
2.24. CXF	56
2.24.1. What's inside	56
2.24.2. Maven coordinates	56
2.24.3. Usage	56
2.24.3.1. General	56
2.24.3.2. Dependency management	56
2.24.3.3. Client	56
2.24.3.4. Server	58
2.24.3.5. Logging of requests and responses	59
2.24.3.6. WS Specifications	60
2.24.3.7. Tooling	61
2.25. DATA FORMAT	61
2.25.1. What's inside	61
2.25.2. Maven coordinates	61
2.26. DATASET	62
2.26.1. What's inside	62
2.26.2. Maven coordinates	62
2.27. DIRECT	62
2.27.1. What's inside	62
2.27.2. Maven coordinates	62
2.28. FHIR	62
2.28.1. What's inside	63
2.28.2. Maven coordinates	63
2.28.3. SSL in native mode	63
2.28.4. Additional Camel Quarkus configuration	63
2.29. FILE	64

2.29.1. What's inside	64
2.29.2. Maven coordinates	65
2.29.3. Additional Camel Quarkus configuration	65
2.29.3.1. Having only a single consumer in a cluster consuming from a given endpoint	65
2.30. FTP	67
2.30.1. What's inside	67
2.30.2. Maven coordinates	67
2.31. GOOGLE BIGQUERY	67
2.31.1. What's inside	68
2.31.2. Maven coordinates	68
2.31.3. Usage	68
2.32. GOOGLE PUBSUB	68
2.32.1. What's inside	68
2.32.2. Maven coordinates	68
2.32.3. Camel Quarkus limitations	68
2.33. GRPC	69
2.33.1. What's inside	69
2.33.2. Maven coordinates	69
2.33.3. Usage	69
2.33.3.1. Protobuf generated code	69
2.33.3.1.1. Scanning proto files with imports	70
2.33.3.1.2. Scanning proto files from dependencies	70
2.33.3.2. Accessing classpath resources in native mode	70
2.33.4. Camel Quarkus limitations	71
2.33.4.1. Integration with Quarkus gRPC is not supported	71
2.33.5. Additional Camel Quarkus configuration	71
2.34. GSON	73
2.34.1. What's inside	73
2.34.2. Maven coordinates	73
2.34.3. Additional Camel Quarkus configuration	74
2.34.3.1. Marshaling/Unmarshaling objects in native mode	74
2.35. HL7	74
2.35.1. What's inside	74
2.35.2. Maven coordinates	74
2.35.3. Camel Quarkus limitations	74
2.36. HTTP	74
2.36.1. What's inside	74
2.36.2. Maven coordinates	75
2.36.3. SSL in native mode	75
2.36.4. Additional Camel Quarkus configuration	75
2.37. INFINISPAN	75
2.37.1. What's inside	75
2.37.2. Maven coordinates	75
2.37.3. Additional Camel Quarkus configuration	75
2.37.3.1. Infinispan Client Configuration	75
2.37.3.2. Camel Infinispan InfinispanRemoteAggregationRepository in native mode	76
2.38. AVRO JACKSON	76
2.38.1. What's inside	76
2.38.2. Maven coordinates	76
2.39. PROTOBUF JACKSON	76
2.39.1. What's inside	76
2.39.2. Maven coordinates	77
2.40. JACKSON	77

2.40.1. What's inside	77
2.40.2. Maven coordinates	77
2.40.3. Usage	77
2.40.3.1. Configuring the Jackson ObjectMapper	77
2.40.3.1.1. ObjectMapper created internally by JacksonDataFormat	77
2.40.3.1.2. Custom ObjectMapper for JacksonDataFormat	77
2.40.3.1.3. Using the Quarkus Jackson ObjectMapper with JacksonDataFormat	78
2.41. JACKSONXML	79
2.41.1. What's inside	79
2.41.2. Maven coordinates	79
2.42. JAVA JOOR DSL	79
2.42.1. What's inside	79
2.42.2. Maven coordinates	79
2.42.3. Camel Quarkus limitations	80
2.43. JAXB	80
2.43.1. What's inside	80
2.43.2. Maven coordinates	80
2.43.3. Usage	80
2.43.3.1. Native mode ObjectFactory instantiation of non-JAXB annotated classes	80
2.44. JDBC	80
2.44.1. What's inside	81
2.44.2. Maven coordinates	81
2.44.3. Additional Camel Quarkus configuration	81
2.44.3.1. Configuring a DataSource	81
2.44.3.1.1. Zero configuration with Quarkus Dev Services	81
2.45. JIRA	81
2.45.1. What's inside	81
2.45.2. Maven coordinates	81
2.45.3. SSL in native mode	82
2.46. JMS	82
2.46.1. What's inside	82
2.46.2. Maven coordinates	82
2.46.3. Usage	82
2.46.3.1. Message mapping with org.w3c.dom.Node	82
2.46.3.2. Native mode support for jakarta.jms.ObjectMessage	82
2.46.3.3. Support for Connection pooling and X/Open XA distributed transactions	83
2.47. JPA	84
2.47.1. What's inside	84
2.47.2. Maven coordinates	84
2.47.3. Additional Camel Quarkus configuration	84
2.47.3.1. Configuring JpaMessageIdRepository	84
2.48. JSLT	85
2.48.1. What's inside	85
2.48.2. Maven coordinates	85
2.48.3. allowContextMapAll option in native mode	85
2.48.4. Additional Camel Quarkus configuration	85
2.48.4.1. Loading JSLT templates from classpath in native mode	85
2.48.4.2. Using JSLT functions in native mode	86
2.49. JSON PATH	86
2.49.1. What's inside	86
2.49.2. Maven coordinates	86
2.50. JTA	86
2.50.1. What's inside	87

2.50.2. Maven coordinates	87
2.50.3. Usage	87
2.51. KAFKA	88
2.51.1. What's inside	88
2.51.2. Maven coordinates	88
2.51.3. Usage	88
2.51.3.1. Quarkus Kafka Dev Services	88
2.51.4. Additional Camel Quarkus configuration	88
2.52. KAMELET	89
2.52.1. What's inside	89
2.52.2. Maven coordinates	89
2.52.3. Usage	89
2.52.3.1. Pre-load Kamelets at build-time	89
2.52.3.2. Using the Kamelet Catalog	89
2.52.4. Additional Camel Quarkus configuration	90
2.53. KUBERNETES	90
2.53.1. Maven coordinates	90
2.53.2. Additional Camel Quarkus configuration	90
2.53.2.1. Automatic registration of a Kubernetes Client instance	90
2.53.2.2. Having only a single consumer in a cluster consuming from a given endpoint	91
2.54. LANGUAGE	97
2.54.1. What's inside	97
2.54.2. Maven coordinates	97
2.54.3. Usage	98
2.54.3.1. Required Dependencies	98
2.54.3.2. Native Mode	98
2.54.4. allowContextMapAll option in native mode	98
2.55. LDAP	98
2.55.1. What's inside	98
2.55.2. Maven coordinates	98
2.55.3. Usage	99
2.55.3.1. Using SSL in Native Mode	99
2.56. LOG	99
2.56.1. What's inside	99
2.56.2. Maven coordinates	99
2.57. MAIL	99
2.57.1. What's inside	99
2.57.2. Maven coordinates	100
2.58. MANAGEMENT	100
2.58.1. Maven coordinates	100
2.58.2. Usage	100
2.58.2.1. Enabling and Disabling JMX	100
2.58.2.2. Native mode	100
2.59. MAPSTRUCT	101
2.59.1. What's inside	101
2.59.2. Maven coordinates	101
2.59.3. Usage	101
2.59.3.1. Annotation Processor	101
2.59.3.1.1. Maven	101
2.59.3.1.2. Gradle	102
2.59.3.2. Mapper definition discovery	102
2.60. MASTER	102
2.60.1. What's inside	102

2.60.2. Maven coordinates	102
2.60.3. Additional Camel Quarkus configuration	102
2.61. MICROMETER	102
2.61.1. What's inside	103
2.61.2. Maven coordinates	103
2.61.3. Usage	103
2.61.4. Camel Quarkus limitations	103
2.61.4.1. Exposing Micrometer statistics in JMX	103
2.61.4.2. Decrement header for Counter is ignored by Prometheus	103
2.61.4.3. Exposing statistics in JMX	103
2.61.5. Additional Camel Quarkus configuration	103
2.62. MICROPROFILE FAULT TOLERANCE	104
2.62.1. What's inside	105
2.62.2. Maven coordinates	105
2.63. MICROPROFILE HEALTH	105
2.63.1. What's inside	105
2.63.2. Maven coordinates	105
2.63.3. Usage	105
2.63.3.1. Provided health checks	105
2.63.3.1.1. Camel Context Health	106
2.63.3.1.2. Camel Route Health	106
2.63.4. Additional Camel Quarkus configuration	106
2.64. MINIO	106
2.64.1. What's inside	106
2.64.2. Maven coordinates	106
2.64.3. Additional Camel Quarkus configuration	107
2.65. MLLP	107
2.65.1. What's inside	107
2.65.2. Maven coordinates	107
2.65.3. Additional Camel Quarkus configuration	107
2.66. MOCK	107
2.66.1. What's inside	107
2.66.2. Maven coordinates	108
2.66.3. Usage	108
2.66.4. Camel Quarkus limitations	109
2.67. MONGODB	109
2.67.1. What's inside	109
2.67.2. Maven coordinates	109
2.67.3. Additional Camel Quarkus configuration	109
2.68. MYBATIS	110
2.68.1. What's inside	110
2.68.2. Maven coordinates	110
2.68.3. Additional Camel Quarkus configuration	110
2.69. NETTY HTTP	111
2.69.1. What's inside	111
2.69.2. Maven coordinates	111
2.69.3. transferException option in native mode	111
2.69.4. Additional Camel Quarkus configuration	111
2.70. NETTY	111
2.70.1. What's inside	111
2.70.2. Maven coordinates	112
2.71. OPENAPI JAVA	112
2.71.1. What's inside	112

