Red Hat build of Apache Camel 4.0

Getting Started with Red Hat build of Apache Camel for Spring Boot

Last Updated: 2023-12-08

Legal Notice

Abstract

This guide introduces Red Hat build of Apache Camel and explains the various ways to create and deploy an application using Red Hat build of Apache Camel.

Table of Contents

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. GETTING STARTED WITH RED HAT BUILD OF APACHE CAMEL FOR SPRING BOOT

This guide introduces Red Hat build of Apache Camel for Spring Boot and demonstrates how to get started building an application using Red Hat build of Apache Camel for Spring Boot:

1.1. RED HAT BUILD OF APACHE CAMEL FOR SPRING BOOT STARTERS

Camel support for Spring Boot provides auto-configuration of the Camel and starters for many Camel components. The opinionated auto-configuration of the Camel context auto-detects Camel routes available in the Spring context and registers the key Camel utilities (such as producer template, consumer template and the type converter) as beans.

NOTE

For information about using a Maven archtype to generate a Camel for Spring Boot application see Generating a Camel for Spring Boot application using Maven .

To get started, you must add the Camel Spring Boot BOM to your Maven **pom.xml** file.

```
<dependencyManagement>

  <dependencies>
    <!-- Camel BOM -->
    <dependency>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>camel-spring-boot-bom</artifactId>
      <version>4.0.0.redhat-00036</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <!-- ... other BOMs or dependencies ... -->
  </dependencies>

</dependencyManagement>
```

The **camel-spring-boot-bom** is a basic BOM that contains the list of Camel Spring Boot starter JARs.

Next, add the Camel Spring Boot starter to startup the Camel Context.

```
<dependencies>
  <!-- Camel Starter -->
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-spring-boot-starter</artifactId>
  </dependency>
  <!-- ... other dependencies ... -->
</dependencies>
```

You must also add the component starters that your Spring Boot application requires. The following example shows how to add the auto-configuration starter to the MQTT5 component.

```
<dependencies>
  <!-- ... other dependencies ... -->
  <dependency>
    <groupId>org.apache.camel.springboot</groupId>
    <artifactId>camel-paho-mqtt5</artifactId>
  </dependency>
</dependencies>
```

1.1.1. Camel Spring Boot BOM vs Camel Spring Boot Dependencies BOM

The curated **camel-spring-boot-dependencies** BOM, which is generated, contains the adjusted JARs that both Spring Boot and Apache Camel use to avoid any conflicts. This BOM is used to test camel-spring-boot itself.

Spring Boot users may choose to use *pure* Camel dependencies by using the **camel-spring-boot-bom** that only has the Camel starter JARs as managed dependencies. However, this may lead to a classpath conflict if a third-party JAR from Spring Boot is not compatible with a particular Camel component.

1.1.2. Spring Boot configuration support

Each starter lists configuration parameters you can configure in the standard **application.properties** or **application.yml** files. These parameters have the form of **camel.component.[component-name].[parameter]**. For example to configure the URL of the MQTT5 broker you can set:

```
camel.component.paho-mqtt5.broker-url=tcp://localhost:61616
```

1.1.3. Adding Camel routes

Camel routes are detected in the Spring application context, for example a route annotated with **org.springframework.stereotype.Component** will be loaded, added to the Camel context and run.

```
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

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

```
        from("...")
            .to("...");
    }

}
```

1.2. SPRING BOOT

Spring Boot automatically configures Camel for you. The opinionated auto-configuration of the Camel context auto-detects Camel routes available in the Spring context and registers the key Camel utilities (like producer template, consumer template and the type converter) as beans.

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

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot</artifactId>
  <version>4.0.0.redhat-00036</version> <!-- use the same version as your Camel core version -->
</dependency>
```

camel-spring-boot jar comes with the **spring.factories** file, so as soon as you add that dependency into your classpath, Spring Boot will automatically auto-configure Camel for you.

1.2.1. Camel Spring Boot Starter

Apache Camel ships a Spring Boot Starter module that allows you to develop Spring Boot applications using starters. There is a sample application in the source code also.