2.71.2. Maven coordinates	112
2.71.3. Usage	112
2.72. OPENTELEMETRY	113
2.72.1. What's inside	113
2.72.2. Maven coordinates	113
2.72.3. Usage	113
2.72.3.1. Exporters	114
2.72.3.2. Tracing CDI bean method execution	114
2.72.4. Additional Camel Quarkus configuration	114
2.73. PAHO MQTT5	115
2.73.1. What's inside	115
2.73.2. Maven coordinates	115
2.74. PAHO	115
2.74.1. What's inside	116
2.74.2. Maven coordinates	116
2.75. PLATFORM HTTP	116
2.75.1. What's inside	116
2.75.2. Maven coordinates	116
2.75.3. Usage	116
2.75.3.1. Basic Usage	116
2.75.3.2. Using platform-http via Camel REST DSL	117
2.75.3.3. Handling multipart/form-data file uploads	117
2.75.3.4. Securing platform-http endpoints	117
2.75.3.5. Implementing a reverse proxy	118
2.75.4. Additional Camel Quarkus configuration	118
2.75.4.1. Platform HTTP server configuration	118
2.75.4.2. Character encodings	118
2.76. QUARTZ	118
2.76.1. What's inside	118
2.76.2. Maven coordinates	118
2.76.3. Usage	119
2.77. REF	119
2.77.1. What's inside	119
2.77.2. Maven coordinates	119
2.77.3. Usage	119
2.78. REST OPENAPI	120
2.78.1. What's inside	120
2.78.2. Maven coordinates	120
2.78.3. Usage	120
2.78.3.1. Required Dependencies	120
2.79. REST	121
2.79.1. What's inside	121
2.79.2. Maven coordinates	121
2.79.3. Additional Camel Quarkus configuration	121
2.79.3.1. Path parameters containing special characters with platform-http	121
2.79.3.2. Configuring alternate REST transport providers	122
2.80. SALESFORCE	122
2.80.1. What's inside	122
2.80.2. Maven coordinates	122
2.80.3. Usage	123
2.80.3.1. Generating Salesforce DTOs with the salesforce-maven-plugin	123
2.80.4. SSL in native mode	123
2.81. SAP	123

2.81.1. What's inside	123
2.81.2. Maven coordinates	124
2.81.2.1. Additional platform restrictions for the SAP component	124
2.81.2.2. SAP JCo and SAP IDoc libraries	124
2.81.3. URI format	124
2.81.3.1. Options for RFC destination endpoints	125
2.81.3.2. Options for RFC server endpoints	126
2.81.3.3. Options for the IDoc List Server endpoint	126
2.81.3.4. Summary of the RFC and IDoc endpoints	126
2.81.3.5. SAP RFC destination endpoint	128
2.81.3.6. SAP RFC server endpoint	128
2.81.3.7. SAP IDoc and IDoc list destination endpoints	129
2.81.3.8. SAP IDoc list server endpoint	129
2.81.3.9. Metadata repositories	129
2.81.4. Configuration	130
2.81.4.1. Configuration Overview	130
2.81.4.2. Destination Configuration	131
2.81.4.2.1. Interceptor for tRFC and qRFC destinations	132
2.81.4.2.2. Log on and authentication options	132
2.81.4.2.3. Connection options	133
2.81.4.2.4. Connection pool options	136
2.81.4.2.5. Secure network connection options	136
2.81.4.2.6. Repository options	137
2.81.4.2.7. Trace configuration options	138
2.81.4.3. Server Configuration	138
2.81.4.3.1. Required options	140
2.81.4.3.2. Secure network connection options	140
2.81.4.3.3. Other options	141
2.81.4.4. Repository Configuration	142
2.81.4.4.1. Function template properties	142
2.81.4.4.2. List field metadata properties	144
2.81.4.4.3. Record metadata properties	145
2.81.4.4.4. Record field metadata properties	146
2.81.5. Message Headers	148
2.81.6. Exchange Properties	149
2.81.7. Message Body for RFC	149
2.81.7.1. Request and response objects	149
2.81.7.2. Structure objects	150
2.81.7.3. Field types	150
2.81.7.3.1. Elementary field types	151
2.81.7.3.2. Character field types	152
2.81.7.3.3. Numeric field types	153
2.81.7.3.4. Hexadecimal field types	153
2.81.7.3.5. String field types	154
2.81.7.3.6. Complex field types	154
2.81.7.3.7. Structure field types	154
2.81.7.3.8. Table field types	154
2.81.7.3.9. Table objects	155
2.81.8. Message Body for IDoc	155
2.81.8.1. IDoc message type	155
2.81.8.2. The IDoc document model	156
2.81.8.3. How an IDoc is related to a Document object	158
2.81.9. Document attributes	159

2.81.9.1. Setting document attributes in Java	161
2.81.9.2. Setting document attributes in XML	161
2.81.10. Transaction Support	162
2.81.10.1. BAPI transaction model	162
2.81.10.2. RFC transaction model	162
2.81.10.3. Which transaction model to use?	162
2.81.10.4. Transactional RFC destination endpoints	162
2.81.10.5. Transactional RFC server endpoints	163
2.81.11. XML Serialization for RFC	163
2.81.11.1. XML namespace	163
2.81.11.2. Request and response XML documents	163
2.81.11.3. Structure fields	163
2.81.11.4. Table fields	164
2.81.11.5. Elementary fields	165
2.81.11.6. Date and time formats	165
2.81.12. XML Serialization for IDoc	165
2.81.12.1. XML namespace	166
2.81.12.2. Built-in type converter	166
2.81.12.3. Sample IDoc message body in XML format	166
2.81.13. Example 1: Reading Data from SAP	167
2.81.13.1. Java DSL for route	167
2.81.13.2. XML DSL for route	167
2.81.13.3. createFlightCustomerGetListRequest bean	167
2.81.13.4. returnFlightCustomerInfo bean	168
2.81.14. Example 2: Writing Data to SAP	169
2.81.14.1. Java DSL for route	169
2.81.14.2. XML DSL for route	169
2.81.14.3. Transaction support	170
2.81.14.4. Populating request parameters	170
2.81.15. Example 3: Handling Requests from SAP	170
2.81.15.1. Java DSL for route	170
2.81.15.2. XML DSL for route	170
2.81.15.3. BookFlightRequest bean	171
2.81.15.4. BookFlightResponse bean	172
2.81.15.5. FlightInfo bean	172
2.81.15.6. ConnectionInfoTable bean	173
2.81.15.7. ConnectionInfo bean	173
2.82. XQUERY	174
2.82.1. What's inside	174
2.82.2. Maven coordinates	174
2.82.3. Additional Camel Quarkus configuration	175
2.83. SCHEDULER	175
2.83.1. What's inside	175
2.83.2. Maven coordinates	175
2.84. SEDA	175
2.84.1. What's inside	175
2.84.2. Maven coordinates	176
2.85. SLACK	176
2.85.1. What's inside	176
2.85.2. Maven coordinates	176
2.85.3. SSL in native mode	176
2.86. SNMP	176
2.86.1. What's inside	176

2.86.2. Maven coordinates	176
2.87. SOAP DATAFORMAT	177
2.87.1. What's inside	177
2.87.2. Maven coordinates	177
2.88. SPLUNK	177
2.88.1. What's inside	177
2.88.2. Maven coordinates	177
2.88.3. SSL in native mode	177
2.89. SQL	178
2.89.1. What's inside	178
2.89.2. Maven coordinates	178
2.89.3. Additional Camel Quarkus configuration	178
2.89.3.1. Configuring a DataSource	178
2.89.3.1.1. Zero configuration with Quarkus Dev Services	178
2.89.3.2. SQL scripts	178
2.89.3.3. SQL aggregation repository in native mode	179
2.90. TELEGRAM	179
2.90.1. What's inside	179
2.90.2. Maven coordinates	179
2.90.3. Usage	179
2.90.4. Webhook Mode	179
2.90.5. SSL in native mode	179
2.91. TIMER	179
2.91.1. What's inside	180
2.91.2. Maven coordinates	180
2.92. VALIDATOR	180
2.92.1. What's inside	180
2.92.2. Maven coordinates	180
2.93. VELOCITY	180
2.93.1. What's inside	180
2.93.2. Maven coordinates	180
2.93.3. Usage	181
2.93.3.1. Custom body as domain object in the native mode	181
2.93.4. allowContextMapAll option in native mode	181
2.93.5. Additional Camel Quarkus configuration	181
2.94. VERT.X HTTP CLIENT	181
2.94.1. What's inside	181
2.94.2. Maven coordinates	182
2.94.3. transferException option in native mode	182
2.94.4. Additional Camel Quarkus configuration	182
2.94.5. allowJavaSerializedObject option in native mode	182
2.94.5.1. Character encodings	182
2.95. VERT.X WEBSOCKET	182
2.95.1. What's inside	182
2.95.2. Maven coordinates	182
2.95.3. Usage	183
2.95.3.1. Vert.x WebSocket consumers	183
2.95.3.2. Vert.x WebSocket producers	183
2.95.4. Additional Camel Quarkus configuration	183
2.95.4.1. Vert.x WebSocket server configuration	183
2.95.4.2. Character encodings	184
2.96. XML IO DSL	184
2.96.1. What's inside	184

2.96.2. Maven coordinates	184
2.96.3. Additional Camel Quarkus configuration	184
2.96.3.1. XML file encodings	184
2.97. XML JAXP	184
2.97.1. Maven coordinates	184
2.98. XPATH	185
2.98.1. What's inside	185
2.98.2. Maven coordinates	185
2.98.3. Additional Camel Quarkus configuration	185
2.99. XSLT SAXON	185
2.99.1. What's inside	185
2.99.2. Maven coordinates	185
2.100. XSLT	186
2.100.1. What's inside	186
2.100.2. Maven coordinates	186
2.100.3. Additional Camel Quarkus configuration	186
2.100.3.1. Configuration	187
2.100.3.2. Extension functions support	187
2.101. YAML DSL	188
2.101.1. What's inside	188
2.101.2. Maven coordinates	188
2.101.3. Usage	188
2.101.3.1. Native mode	188
2.101.3.1.1. Bean definitions	189
2.101.3.1.2. Exception handling	189
2.102. ZIP DEFLATE COMPRESSION	190
2.102.1. What's inside	190
2.102.2. Maven coordinates	190
2.103. ZIP FILE	191
2.103.1. What's inside	191
2.103.2. Maven coordinates	191

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE


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

CHAPTER 1. EXTENSIONS OVERVIEW

1.1. SUPPORT LEVEL DEFINITIONS

New features, services, and components go through a number of support levels before inclusion in 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	<p>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.</p> <div>  <p>NOTE</p> <p>Community extensions are listed on the extensions reference page of the Camel Quarkus community project.</p> </div>
Technology Preview	<p>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.</p>
Production Support	<p>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.</p>

1.2. SUPPORTED EXTENSIONS

There are 87 extensions.

Table 1.2. Red Hat build of Apache Camel for Quarkus Support Matrix Extensions

Extension	Artifact	JVM Support Level	Native Support Level	Description
Amqp	camel-quarkus-amqp	Production Support	Production Support	Messaging with AMQP protocol using Apache QPid Client.
Attachments	camel-quarkus-attachments	Production Support	Production Support	Support for attachments on Camel messages
Aws2-cw	camel-quarkus-aws2-cw	Production Support	Production Support	Sending metrics to AWS CloudWatch using AWS SDK version 2.x.
Aws2-ddb	camel-quarkus-aws2-ddb	Production Support	Production Support	Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream using AWS SDK version 2.x.
Aws2-kinesis	camel-quarkus-aws2-kinesis	Production Support	Production Support	Consume and produce records from AWS Kinesis Streams using AWS SDK version 2.x.
Aws2-lambda	camel-quarkus-aws2-lambda	Production Support	Production Support	Manage and invoke AWS Lambda functions using AWS SDK version 2.x.
Aws2-s3	camel-quarkus-aws2-s3	Production Support	Production Support	Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.
Aws2-sns	camel-quarkus-aws2-sns	Production Support	Production Support	Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Aws2-sqs	camel-quarkus-aws2-sqs	Production Support	Production Support	Send and receive messages to/from AWS SQS service using AWS SDK version 2.x.
Azure-servicebus	camel-quarkus-azure-servicebus	Technology Preview	None	Send and receive messages to/from Azure Service Bus.
Azure-storage-blob	camel-quarkus-azure-storage-blob	Technology Preview	Technology Preview	Store and retrieve blobs from Azure Storage Blob Service using SDK v12.
Azure-storage-queue	camel-quarkus-azure-storage-queue	Technology Preview	Technology Preview	The azure-storage-queue component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.
Bean	camel-quarkus-bean	Production Support	Production Support	Invoke methods of Java beans
Bean-validator	camel-quarkus-bean-validator	Production Support	Production Support	Validate the message body using the Java Bean Validation API.
Browse	camel-quarkus-browse	Production Support	Production Support	Inspect the messages received on endpoints supporting <code>BrowsableEndpoint</code> .

Extension	Artifact	JVM Support Level	Native Support Level	Description
Cassandraql	camel-quarkus-cassandraql	Production Support	Production Support	Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.
CLI-connector	camel-quarkus-cli-connector	Production Support	Production Support	Runtime adapter connecting with Camel CLI
Controlbus	camel-quarkus-controlbus	Production Support	Production Support	Manage and monitor Camel routes.
Core	camel-quarkus-core	Production Support	Production Support	Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Simple and Tokenize
Crypto	camel-quarkus-crypto	Production Support	Production Support	Sign and verify exchanges using the Signature Service of the Java Cryptographic Extension (JCE).
Cron	camel-quarkus-cron	Production Support	Production Support	A generic interface for triggering events at times specified through the Unix cron syntax.
CXF-soap	camel-quarkus-cxf-soap	Production Support	Production Support	Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Dataformat	camel-quarkus-dataformat	Production Support	Production Support	Use a Camel Data Format as a regular Camel Component.
Dataset	camel-quarkus-dataset	devSupport	devSupport	Provide data for load and soak testing of your Camel application.
Direct	camel-quarkus-direct	Production Support	Production Support	Call another endpoint from the same Camel Context synchronously.
Fhir	camel-quarkus-fhir	Production Support	Production Support	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.
File	camel-quarkus-file	Production Support	Production Support	Read and write files.
Ftp	camel-quarkus-ftp	Production Support	Production Support	Upload and download files to/from SFTP, FTP or SFTP servers
Google-bigquery	camel-quarkus-google-bigquery	Production Support	Production Support	Access Google Cloud BigQuery service using SQL queries or Google Client Services API

Extension	Artifact	JVM Support Level	Native Support Level	Description
Google-pubsub	camel-quarkus-google-pubsub	Production Support	Production Support	Send and receive messages to/from Google Cloud Platform PubSub Service.
Grpc	camel-quarkus-grpc	Production Support	Production Support	Expose gRPC endpoints and access external gRPC endpoints.
Http	camel-quarkus-http	Production Support	Production Support	Send requests to external HTTP servers using Apache HTTP Client 5.x.
Infinispan	camel-quarkus-infinispan	Production Support	Production Support	Read and write from/to Infinispan distributed key/value store and data grid.
Java-joor-dsl	camel-quarkus-java-joor-dsl	community	community	Support for parsing Java route definitions at runtime
Jdbc	camel-quarkus-jdbc	Production Support	Production Support	Access databases through SQL and JDBC.
Jira	camel-quarkus-jira	Production Support	Production Support	Interact with JIRA issue tracker.
Jms	camel-quarkus-jms	Production Support	Production Support	Send and receive messages to/from a JMS Queue or Topic.
Jpa	camel-quarkus-jpa	Production Support	Production Support	Store and retrieve Java objects from databases using Java Persistence API (JPA).

Extension	Artifact	JVM Support Level	Native Support Level	Description
Jta	camel-quarkus-jta	Production Support	Production Support	Enclose Camel routes in transactions using Java Transaction API (JTA) and Narayana transaction manager
Kafka	camel-quarkus-kafka	Production Support	Production Support	Sent and receive messages to/from an Apache Kafka broker.
Kamelet	camel-quarkus-kamelet	Production Support	Production Support	Materialize route templates
Kubernetes	camel-quarkus-kubernetes	Technology Preview	Technology Preview	Perform operations against Kubernetes API
Language	camel-quarkus-language	Production Support	Production Support	Execute scripts in any of the languages supported by Camel.
Ldap	camel-quarkus-ldap	Production Support	Production Support	Perform searches on LDAP servers.
Log	camel-quarkus-log	Production Support	Production Support	Log messages to the underlying logging mechanism.
Mail	camel-quarkus-mail	Production Support	Production Support	Send and receive emails using imap, pop3 and smtp protocols. Marshal Camel messages with attachments into MIME-Multipart messages and back.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Management	camel-quarkus-management	Production Support	Production Support	JMX management strategy and associated managed resources.
Mapstruct	camel-quarkus-mapstruct	Production Support	Production Support	Type Conversion using Mapstruct
Master	camel-quarkus-master	Production Support	Production Support	Have only a single consumer in a cluster consuming from a given endpoint; with automatic failover if the JVM dies.
Micrometer	camel-quarkus-micrometer	Production Support	Production Support	Collect various metrics directly from Camel routes using the Micrometer library.
Microprofile-fault-tolerance	camel-quarkus-microprofile-fault-tolerance	Production Support	Production Support	Circuit Breaker EIP using Microprofile Fault Tolerance
Microprofile-health	camel-quarkus-microprofile-health	Production Support	Production Support	Expose Camel health checks via MicroProfile Health
Minio	camel-quarkus-minio	Production Support	Production Support	Store and retrieve objects from Minio Storage Service using Minio SDK.
Mllp	camel-quarkus-mllp	Production Support	Production Support	Communicate with external systems using the MLLP protocol.
Mybatis	camel-quarkus-mybatis	Production Support	Production Support	Performs a query, poll, insert, update or delete in a relational database using MyBatis.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Mock	camel-quarkus-mock	Production Support	Production Support	Test routes and mediation rules using mocks.
Mongodb	camel-quarkus-mongodb	Technology Preview	Technology Preview	Perform operations on MongoDB documents and collections.
Netty	camel-quarkus-netty	Production Support	Production Support	Socket level networking using TCP or UDP with Netty 4.x.
Netty-http	camel-quarkus-netty-http	Production Support	Production Support	Netty HTTP server and client using the Netty 4.x.
Openapi-java	camel-quarkus-openapi-java	Production Support	Production Support	Expose OpenAPI resources defined in Camel REST DSL
OpenTelemetry	camel-quarkus-opentelemetry	Technology Preview	Technology Preview	Distributed tracing using OpenTelemetry
Quartz	camel-quarkus-quartz	Production Support	Production Support	Schedule sending of messages using the Quartz 2.x scheduler.
Paho	camel-quarkus-paho	Production Support	Production Support	Communicate with MQTT message brokers using Eclipse Paho MQTT Client.
Paho-mqtt5	camel-quarkus-paho-mqtt5	Production Support	Production Support	Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Platform-http	camel-quarkus-platform-http	Production Support	Production Support	Expose HTTP endpoints using the HTTP server available in the current platform.
Ref	camel-quarkus-ref	Production Support	Production Support	Route messages to an endpoint looked up dynamically by name in the Camel Registry.
Rest	camel-quarkus-rest	Production Support	Production Support	Expose REST services and their OpenAPI Specification or call external REST services.
Rest-openapi	camel-quarkus-rest-openapi	Production Support	Production Support	Configure REST producers based on an OpenAPI specification document delegating to a component implementing the RestProducerFactory interface.
Salesforce	camel-quarkus-salesforce	Production Support	Production Support	Communicate with Salesforce using Java DTOs.
SAP	camel-quarkus-sap	Production Support	None	Provides SAP Camel Component.
Saxon	camel-quarkus-saxon	Production Support	Production Support	Query and/or transform XML payloads using XQuery and Saxon.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Scheduler	camel-quarkus-scheduler	Production Support	Production Support	Generate messages in specified intervals using <code>java.util.concurrent.ScheduledExecutorService</code> .
Seda	camel-quarkus-seda	Production Support	Production Support	Asynchronously call another endpoint from any Camel Context in the same JVM.
Slack	camel-quarkus-slack	Production Support	Production Support	Send and receive messages to/from Slack.
SNMP	camel-quarkus-snmp	Production Support	None	Receive traps and poll SNMP (Simple Network Management Protocol) capable devices.
Splunk	camel-quarkus-splunk	Production Support	Production Support	Publish or search for events in Splunk.
Sql	camel-quarkus-sql	Production Support	Production Support	Perform SQL queries.
Telegram	camel-quarkus-telegram	Production Support	Production Support	Send and receive messages acting as a Telegram Bot Telegram Bot API.
Timer	camel-quarkus-timer	Production Support	Production Support	Generate messages in specified intervals using <code>java.util.Timer</code> .
Validator	camel-quarkus-validator	Production Support	Production Support	Validate the payload using XML Schema and JAXP Validation.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Velocity	camel-quarkus-velocity	Production Support	Production Support	Transform messages using a Velocity template.
Vertx-http	camel-quarkus-vertx-http	Production Support	Production Support	Camel HTTP client support with Vert.x
Vertx-websocket	camel-quarkus-vertx-websocket	Production Support	Production Support	Camel WebSocket support with Vert.x
XML IO DSL	camel-quarkus-xml-io-dsl	Production Support	Production Support	An XML stack for parsing XML route definitions
XSLT	camel-quarkus-xslt	Production Support	Production Support	Transforms XML payload using an XSLT template.
XSLT-Saxon	camel-quarkus-xslt-saxon	Production Support	Production Support	Transform XML payloads using an XSLT template using Saxon.
Yaml-dsl	camel-quarkus-yaml-dsl	Production Support	Production Support	An YAML stack for parsing YAML route definitions
Zipfile	camel-quarkus-zipfile	Production Support	Production Support	Compression and decompress streams using <code>java.util.zip.ZipStream</code> .
Zip-deflater	camel-quarkus-zip-deflater	Production Support	Production Support	Compress and decompress streams using <code>java.util.zip.Deflater</code> , <code>java.util.zip.Inflater</code> or <code>java.util.zip.GZIPStream</code> .