To use the starter, add the following to your spring boot pom.xml file:

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot-bom</artifactId>
  <version>4.0.0.redhat-00036</version> <!-- use the same version as your Camel core version -->
</dependency>
```

Then you can just add classes with your Camel routes such as:

```
package com.example;

import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyRoute extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("timer:foo").to("log:bar");
  }
}
```

Then these routes will be started automatically.

You can customize the Camel application in the **application.properties** or **application.yml** file.

1.2.2. Spring Boot Auto-configuration

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

```
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot-starter</artifactId>
  <version>4.0.0.redhat-00036</version> <!-- use the same version as your Camel core version -->
</dependency>
```

1.2.3. Auto-configured Camel context

The most important piece of functionality provided by the Camel auto-configuration is the **CamelContext** instance. Camel auto-configuration creates a **SpringCamelContext** for you and takes care of the proper initialization and shutdown of that context. The created Camel context is also registered in the Spring application context (under the **camelContext** bean name), so you can access it like any other Spring bean.

```
@Configuration
public class MyAppConfig {

  @Autowired
  CamelContext camelContext;

  @Bean
  MyService myService() {
    return new DefaultMyService(camelContext);
  }

}
```

1.2.4. Auto-detecting Camel routes

Camel auto-configuration collects all the **RouteBuilder** instances from the Spring context and automatically injects them into the provided **CamelContext**. This means that creating new Camel routes with the Spring Boot starter is as simple as adding the **@Component** annotated class to your classpath:

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("jms:invoices").to("file:/invoices");
  }

}
```

Or creating a new route **RouteBuilder** bean in your **@Configuration** class:

```
@Configuration
```

```
public class MyRouterConfiguration {

  @Bean
  RoutesBuilder myRouter() {
    return new RouteBuilder() {

      @Override
      public void configure() throws Exception {
        from("jms:invoices").to("file:/invoices");
      }

    };
  }

}
```

1.2.5. Camel properties

Spring Boot auto-configuration automatically connects to Spring Boot external configuration (which may contain properties placeholders, OS environment variables or system properties) with the Camel properties support. It basically means that any property defined in **application.properties** file:

```
route.from = jms:invoices
```

Or set via system property:

```
java -Droute.to=jms:processed.invoices -jar mySpringApp.jar
```

can be used as placeholders in Camel route:

```
@Component
public class MyRouter extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("{{route.from}}").to("{{route.to}}");
  }

}
```

1.2.6. Custom Camel context configuration

If you want to perform some operations on **CamelContext** bean created by Camel auto-configuration, register **CamelContextConfiguration** instance in your Spring context:

```
@Configuration
public class MyAppConfig {

  @Bean
  CamelContextConfiguration contextConfiguration() {
    return new CamelContextConfiguration() {
      @Override
      void beforeApplicationStart(CamelContext context) {
```

```
        // your custom configuration goes here
      }
    };
  }

}
```

The method **beforeApplicationStart** will be called just before the Spring context is started, so the **CamelContext** instance passed to this callback is fully auto-configured. If you add multiple instances of **CamelContextConfiguration** into your Spring context, each instance is executed.

1.2.7. Auto-configured consumer and producer templates

Camel auto-configuration provides pre-configured **ConsumerTemplate** and **ProducerTemplate** instances. You can simply inject them into your Spring-managed beans:

```
@Component
public class InvoiceProcessor {

  @Autowired
  private ProducerTemplate producerTemplate;

  @Autowired
  private ConsumerTemplate consumerTemplate;

  public void processNextInvoice() {
    Invoice invoice = consumerTemplate.receiveBody("jms:invoices", Invoice.class);
    ...
    producerTemplate.sendBody("netty-http:http://invoicing.com/received/" + invoice.id());
  }

}
```

By default, consumer templates and producer templates come with the endpoint cache sizes set to 1000. You can change these values by modifying the following Spring properties:

```
camel.springboot.consumer-template-cache-size = 100
camel.springboot.producer-template-cache-size = 200
```

1.2.8. Auto-configured TypeConverter

Camel auto-configuration registers a **TypeConverter** instance named **typeConverter** in the Spring context.

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public long parseInvoiceValue(Invoice invoice) {
    String invoiceValue = invoice.grossValue();
    return typeConverter.convertTo(Long.class, invoiceValue);
```

```
  }

}
```

1.2.8.1. Spring type conversion API bridge

Spring comes with the powerful type conversion API. The Spring API is similar to the Camel type converter API. As both APIs are so similar, Camel Spring Boot automatically registers a bridge converter (**SpringTypeConverter**) that delegates to the Spring conversion API. This means that out-of-the-box Camel will treat Spring Converters like Camel ones. With this approach you can use both Camel and Spring converters accessed via Camel **TypeConverter** API:

```
@Component
public class InvoiceProcessor {

  @Autowired
  private TypeConverter typeConverter;

  public UUID parseInvoiceId(Invoice invoice) {
    // Using Spring's StringToUUIDConverter
    UUID id = invoice.typeConverter.convertTo(UUID.class, invoice.getId());
  }