1.3. SUPPORTED LANGUAGES

There are 7 languages.

Table 1.3. Red Hat build of Apache Camel for Quarkus Support Matrix Languages

Extension	Artifact	JVM Support Level	Native Support Level	Description
Bean	camel-quarkus-bean	Production Support	Production Support	Invoke methods of Java beans
Core	camel-quarkus-core	Production Support	Production Support	Camel core functionality and basic Camel languages/ Constant, ExchangeProperty, Header, Ref, Simple and Tokenize
HL7	camel-quarkus-hl7	Production Support	Production Support	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
Jsonpath	camel-quarkus-jsonpath	Production Support	Production Support	Evaluate a JSONPath expression against a JSON message body
Jslt	camel-quarkus-jslt	Production Support	Production Support	Query or transform JSON payloads using an JSLT.
Saxon	camel-quarkus-saxon	Production Support	Production Support	Query and/or transform XML payloads using XQuery and Saxon.
Xpath	camel-quarkus-xpath	Production Support	Production Support	Evaluates an XPath expression against an XML payload

1.4. SUPPORTED DATA FORMATS

There are 10 data formats.

Table 1.4. Red Hat build of Apache Camel for Quarkus Support Matrix Data formats

Extension	Artifact	JVM Support Level	Native Support Level	Description
Avro	camel-quarkus-avro	Production Support	Production Support	Serialize and deserialize messages using Apache Avro binary data format.
Bindy	camel-quarkus-bindy	Production Support	Production Support	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
Crypto (Java Cryptographic Extension)	camel-quarkus-crypto	Production Support	Production Support	Symmetric (shared-key) encryption and decryption using Camel's marshal and unmarshal formatting mechanism.
Gson	camel-quarkus-gson	Production Support	Production Support	Marshal POJOs to JSON and back using Gson
HL7	camel-quarkus-hl7	Production Support	Production Support	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
Jackson	camel-quarkus-jackson	Production Support	Production Support	Marshal POJOs to JSON and back using Jackson
Jackson-avro	camel-quarkus-jackson-avro	Production Support	Production Support	Marshal POJOs to Avro and back using Jackson.

Extension	Artifact	JVM Support Level	Native Support Level	Description
Jackson-protobuf	camel-quarkus-jackson-protobuf	Production Support	Production Support	Marshal POJOs to Protobuf and back using Jackson.
Jacksonxml	camel-quarkus-jacksonxml	Production Support	Production Support	Unmarshal an XML payloads to POJOs and back using XMLMapper extension of Jackson.
Jaxb	camel-quarkus-jaxb	Production Support	Production Support	Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.
Xml-jaxp	camel-quarkus-xml-jaxp	Production Support	Production Support	XML JAXP type converters and parsers
PGP	camel-quarkus-crypto	Production Support	Production Support	Symmetric (shared-key) encryption and decryption using Camel's marshal and unmarshal formatting mechanism.
Soap	camel-quarkus-soap	Production Support	Production Support	Marshal Java objects to SOAP messages and back.
Xml-JAXP	camel-quarkus-xml-jaxp	Production Support	Production Support	XML JAXP type converters and parsers

CHAPTER 2. EXTENSIONS REFERENCE

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

2.1. AMQP

Messaging with AMQP protocol using Apache QPid Client.

2.1.1. What's inside

- [AMQP component](#), URI syntax: **amqp:destinationType:destinationName**

Refer to the above link for usage and configuration details.

2.1.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.1.3. Usage

2.1.3.1. 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.

2.1.3.2. 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.

2.1.3.3. 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
```

2.1.4. 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)
```

2.1.5. 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](#).

2.2. ATTACHMENTS

Support for attachments on Camel messages

2.2.1. What's inside

- [Attachments](#)

Refer to the above link for usage and configuration details.

2.2.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.3. AVRO

Serialize and deserialize messages using Apache Avro binary data format.

2.3.1. What's inside

- [Avro data format](#)

Refer to the above link for usage and configuration details.

2.3.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.3.3. 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](#).

2.4. AWS 2 CLOUDWATCH

Sending metrics to AWS CloudWatch.

2.4.1. What's inside

- [AWS CloudWatch component](#), URI syntax: **aws2-cw:namespace**

Refer to the above link for usage and configuration details.

2.4.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.4.3. 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](#).

2.5. AWS 2 DYNAMODB

Store and retrieve data from AWS DynamoDB service or receive messages from AWS DynamoDB Stream.

2.5.1. 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.

2.5.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.5.3. SSL in native mode

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

2.5.4. Additional Camel Quarkus configuration

2.5.4.1. Optional integration with Quarkus Amazon 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;
}
```

2.6. AWS 2 KINESIS

Consume and produce records from AWS Kinesis Streams.

2.6.1. 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.

2.6.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.6.3. SSL in native mode

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

2.7. AWS 2 LAMBDA

Manage and invoke AWS Lambda functions.

2.7.1. What's inside

- [AWS Lambda component](#), URI syntax: **aws2-lambda:function**

Refer to the above link for usage and configuration details.

2.7.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.7.3. SSL in native mode

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

2.7.4. Additional Camel Quarkus configuration

2.7.4.1. 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.

2.8. AWS 2 S3 STORAGE SERVICE

Store and retrieve objects from AWS S3 Storage Service.

2.8.1. What's inside

- [AWS S3 Storage Service component](#), URI syntax: **aws2-s3://bucketNameOrArn**

Refer to the above link for usage and configuration details.

2.8.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.8.3. 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](#).

2.8.4. Additional Camel Quarkus configuration

2.8.4.1. 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;
}
```

2.9. AWS 2 SIMPLE NOTIFICATION SYSTEM (SNS)

Send messages to an AWS Simple Notification Topic.

2.9.1. What's inside

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

Refer to the above link for usage and configuration details.

2.9.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.9.3. 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](#).

2.9.4. Additional Camel Quarkus configuration

2.9.4.1. 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;
}
```

2.10. AWS 2 SIMPLE QUEUE SERVICE (SQS)

Send and receive messages to/from AWS SQS service.

2.10.1. What's inside

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

Refer to the above link for usage and configuration details.

2.10.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.10.3. 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](#).

2.10.4. Additional Camel Quarkus configuration

2.10.4.1. 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;
}
```

2.11. AZURE SERVICEBUS

Send and receive messages to/from Azure Service Bus.

2.11.1. What's inside

- [Azure ServiceBus component](#), URI syntax: **azure-servicebus:topicOrQueueName**

Refer to the above link for usage and configuration details.

2.11.2. Maven coordinates

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

2.12. AZURE STORAGE BLOB SERVICE

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

2.12.1. 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.

2.12.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.12.3. Usage

2.12.3.1. 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>
```

2.12.4. 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](#).

2.13. 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.

2.13.1. 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.

2.13.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.13.3. Usage

2.13.3.1. 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>
```

2.13.4. 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](#).

2.14. BEAN VALIDATOR

Validate the message body using the Java Bean Validation API.

2.14.1. What's inside

- [Bean Validator component](#), URI syntax: **bean-validator:label**

Refer to the above link for usage and configuration details.

2.14.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.14.3. Usage

2.14.3.1. 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.