}
```

Under the hood Camel Spring Boot delegates conversion to the Spring's **ConversionService** instances available in the application context. If no **ConversionService** instance is available, Camel Spring Boot auto-configuration will create one for you.

1.2.9. Keeping the application alive

Camel applications which have this feature enabled launch a new thread on startup for the sole purpose of keeping the application alive by preventing JVM termination. This means that after you start a Camel application with Spring Boot, your application waits for a **Ctrl+C** signal and does not exit immediately.

The controller thread can be activated using the **camel.springboot.main-run-controller** to **true**.

```
camel.springboot.main-run-controller = true
```

Applications using web modules (for example, applications that import the **org.springframework.boot:spring-boot-web-starter** module), usually don't need to use this feature because the application is kept alive by the presence of other non-daemon threads.

1.2.10. Adding XML routes

By default, you can put Camel XML routes in the classpath under the directory camel, which camel-spring-boot will auto-detect and include. You can configure the directory name or turn this off using the configuration option:

```
# turn off
camel.springboot.routes-include-pattern = false
```

```
# scan only in the com/foo/routes classpath
camel.springboot.routes-include-pattern = classpath:com/foo/routes/*.xml
```

The XML files should be Camel XML routes (**not <CamelContext>**) such as:

```
<routes xmlns="http://camel.apache.org/schema/spring">
  <route id="test">
    <from uri="timer://trigger"/>
    <transform>
      <simple>ref:myBean</simple>
    </transform>
    <to uri="log:out"/>
  </route>
</routes>
```

1.2.11. Testing the JUnit 5 way

For testing, Maven users will need to add the following dependencies to their **pom.xml**:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <version>3.1.5</version> <!-- Use the same version as your Spring Boot version -->
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-test-spring-junit5</artifactId>
  <version>4.0.0.redhat-00027</version> <!-- use the same version as your Camel core version -->
  <scope>test</scope>
</dependency>
```

To test a Camel Spring Boot application, annotate your test class(es) with **@CamelSpringBootTest**. This brings Camel's Spring Test support to your application, so that you can write tests using Spring Boot test conventions.

To get the **CamelContext** or **ProducerTemplate**, you can inject them into the class in the normal Spring manner, using **@Autowired**.

You can also use camel-test-spring-junit5 to configure tests declaratively. This example uses the **@MockEndpoints** annotation to auto-mock an endpoint:

```
@CamelSpringBootTest
@SpringBootApplication
@MockEndpoints("direct:end")
public class MyApplicationTest {

  @Autowired
  private ProducerTemplate template;

  @EndpointInject("mock:direct:end")
  private MockEndpoint mock;

  @Test
```

```
    public void testReceive() throws Exception {
        mock.expectedBodiesReceived("Hello");
        template.sendBody("direct:start", "Hello");
        mock.assertIsSatisfied();
    }

}
```

1.3. COMPONENT STARTERS

Camel Spring Boot supports the following Camel artifacts as Spring Boot Starters:

- Table 1.1, “Camel Components”
- Table 1.2, “Camel Data Formats”
- Table 1.3, “Camel Languages”
- Table 1.4, “Miscellaneous Extensions”

Table 1.1. Camel Components

Component	Artifact	Description
AMQP	camel-amqp-starter	Messaging with AMQP protocol using Apache QPid Client.
AWS Cloudwatch	camel-aws2-cw-starter	Sending metrics to AWS CloudWatch using AWS SDK version 2.x.
AWS DynamoDB	camel-aws2-ddb-starter	Store and retrieve data from AWS DynamoDB service using AWS SDK version 2.x.
AWS Kinesis	camel-aws2-kinesis-starter	Consume and produce records from and to AWS Kinesis Streams using AWS SDK version 2.x.
AWS Lambda	camel-aws2-lambda-starter	Manage and invoke AWS Lambda functions using AWS SDK version 2.x.
AWS S3 Storage Service	camel-aws2-s3-starter	Store and retrieve objects from AWS S3 Storage Service using AWS SDK version 2.x.
AWS Simple Notification System (SNS)	camel-aws2-sns-starter	Send messages to an AWS Simple Notification Topic using AWS SDK version 2.x.

Component	Artifact	Description
AWS Simple Queue Service (SQS)	camel-aws2-sqs-starter	Send and receive messages to/from AWS SQS service using AWS SDK version 2.x.
Azure ServiceBus	camel-azure-servicebus-starter	Send and receive messages to/from Azure Event Bus.
Azure Storage Blob Service	camel-azure-storage-blob-starter	Store and retrieve blobs from Azure Storage Blob Service using SDK v12.
Azure Storage Queue Service	camel-azure-storage-queue-starter	The azure-storage-queue component is used for storing and retrieving the messages to/from Azure Storage Queue using Azure SDK v12.
Bean	camel-bean-starter	Invoke methods of Java beans stored in Camel registry.
Bean Validator	camel-bean-validator-starter	Validate the message body using the Java Bean Validation API.
Browse	camel-browse-starter	Inspect the messages received on endpoints supporting BrowsableEndpoint.
Cassandra CQL	camel-cassandraql-starter	Integrate with Cassandra 2.0 using the CQL3 API (not the Thrift API). Based on Cassandra Java Driver provided by DataStax.
Control Bus	camel-controlbus-starter	Manage and monitor Camel routes.
Cron	camel-cron-starter	A generic interface for triggering events at times specified through the Unix cron syntax.
Crypto (JCE)	camel-crypto-starter	Sign and verify exchanges using the Signature Service of the Java Cryptographic Extension (JCE).

Component	Artifact	Description
CXF	camel-cxf-soap-starter	Expose SOAP WebServices using Apache CXF or connect to external WebServices using CXF WS client.
Data Format	camel-dataformat-starter	Use a Camel Data Format as a regular Camel Component.
Dataset	camel-dataset-starter	Provide data for load and soak testing of your Camel application.
Direct	camel-direct-starter	Call another endpoint from the same Camel Context synchronously.
Elastic Search	camel-elasticsearch-starter	Send requests to ElasticSearch via Java Client API.
FHIR	camel-fhir-starter	Exchange information in the healthcare domain using the FHIR (Fast Healthcare Interoperability Resources) standard.
File	camel-file-starter	Read and write files.
FTP	camel-ftp-starter	Upload and download files to/from FTP servers.
Google BigQuery	camel-google-bigquery-starter	Google BigQuery data warehouse for analytics.
Google Pubsub	camel-google-pubsub-starter	Send and receive messages to/from Google Cloud Platform PubSub Service.
gRPC	camel-grpc-starter	Expose gRPC endpoints and access external gRPC endpoints.
HTTP	camel-http-starter	Send requests to external HTTP servers using Apache HTTP Client 4.x.
Infinispan	camel-infinispan-starter	Read and write from/to Infinispan distributed key/value store and data grid.

Component	Artifact	Description
Infinispan Embedded	camel-infinispan-embedded-starter	Read and write from/to Infinispan distributed key/value store and data grid.
JDBC	camel-jdbc-starter	Access databases through SQL and JDBC.
Jira	camel-jira-starter	Interact with JIRA issue tracker.
JMS	camel-jms-starter	Sent and receive messages to/from a JMS Queue or Topic.
JPA	camel-jpa-starter	Store and retrieve Java objects from databases using Java Persistence API (JPA).
JSLT	camel-jslt-starter	Query or transform JSON payloads using an JSLT.
Kafka	camel-kafka-starter	Sent and receive messages to/from an Apache Kafka broker.
Kamelet	camel-kamelet-starter	To call Kamelets
Language	camel-language-starter	Execute scripts in any of the languages supported by Camel.
LDAP	camel-ldap-starter	Perform searches on LDAP servers.
Log	camel-log-starter	Log messages to the underlying logging mechanism.
Mail	camel-mail-starter	Send and receive emails using imap, pop3 and smtp protocols.
Mail Microsoft OAuth	camel-mail-microsoft-oauth-starter	Camel Mail OAuth2 Authenticator for Microsoft Exchange Online
MapStruct	camel-mapstruct-starter	Type Conversion using Mapstruct
Master	camel-master-starter	Have only a single consumer in a cluster consuming from a given endpoint; with automatic failover if the JVM dies.