2.14.3.2. 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 {
}
```

2.14.4. 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](#)).

2.15. BEAN

Invoke methods of Java beans

2.15.1. 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.

2.15.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.15.3. Usage

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

2.16. 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

2.16.1. 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.

2.16.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.16.3. 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.

2.17. BROWSE

Inspect the messages received on endpoints supporting `BrowsableEndpoint`.

2.17.1. What's inside

- [Browse component](#), URI syntax: **browse:name**

Refer to the above link for usage and configuration details.

2.17.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.18. CASSANDRA CQL

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

2.18.1. What's inside

- [Cassandra CQL component](#), URI syntax: **cql:beanRef:hosts:port/keyspace**

Refer to the above link for usage and configuration details.

2.18.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.18.3. Additional Camel Quarkus configuration

2.18.3.1. 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)**.

2.19. CLI CONNECTOR

Runtime adapter connecting with Camel CLI

2.19.1. What's inside

- [CLI Connector](#)

Refer to the above link for usage and configuration details.

2.19.2. Maven coordinates

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

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

2.20. CONTROL BUS

Manage and monitor Camel routes.

2.20.1. What's inside

- [Control Bus component](#), URI syntax: **controlbus:command:language**

Refer to the above link for usage and configuration details.

2.20.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.20.3. Usage

2.20.3.1. Languages

2.20.3.1.1. 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**.

2.20.3.1.2. 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
```

2.20.4. Camel Quarkus limitations

2.20.4.1. Statistics

2.21. CORE

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

2.21.1. 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.

2.21.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.21.3. Additional Camel Quarkus configuration

2.21.3.1. Simple language

2.21.3.1.1. 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](#).

2.21.3.1.2. 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](#).

2.21.3.1.3. 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
```




2.21.3.1.4. 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
 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	





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

Configuration property	Type	Default
 quarkus.camel.runtime-catalog.dataformats If true the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel data formats available in the application; otherwise data format JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to false helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to false except for making the behavior consistent with native mode.	boolean	true
 quarkus.camel.runtime-catalog-models If true the Runtime Camel Catalog embedded in the application will contain JSON schemas of Camel EIP models available in the application; otherwise EIP model JSON schemas will not be available in the Runtime Camel Catalog and any attempt to access those will result in a RuntimeException. Setting this to false helps to reduce the size of the native image. In JVM mode, there is no real benefit of setting this flag to false except for making the behavior consistent with native mode.	boolean	true
 quarkus.camel.routes-discovery.enabled Enable automatic discovery of routes during static initialization.	boolean	true
 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: <code>**/Bar*</code> To exclude all routes form a specific package use: <code>com/mycompany/bar/*</code> To exclude all routes form a specific package and its sub-packages use double wildcards: <code>com/mycompany/bar/**</code> And to exclude all routes from two specific packages use: <code>com/mycompany/bar/*,com/mycompany/stuff/*</code>	string	

Configuration property	Type	Default
 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: <code>**/Foo*</code> To include all routes form a specific package use: <code>com/mycompany/foo/*</code> To include all routes form a specific package and its sub-packages use double wildcards: <code>com/mycompany/foo/**</code> And to include all routes from two specific packages use: <code>com/mycompany/foo/*,com/mycompany/stuff/*</code>	string	
 quarkus.camel.native.reflection.exclude-patterns A comma separated list of Ant-path style patterns to match class names that should be excluded from registering for reflection. Use the class name format as returned by the java.lang.Class.getName() method: package segments delimited by period <code>.</code> and inner classes by dollar sign <code>\$</code> . This option narrows down the set selected by include-patterns . By default, no classes are excluded. This option cannot be used to unregister classes which have been registered internally by Quarkus extensions.	string	

Configuration property	Type	Default
<p> quarkus.camel.native.reflection.include-patterns</p> <p>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 <code>.</code> and inner classes by dollar sign <code>\$</code>.</p> <p>By default, no classes are included. The set selected by this option can be narrowed down by exclude-patterns.</p> <p>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.</p> <p>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.</p> <p>For this option to work properly, at least one of the following conditions must be satisfied:</p> <ul style="list-style-type: none"> - There are no wildcards (<code>*</code> or <code>/</code>) 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. <pre>` quarkus.index-dependency.my-dep.group-id = org.my-group quarkus.index-dependency.my-dep.artifact-id = my-artifact `</pre> <p>where my-dep is a label of your choice to tell Quarkus that org.my-group and with my-artifact belong together.</p>	string	
<p> quarkus.camel.native.reflection.serialization-enabled</p> <p>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.</p>	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
<div> quarkus.camel.main.arguments.on-unknown</div> <div>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</div>	org.apache.camel.quarkus.core.CamelConfig.FailureRem	warn

Configuration property	e T y p e	Default
{doc-link-icon-lock}[title=Fixed at build time] Configuration property fixed at build time. All other configuration properties are overridable at runtime.		

2.22. CRON

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

2.22.1. What's inside

- [Cron component](#), URI syntax: **cron:name**

Refer to the above link for usage and configuration details.

2.22.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.22.3. 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.

2.23. CRYPTO (JCE)

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

2.23.1. 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.

2.23.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

■

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

2.23.3. 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](#).

2.24. CXF

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

2.24.1. What's inside

- [CXF component](#), URI syntax: **cxf:beanId:address**

Refer to the above link for usage and configuration details.

2.24.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.24.3. Usage

2.24.3.1. 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

To learn about supported use cases and WS specifications, see the [Quarkus CXF Reference](#).

2.24.3.2. 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.

2.24.3.3. 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) ❶
public interface CalculatorService {

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

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

    @WebMethod ❸
    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 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.

2.24.3.4. 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.setWsdIURL("wsdl/HelloService.wsdl");
        return result;
    }
}

```

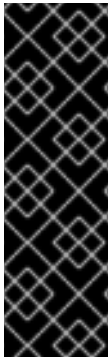
The path under which these two services will be served depends on the value of **quarkus.cxf.path** configuration property which can for example be set in **application.properties**:

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.

2.24.3.5. 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.setWsdIURL("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.setWsdIURL("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.

2.24.3.6. 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.

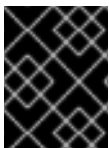
TIP

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.

2.24.3.7. 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
```

2.25. 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](#).

2.25.1. What's inside

- [Data Format component](#), URI syntax: **dataformat:name:operation**

Refer to the above link for usage and configuration details.

2.25.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

```
</dependency>
```

2.26. DATASET

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

2.26.1. 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.

2.26.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.27. DIRECT

Call another endpoint from the same Camel Context synchronously.

2.27.1. What's inside

- [Direct component](#), URI syntax: **direct:name**

Refer to the above link for usage and configuration details.

2.27.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.28. 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.

2.28.1. 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.

2.28.2. Maven coordinates

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

Or add the coordinates to your existing project:



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





2.28.3. 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](#).

2.28.4. 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
 quarkus.camel.fhir.enable-dstu2_hl7org Enable FHIR DSTU2_HL7ORG Specs in native mode.	boolean	false

Configuration property	Type	Default
 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.

2.29. FILE

Read and write files.

2.29.1. What's inside

- [File component](#), URI syntax: **file:directoryName**

Refer to the above link for usage and configuration details.

2.29.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.29.3. Additional Camel Quarkus configuration

2.29.3.1. 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
 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	

Configuration property	Type	Default
 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 <p>The custom attributes associated to the service (defaults to empty map).</p>	Map<String, String>	

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

2.30. FTP

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

2.30.1. 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.

2.30.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.31. GOOGLE BIGQUERY

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

2.31.1. 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.

2.31.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.31.3. 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.

2.32. GOOGLE PUBSUB

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

2.32.1. What's inside

- [Google Pubsub component](#), URI syntax: **google-pubsub:projectId:destinationName**

Refer to the above link for usage and configuration details.

2.32.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.32.3. 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.

2.33. GRPC

Expose gRPC endpoints and access external gRPC endpoints.

2.33.1. What's inside

- [gRPC component](#), URI syntax: **grpc:host:port/service**

Refer to the above link for usage and configuration details.

2.33.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.33.3. Usage

2.33.3.1. 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>
```

```

    </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.

2.33.3.1.1. 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**.

2.33.3.1.2. 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**.

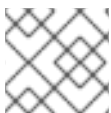
First add a dependency for the artifact(s) containing proto files to your project. Next, enable proto file dependency scanning.

```
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 ****.

2.33.3.2. 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
```

2.33.4. Camel Quarkus limitations


2.33.4.1. 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.

2.33.5. 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: - <i>none</i> – default – don't scan dependencies – a comma separated list of <i>groupId:artifactId</i> coordinates to scan – <i>all</i> – scan all dependencies	string	none

Configuration property	Type	Default
 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: - <i>none</i> - default - don't scan dependencies - a comma separated list of <i>groupId:artifactId</i> coordinates to scan - <i>all</i> - scan all dependencies The default is <i>com.google.protobuf:protobuf-java</i> .	string	com.google.protobuf:protobuf-java
 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.	Mapping<String, List<String>>	

Configuration property	Type	Default
 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.	Mapping List<String>	

{doc-link-icon-lock}[title=Fixed at build time] Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.34. GSON

Marshal POJOs to JSON and back using Gson

2.34.1. What's inside

- [JSON Gson data format](#)

Refer to the above link for usage and configuration details.

2.34.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.34.3. Additional Camel Quarkus configuration

2.34.3.1. 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](#).

2.35. HL7

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

2.35.1. What's inside

- [HL7 data format](#)
- [HL7 Terser language](#)

Refer to the above links for usage and configuration details.

2.35.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.35.3. Camel Quarkus limitations

For MLLP with TCP, Netty is the only supported means of running an HL7 MLLP listener. Mina is not supported since it has no GraalVM native support at present.

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

2.36. HTTP

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

2.36.1. 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.

2.36.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.36.3. SSL in native mode

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

2.36.4. 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.

2.37. INFINISPAN

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

2.37.1. What's inside

- [Infinispan component](#), URI syntax: **infinispan:cacheName**

Refer to the above link for usage and configuration details.

2.37.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.37.3. Additional Camel Quarkus configuration

2.37.3.1. 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...  
    }  
}
```

2.37.3.2. Camel Infinispan InfinispanRemoteAggregationRepository in native mode

If you chose to use the **InfinispanRemoteAggregationRepository** in native mode, then you must [enable native serialization support](#).

2.38. AVRO JACKSON

Marshal POJOs to Avro and back using Jackson.

2.38.1. What's inside

- [Avro Jackson data format](#)

Refer to the above link for usage and configuration details.

2.38.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.39. PROTOBUF JACKSON

Marshal POJOs to Protobuf and back using Jackson.

2.39.1. What's inside

- [Protobuf Jackson data format](#)

Refer to the above link for usage and configuration details.

2.39.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.40. JACKSON

Marshal POJOs to JSON and back using Jackson

2.40.1. What's inside

- [JSON Jackson data format](#)

Refer to the above link for usage and configuration details.

2.40.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.40.3. Usage

2.40.3.1. Configuring the Jackson **ObjectMapper**

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

2.40.3.1.1. **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.

2.40.3.1.2. 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)
    }
}

```

2.40.3.1.3. Using the Quarkus JacksonObjectMapper 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)
    }
}
```

2.41. JACKSONXML

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

2.41.1. What's inside

- [Jackson XML data format](#)

Refer to the above link for usage and configuration details.

2.41.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.42. JAVA JOOR DSL

Support for parsing Java route definitions at runtime

2.42.1. What's inside

- [Java DSL \(runtime compiled\)](#)

Refer to the above link for usage and configuration details.

2.42.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.42.3. 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.

2.43. JAXB

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

2.43.1. What's inside

- [JAXB data format](#)

Refer to the above link for usage and configuration details.

2.43.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.43.3. Usage

2.43.3.1. 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.

2.44. JDBC

Access databases through SQL and JDBC.

2.44.1. What's inside

- [JDBC component](#), URI syntax: **jdbc:dataSourceName**

Refer to the above link for usage and configuration details.

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

2.44.3. Additional Camel Quarkus configuration

2.44.3.1. 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")**.

2.44.3.1.1. 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")**.

2.45. JIRA

Interact with JIRA issue tracker.

2.45.1. What's inside

- [Jira component](#), URI syntax: **jira:type**

Refer to the above link for usage and configuration details.

2.45.2. 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-jira</artifactId>
</dependency>
```

2.45.3. SSL in native mode

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

2.46. JMS

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

2.46.1. What's inside

- [JMS component](#), URI syntax: **jms:destinationType:destinationName**

Refer to the above link for usage and configuration details.

2.46.2. 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-jms</artifactId>
</dependency>
```

2.46.3. Usage

2.46.3.1. 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.

2.46.3.2. 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.

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

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-qpuid-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);
}

```

2.47. JPA

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

2.47.1. What's inside

- [JPA component](#), URI syntax: **jpa:entityType**

Refer to the above link for usage and configuration details.

2.47.2. 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-jpa</artifactId>
</dependency>

```

2.47.3. 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**.

2.47.3.1. 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.

2.48. JSLT

Query or transform JSON payloads using an JSLT.

2.48.1. What's inside

- [JSLT component](#), URI syntax: **jslt:resourceUri**

Refer to the above link for usage and configuration details.

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

2.48.3. 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.

2.48.4. Additional Camel Quarkus configuration

2.48.4.1. 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 (and 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
```

2.48.4.2. 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;
}
```

2.49. JSON PATH

Evaluate a JSONPath expression against a JSON message body

2.49.1. What's inside

- [JSONPath language](#)

Refer to the above link for usage and configuration details.

2.49.2. 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-jsonpath</artifactId>
</dependency>
```

2.50. JTA

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

2.50.1. What's inside

- [JTA](#)

Refer to the above link for usage and configuration details.

2.50.2. 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-jta</artifactId>
</dependency>
```

2.50.3. 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.

Policy	Description
PROPAGATION_REQUIRES_NEW	Create a new transaction, suspending the current transaction if one exists.
PROPAGATION_SUPPORTS	Support a current transaction; execute non-transactionally if none exists.

2.51. KAFKA

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

2.51.1. What's inside

- [Kafka component](#), URI syntax: **kafka:topic**

Refer to the above link for usage and configuration details.

2.51.2. 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-kafka</artifactId>
</dependency>
```

2.51.3. Usage

2.51.3.1. 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**.

2.51.4. 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.

2.52. KAMELET

Materialize route templates

2.52.1. What's inside

- [Kamelet component](#), URI syntax: **kamelet:templateId/routeId**

Refer to the above link for usage and configuration details.

2.52.2. 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-kamelet</artifactId>
</dependency>
```

2.52.3. Usage

2.52.3.1. 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.


2.52.3.2. 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.

2.52.4. Additional Camel Quarkus configuration

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

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

2.53. KUBERNETES

Perform operations against Kubernetes API

2.53.1. 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>
```

2.53.2. Additional Camel Quarkus configuration

2.53.2.1. 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](#).

2.53.2.2. 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	



Configuration property	Type	Default
 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	
 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	


Configuration property	Type	Default
 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	
 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	

Configuration property	Type	Default
 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	
 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.c.a	

Configuration property	Component.kubernetes.clusters.LeaseResourceType	Default

Configuration property	Type	Default

Configuration property	Type	Default
 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.

2.54. LANGUAGE

Execute scripts in any of the languages supported by Camel.

2.54.1. What's inside

- [Language component](#), URI syntax: **language:languageName:resourceUri**

Refer to the above link for usage and configuration details.

2.54.2. 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-language</artifactId>
```

```
</dependency>
```

2.54.3. Usage

2.54.3.1. 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.

2.54.3.2. 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
```

2.54.4. 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.

2.55. LDAP

Perform searches on LDAP servers.

2.55.1. What's inside

- [LDAP component](#), URI syntax: **ldap:dirContextName**

Refer to the above link for usage and configuration details.

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

2.55.3. Usage

2.55.3.1. 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.
}
```

2.56. LOG

Log messages to the underlying logging mechanism.

2.56.1. What's inside

- [Log component](#), URI syntax: **log:loggerName**

Refer to the above link for usage and configuration details.

2.56.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.57. MAIL

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

2.57.1. 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.

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

2.58. MANAGEMENT

JMX management strategy and associated managed resources.

2.58.1. 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>
```

2.58.2. Usage

For information on using Managed Beans in Camel, consult the [JMX section of the Camel Manual](#).

2.58.2.1. 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.

2.58.2.2. 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
```

For more information, refer to the [Quarkus native guide](#).

2.59. MAPSTRUCT

Type Conversion using Mapstruct

2.59.1. What's inside

- [MapStruct component](#), URI syntax: **mapstruct:className**

Refer to the above link for usage and configuration details.

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

2.59.3. Usage

2.59.3.1. Annotation Processor

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

2.59.3.1.1. 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>
```

2.59.3.1.2. Gradle

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

2.59.3.2. 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
```

2.60. MASTER

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

2.60.1. What's inside

- [Master component](#), URI syntax: **master:namespace:delegateUri**

Refer to the above link for usage and configuration details.

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

2.60.3. Additional Camel Quarkus configuration

This extension can be used in conjunction with extensions below:

- [Camel Quarkus File](#)
- [Camel Quarkus Kubernetes](#)

2.61. MICROMETER

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

2.61.1. What's inside

- [Micrometer component](#), URI syntax: **micrometer:metricsType:metricsName**

Refer to the above link for usage and configuration details.

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

2.61.3. 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.

2.61.4. Camel Quarkus limitations

2.61.4.1. 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.






2.61.4.2. Decrement header for Counter is ignored by Prometheus


Prometheus backend ignores negative values during increment of Counter metrics.

2.61.4.3. 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.

2.61.5. 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
 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.

2.62. MICROPROFILE FAULT TOLERANCE

Circuit Breaker EIP using Microprofile Fault Tolerance

2.62.1. What's inside

- [Microprofile Fault Tolerance](#)

Refer to the above link for usage and configuration details.

2.62.2. 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-microprofile-fault-tolerance</artifactId>
</dependency>
```

2.63. MICROPROFILE HEALTH

Expose Camel health checks via MicroProfile Health

2.63.1. What's inside

- [Microprofile Health](#)

Refer to the above link for usage and configuration details.

2.63.2. Maven coordinates

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

Or add the coordinates to your existing project:

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

2.63.3. Usage

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

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

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

Refer to the [Quarkus health guide](#) for further information.

2.63.3.1. Provided health checks

Some checks are automatically registered for your application.


2.63.3.1.1. Camel Context Health


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

2.63.3.1.2. Camel Route Health

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

2.63.4. Additional Camel Quarkus configuration

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

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

2.64. MINIO

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

2.64.1. What's inside

- [Minio component](#), URI syntax: **minio:bucketName**

Refer to the above link for usage and configuration details.

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

2.64.3. 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
```

2.65. MLLP

Communicate with external systems using the MLLP protocol.

2.65.1. What's inside

- [MLLP component](#), URI syntax: **mlp:hostname:port**

Refer to the above link for usage and configuration details.

2.65.2. 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-mlp</artifactId>
</dependency>
```

2.65.3. Additional Camel Quarkus configuration

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

2.66. MOCK

Test routes and mediation rules using mocks.

2.66.1. What's inside

- [Mock component](#), URI syntax: **mock:name**

Refer to the above link for usage and configuration details.

2.66.2. 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-mock</artifactId>
</dependency>
```

2.66.3. 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");
    }
}

```

2.66.4. Camel Quarkus limitations

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

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

2.67. MONGODB

Perform operations on MongoDB documents and collections.

2.67.1. What's inside

- [MongoDB component](#), URI syntax: **mongodb:connectionBean**

Refer to the above link for usage and configuration details.

2.67.2. 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-mongodb</artifactId>
</dependency>

```

2.67.3. Additional Camel Quarkus configuration

The extension leverages the [Quarkus MongoDB Client](#) extension. The Mongo client can be configured via the Quarkus MongoDB Client [configuration options](#).

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

```

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

```

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

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

```
//Routes.java

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

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

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

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

2.68. MYBATIS

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

2.68.1. 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.

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

2.68.3. 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.

2.69. NETTY HTTP

The Netty HTTP extension provides HTTP transport on top of the [Netty](#) extension.

2.69.1. What's inside

- [Netty HTTP component](#), URI syntax: **netty-http:protocol://host:port/path**

Refer to the above link for usage and configuration details.

2.69.2. 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-netty-http</artifactId>
</dependency>
```

2.69.3. 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)
```

2.69.4. 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.

2.70. NETTY

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

2.70.1. What's inside

- [Netty component](#), URI syntax: **netty:protocol://host:port**

Refer to the above link for usage and configuration details.

2.70.2. 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-netty</artifactId>
</dependency>
```

2.71. OPENAPI JAVA

Expose OpenAPI resources defined in Camel REST DSL

2.71.1. What's inside

- [Openapi Java](#)

Refer to the above link for usage and configuration details.

2.71.2. 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-openapi-java</artifactId>
</dependency>
```

2.71.3. 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.

2.72. OPENTELEMETRY

Distributed tracing using OpenTelemetry

2.72.1. What's inside

- [OpenTelemetry](#)

Refer to the above link for usage and configuration details.

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

2.72.3. 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:*
```

2.72.3.1. Exporters

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

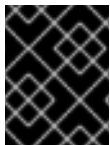
2.72.3.2. Tracing CDI bean method execution

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](#).

2.72.4. 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
 quarkus.camel.opentelemetry.exclude-patterns Sets whether to disable tracing for endpoint URIs that match the given patterns. The pattern can take the following forms: <ol style="list-style-type: none"> 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	

{doc-link-icon-lock}[title=Fixed at build time] Configuration property fixed at build time. All other configuration properties are overridable at runtime.

2.73. PAHO MQTT5

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

2.73.1. What's inside

- [Paho MQTT 5 component](#), URI syntax: **paho-mqtt5:topic**

Refer to the above link for usage and configuration details.

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

2.74. PAHO

Communicate with MQTT message brokers using Eclipse Paho MQTT Client.

2.74.1. What's inside

- [Paho component](#), URI syntax: **paho:topic**

Refer to the above link for usage and configuration details.

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

2.75. 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.

2.75.1. What's inside

- [Platform HTTP component](#), URI syntax: **platform-http:path**

Refer to the above link for usage and configuration details.

2.75.2. 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-platform-http</artifactId>
</dependency>
```

2.75.3. Usage

2.75.3.1. Basic Usage

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

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

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


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

2.75.3.2. 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");
```

2.75.3.3. 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);
        }
      });
    }
  });
```

2.75.3.4. 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(VertexPlatformHttpConstants.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**.

2.75.3.5. 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 + "}");
```

2.75.4. Additional Camel Quarkus configuration

2.75.4.1. Platform HTTP server configuration

Configuration of the platform HTTP server is managed by Quarkus. Refer to the [Quarkus HTTP configuration guide](#) for the full list of configuration options.

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

2.75.4.2. 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.

2.76. QUARTZ

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

2.76.1. What's inside

- [Quartz component](#), URI syntax: **quartz:groupName/triggerName**

Refer to the above link for usage and configuration details.

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

2.76.3. Usage

2.77. REF

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

2.77.1. What's inside

- [Ref component](#), URI syntax: **ref:name**

Refer to the above link for usage and configuration details.

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

2.77.3. 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");
    }
}

```

2.78. REST OPENAPI

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

2.78.1. What's inside

- [REST OpenApi component](#), URI syntax: **rest-openapi:specificationUri#operationId**

Refer to the above link for usage and configuration details.

2.78.2. 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-rest-openapi</artifactId>
</dependency>

```

2.78.3. Usage

2.78.3.1. Required Dependencies

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

- camel-quarkus-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
```

2.79. REST

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

2.79.1. 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.

2.79.2. 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-rest</artifactId>
</dependency>
```

2.79.3. Additional Camel Quarkus configuration

This extension depends on the [Platform HTTP](#) extension and configures it as the component that provides the REST transport.

2.79.3.1. 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](#).

2.79.3.2. Configuring alternate REST transport providers

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

```

import org.apache.camel.builder.RouteBuilder;

public class CamelRoute extends RouteBuilder {

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

```

2.80. SALESFORCE

Communicate with Salesforce using Java DTOs.

2.80.1. What's inside

- [Salesforce component](#), URI syntax: **salesforce:operationName:topicName**

Refer to the above link for usage and configuration details.

2.80.2. 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-salesforce</artifactId>
</dependency>
```

2.80.3. Usage

2.80.3.1. Generating Salesforce DTOs with the `salesforce-maven-plugin`

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>
```

2.80.4. SSL in native mode

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

2.81. SAP

Provides SAP Camel Component

2.81.1. 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.

2.81.2. Maven coordinates

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

2.81.2.1. Additional platform restrictions for the SAP component

Because the SAP component depends on the third-party JCo 3 and IDoc 3 libraries, it can only be installed on the platforms that these libraries support.

2.81.2.2. SAP JCo and SAP IDoc libraries

A prerequisite for using the SAP component is that the SAP Java Connector (SAP JCo) libraries and the SAP IDoc library are installed into the **lib/** directory of the Java runtime. You must make sure that you download the appropriate set of SAP libraries for your target operating system from the SAP Service Marketplace.

The names of the library files vary depending on the target operating system, as shown below.

Table 2.1. 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

2.81.3. 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[:applicationRelease]]]
sap-qidoclist-
destination:destinationName:queueName:idocType[:idocTypeExtension[:systemRelease[:applicationRelease]]]
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.

2.81.3.1. 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

Name	Default	Description
transacted	false	If true , specifies that this endpoint initiates an SAP transaction

2.81.3.2. 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.

2.81.3.3. Options for the IDoc List Server endpoint

The SAP IDoc List Server endpoint (**sap-idoclist-server**) supports the following URI options:

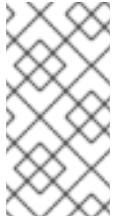
Name	Default	Description
stateful	false	If true , specifies that this endpoint initiates an SAP stateful session.
propagateExceptions	false	If true , specifies that this endpoint propagates exceptions back to the caller in SAP, instead of the exchange's exception handler.

2.81.3.4. Summary of the RFC and IDoc endpoints

The SAP component package provides the following RFC and IDoc endpoints:

sap-srfc-destination

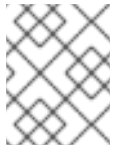
Camel SAP Synchronous Remote Function Call Destination Camel component. This endpoint should be used in cases where Camel routes require synchronous delivery of requests to and responses from an SAP system.

**NOTE**

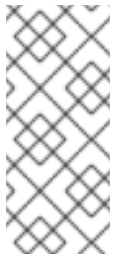
The sRFC protocol used by this component delivers requests and responses to and from an SAP system, using **best effort**. In case of a communication error while sending a request, the completion status of a remote function call in the receiving SAP system remains **in doubt**.

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 **tids** 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.

2.81.3.5. 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.

2.81.3.6. 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.

2.81.3.7. 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.

2.81.3.8. 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*.

2.81.3.9. 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.

2.81.4. 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.

2.81.4.1. 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)

2.81.4.2. 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 URL:

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

2.81.4.2.1. 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.

2.81.4.2.2. 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.
mysapssso2		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).

2.81.4.2.3. 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 their 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 <code>sapmsXXX</code> 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 gwshost. 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 .

2.81.4.2.4. 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.

2.81.4.2.5. 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.

2.81.4.2.6. 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		<p>Enable the RFC_METADATA_GET API, which provides the repository data in one single round trip.</p> <p>1 Activates use of RFC_METADATA_GET in ABAP System.</p> <p>0 Deactivates RFC_METADATA_GET in ABAP System.</p> <p>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.</p> <p>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.</p>
--	--	---

2.81.4.2.7. 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].

2.81.4.3. 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
```

2.81.4.3.1. 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.

2.81.4.3.2. 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.

2.81.4.3.3. 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 .
-----------------------------	--	--

2.81.4.4. 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.fusesource.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);
}
```

2.81.4.4.1. 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(...);
```

2.81.4.4.2. 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);
```

2.81.4.4.3. 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(...);
```

2.81.4.4.4. 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);
```

2.81.5. Message Headers

The SAP component supports the following message headers:

Header	Description
CamelSap.scheme	<p>The URI scheme of the last endpoint to process the message. Use one of the following values: sap-srfc-destination</p> <p>sap-trfc-destination</p> <p>sap-qrfc-destination</p> <p>sap-srfc-server</p> <p>sap-trfc-server</p> <p>sap-idoc-destination</p> <p>sap-idoclist-destination</p> <p>sap-qidoc-destination</p> <p>sap-qidoclist-destination</p> <p>sap-idoclist-server</p>

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.

2.81.6. 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.

2.81.7. Message Body for RFC

2.81.7.1. 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;
    ...
}
```

2.81.7.2. 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. Please refer to the EMF documentation for further details.



NOTE

Attempts to get a parameter not defined on a structure object will return null. Attempts to set a parameter not defined on a structure will throw an exception as well as attempts to set the value of a parameter with an incorrect type.

As discussed in the following sections, structure objects can contain fields that contain values of the complex field types, **STRUCTURE**, and **TABLE**.



NOTE

It is unnecessary to create instances of these types and add them to the structure. Instances of these field values are created on demand if necessary when accessed in the enclosing structure.

2.81.7.3. 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.

2.81.7.3.1. 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.math.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

2.81.7.3.2. 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.

2.81.7.3.3. 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**.

2.81.7.3.4. 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.

2.81.7.3.5. 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.

2.81.7.3.6. 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

2.81.7.3.7. 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**.

2.81.7.3.8. 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**.

2.81.7.3.9. 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.

2.81.8. Message Body for IDoc

2.81.8.1. 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
```

2.81.8.2. 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 getApplicationRelease();

int getNumFields();

long getMaxOccurrence();

long getMinOccurrence();

boolean isMandatory();

boolean isQualified();

int getRecordLength();

<T> T get(Object key, Class<T> type);
}

```

The **getChildren(String segmentType)** method is particularly useful for adding new (nested) children to a segment. It returns an object of type, **SegmentList**, which is defined as follows:

```

// Java
package org.fusesource.camel.component.sap.model.idoc;
...
public interface SegmentList<S extends Segment> extends EObject, EList<S> {
    S add();

    S add(int index);
}

```

Hence, to create a data record of **E1SCU_CRE** type, you could use Java code like the following:

```
Segment rootSegment = document.getRootSegment();

Segment E1SCU_CRE_Segment = rootSegment.getChildren("E1SCU_CRE").add();
```

2.81.8.3. 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 2.1. 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");

```

2.81.9. Document attributes

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

Table 2.2. 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

2.81.9.1. 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");
```

2.81.9.2. 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>
```

2.81.10. Transaction Support

2.81.10.1. 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.

2.81.10.2. 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.

2.81.10.3. 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.

2.81.10.4. 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.

2.81.10.5. 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.

2.81.11. 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.

2.81.11.1. 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.

2.81.11.2. 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>
```

2.81.11.3. 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.

2.81.11.4. 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:complexType name="CONNECTION_INFO_STRUCTURE_TABLE">
    <xs:sequence>
      <xs:element
        name="row"
        minOccurs="0"
        maxOccurs="unbounded"
        type="CONNECTION_INFO_STRUCTURE"/>
    </xs:sequence>
  </xs:complexType>
```



```

    ...
    <xs:sequence>
  </xs:sequence>
</xs:complexType>

  <xs:complexType name="CONNECTION_INFO_STRUCTURE">
    ...
  </xs:complexType>
  ...
</xs:schema>

```

This distinction will be important when specifying a JAXB bean to marshal and unmarshal the structure.

2.81.11.5. 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"/>

```

2.81.11.6. 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"
```

2.81.12. 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.

2.81.12.1. 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/applicati
onRelease
```

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///
```

2.81.12.2. 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">
```

2.81.12.3. 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 2.2. 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>

```

2.81.13. 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.

2.81.13.1. 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");

```

2.81.13.2. 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>

```

2.81.13.3. 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);
}
```

2.81.13.4. 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);
}

```

2.81.14. 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.

2.81.14.1. 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");

```

2.81.14.2. 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>

```

2.81.14.3. 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.

2.81.14.4. 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.

2.81.15. 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.

2.81.15.1. Java DSL for route

The Java DSL for the example route is as follows:

```

DateFormat jaxb = new JaxbDateFormat("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);

```

2.81.15.2. 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>

```

2.81.15.3. 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;
}

```

```
...
}
```

2.81.15.4. BookFlightResponse bean

The following listing illustrates a JAXB bean which marshals to the serialized form of an SAP **BOOK_FLIGHT** response object:

```
@XmlElement(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;

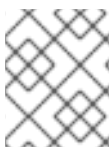
    @XmlAttribute(name="PASSNAME")
    private String passengerName;

    @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.

2.81.15.5. 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;

    ...
}

```

2.81.15.6. 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;

    ...
}

```

2.81.15.7. 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;

    ...
}

```

2.82. XQUERY

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

2.82.1. What's inside

- [XQuery component](#), URI syntax: **xquery:resourceUri**
- [XQuery language](#)

Refer to the above links for usage and configuration details.

2.82.2. 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-saxon</artifactId>
</dependency>
```

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

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

2.83. SCHEDULER

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

2.83.1. What's inside

- [Scheduler component](#), URI syntax: **scheduler:name**

Refer to the above link for usage and configuration details.

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

2.84. SEDA

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

2.84.1. What's inside

- [SEDA component](#), URI syntax: **seda:name**

Refer to the above link for usage and configuration details.

2.84.2. 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-seda</artifactId>
</dependency>
```

2.85. SLACK

Send and receive messages to/from Slack.

2.85.1. What's inside

- [Slack component](#), URI syntax: **slack:channel**

Refer to the above link for usage and configuration details.

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

2.85.3. SSL in native mode

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

2.86. SNMP

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

2.86.1. What's inside

- [SNMP component](#), URI syntax: **snmp:host:port**

Refer to the above link for usage and configuration details.

2.86.2. Maven coordinates

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

2.87. SOAP DATAFORMAT

Marshal Java objects to SOAP messages and back.

2.87.1. What's inside

- [SOAP data format](#)

Refer to the above link for usage and configuration details.

2.87.2. Maven coordinates

[Create a new project with this extension on code.quarkus.redhat.com](#)

Or add the coordinates to your existing project:

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

2.88. SPLUNK

Publish or search for events in Splunk.

2.88.1. What's inside

- [Splunk component](#), URI syntax: **splunk:name**

Refer to the above link for usage and configuration details.

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

2.88.3. SSL in native mode

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

2.89. SQL

Perform SQL queries.

2.89.1. 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.

2.89.2. 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-sql</artifactId>
</dependency>
```

2.89.3. Additional Camel Quarkus configuration

2.89.3.1. Configuring a DataSource

This extension leverages [Quarkus Agroal](#) for **DataSource** support. Setting up a **DataSource** can be achieved via configuration properties.

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

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

2.89.3.1.1. Zero configuration with Quarkus Dev Services

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

2.89.3.2. SQL scripts

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

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

2.89.3.3. 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)**.

2.90. TELEGRAM

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

2.90.1. What's inside

- [Telegram component](#), URI syntax: **telegram:type**

Refer to the above link for usage and configuration details.

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

2.90.3. Usage

2.90.4. 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>
```

2.90.5. 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](#).

2.91. TIMER

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

2.91.1. What's inside

- [Timer component](#), URI syntax: **timer:timerName**

Refer to the above link for usage and configuration details.

2.91.2. 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-timer</artifactId>
</dependency>
```

2.92. VALIDATOR

Validate the payload using XML Schema and JAXP Validation.

2.92.1. What's inside

- [Validator component](#), URI syntax: **validator:resourceUri**

Refer to the above link for usage and configuration details.

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

2.93. VELOCITY

Transform messages using a Velocity template.

2.93.1. What's inside

- [Velocity component](#), URI syntax: **velocity:resourceUri**

Refer to the above link for usage and configuration details.

2.93.2. 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>
```

2.93.3. Usage

2.93.3.1. 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 {
}
```

2.93.4. 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.

2.93.5. 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 (and 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
```

2.94. VERT.X HTTP CLIENT

Camel HTTP client support with Vert.x

2.94.1. What's inside

- [Vert.x HTTP Client component](#), URI syntax: **vertx-http:httpUri**

Refer to the above link for usage and configuration details.

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

2.94.3. 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)
```

2.94.4. Additional Camel Quarkus configuration

2.94.5. 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.

2.94.5.1. 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.

2.95. 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.

2.95.1. What's inside

- [Vert.x WebSocket component](#), URI syntax: **vertx-websocket:host:port/path**

Refer to the above link for usage and configuration details.

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

2.95.3. Usage

2.95.3.1. 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.

2.95.3.2. 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).

2.95.4. Additional Camel Quarkus configuration

2.95.4.1. 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.

2.95.4.2. 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.

2.96. XML IO DSL

An XML stack for parsing XML route definitions

2.96.1. What's inside

- [XML DSL](#)

Refer to the above link for usage and configuration details.

2.96.2. 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>
```

2.96.3. Additional Camel Quarkus configuration

2.96.3.1. 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.

2.97. XML JAXP

XML JAXP type converters and parsers

2.97.1. 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>
```

2.98. XPATH

Evaluates an XPath expression against an XML payload

2.98.1. What's inside

- [XPath language](#)

Refer to the above link for usage and configuration details.

2.98.2. 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-xpath</artifactId>
</dependency>
```

2.98.3. 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 (and 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
```

2.99. XSLT SAXON

Transform XML payloads using an XSLT template using Saxon.

2.99.1. What's inside

- [XSLT Saxon component](#), URI syntax: **xslt-saxon:resourceUri**

Refer to the above link for usage and configuration details.

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

2.100. XSLT

Transforms XML payload using an XSLT template.

2.100.1. What's inside

- [XSLT component](#), URI syntax: **xslt:resourceUri**

Refer to the above link for usage and configuration details.

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

2.100.3. 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.xml, classpath:path/to/my/file.xml
```

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&antlInclude=*.xml")
  .routeId("aggregate").noAutoStartup()
  .aggregate(new XsltSaxonAggregationStrategy("xslt/aggregate.xml"))
  .constant(true)
  .completionFromBatchConsumer()
  .log("after aggregate body: ${body}")
  .to("mock:transformed");
```

Also, it's only supported on JVM mode.

2.100.3.1. Configuration

TransformerFactory features can be configured using following property:

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

2.100.3.2. 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

Configuration property	Type	Default
 quarkus.camel.xslt.features TransformerFactory features.	Mapping, Boolean	

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

2.101. YAML DSL

An YAML stack for parsing YAML route definitions

2.101.1. What's inside

- [YAML DSL](#)

Refer to the above link for usage and configuration details.

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

2.101.3. Usage

2.101.3.1. 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.

2.101.3.1.1. 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 {
}
```

2.101.3.1.2. 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 {
}

```

2.102. ZIP DEFLATE COMPRESSION

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

2.102.1. What's inside

- [GZip Deflater data format](#)
- [Zip Deflater data format](#)

Refer to the above links for usage and configuration details.

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

2.103. ZIP FILE

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

2.103.1. What's inside

- [Zip File data format](#)

Refer to the above link for usage and configuration details.

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