Component	Artifact	Description
Micrometer	camel-micrometer-starter	Collect various metrics directly from Camel routes using the Micrometer library.
Minio	camel-minio-starter	Store and retrieve objects from Minio Storage Service using Minio SDK.
MLLP	camel-mllp-starter	Communicate with external systems using the MLLP protocol.
Mock	camel-mock-starter	Test routes and mediation rules using mocks.
MongoDB	camel-mongodb-starter	Perform operations on MongoDB documents and collections.
MyBatis	camel-mybatis-starter	Performs a query, poll, insert, update or delete in a relational database using MyBatis.
Netty	camel-netty-starter	Socket level networking using TCP or UDP with Netty 4.x.
Netty HTTP	camel-netty-http-starter	Netty HTTP server and client using the Netty 4.x.
Paho	camel-paho-starter	Communicate with MQTT message brokers using Eclipse Paho MQTT Client.
Paho MQTT 5	camel-paho-mqtt5-starter	Communicate with MQTT message brokers using Eclipse Paho MQTT v5 Client.
Platform HTTP	camel-platform-http-starter	Expose HTTP endpoints using the HTTP server available in the current platform.
Quartz	camel-quartz-starter	Schedule sending of messages using the Quartz 2.x scheduler.
Ref	camel-ref-starter	Route messages to an endpoint looked up dynamically by name in the Camel Registry.

Component	Artifact	Description
REST	camel-rest-starter	Expose REST services or call external REST services.
Saga	camel-saga-starter	Execute custom actions within a route using the Saga EIP.
Salesforce	camel-salesforce-starter	Communicate with Salesforce using Java DTOs.
SAP	camel-sap-starter	Uses the SAP Java Connector (SAP JCo) library to facilitate bidirectional communication with SAP and the SAP IDoc library to facilitate the transmission of documents in the Intermediate Document (IDoc) format.
Scheduler	camel-scheduler-starter	Generate messages in specified intervals using java.util.concurrent.ScheduledExecutorService.
SEDA	camel-seda-starter	Asynchronously call another endpoint from any Camel Context in the same JVM.
Servlet	camel-servlet-starter	Serve HTTP requests by a Servlet.
Slack	camel-slack-starter	Send and receive messages to/from Slack.
SNMP	camel-snmp-starter	Receive traps and poll SNMP (Simple Network Management Protocol) capable devices.
Spring Batch	camel-spring-batch-starter	Send messages to Spring Batch for further processing.
Spring JDBC	camel-spring-jdbc-starter	Access databases through SQL and JDBC with Spring Transaction support.
Spring LDAP	camel-spring-ldap-starter	Perform searches in LDAP servers using filters as the message payload.

Component	Artifact	Description
Spring RabbitMQ	camel-spring-rabbitmq-starter	Send and receive messages from RabbitMQ using Spring RabbitMQ client.
Spring Redis	camel-spring-redis-starter	Send and receive messages from Redis.
Spring Webservice	camel-spring-ws-starter	You can use this component to integrate with Spring Web Services. It offers client-side support for accessing web services and server-side support for creating your contract-first web services.
SQL	camel-sql-starter	Perform SQL queries using Spring JDBC.
Stub	camel-stub-starter	Stub out any physical endpoints while in development or testing.
Telegram	camel-telegram-starter	Send and receive messages acting as a Telegram Bot Telegram Bot API.
Timer	camel-timer-starter	Generate messages in specified intervals using java.util.Timer.
Validator	camel-validator-starter	Validate the payload using XML Schema and JAXP Validation.
Velocity	camel-velocity-starter	Transform messages using a Velocity template.
Vert.x HTTP Client	camel-vertx-http-starter	Send requests to external HTTP servers using Vert.x.
Vert.x WebSocket	camel-vertx-websocket-starter	Expose WebSocket endpoints and connect to remote WebSocket servers using Vert.x.
Webhook	camel-webhook-starter	Expose webhook endpoints to receive push notifications for other Camel components.
XJ	camel-xj-starter	Transform JSON and XML message using a XSLT.

Component	Artifact	Description
XSLT	camel-xslt-starter	Transforms XML payload using an XSLT template.
XSLT Saxon	camel-xslt-saxon-starter	Transform XML payloads using an XSLT template using Saxon.

Table 1.2. Camel Data Formats

Component	Artifact	Description
Avro	camel-avro-starter	Serialize and deserialize messages using Apache Avro binary data format.
Avro Jackson	camel-jackson-avro-starter	Marshal POJOs to Avro and back using Jackson.
Bindy	camel-bindy-starter	Marshal and unmarshal between POJOs and key-value pair (KVP) format using Camel Bindy
HL7	camel-hl7-starter	Marshal and unmarshal HL7 (Health Care) model objects using the HL7 MLLP codec.
JacksonXML	camel-jacksonxml-starter	Unmarshal a XML payloads to POJOs and back using XMLMapper extension of Jackson.
JAXB	camel-jaxb-starter	Unmarshal XML payloads to POJOs and back using JAXB2 XML marshalling standard.
JSON Gson	camel-gson-starter	Marshal POJOs to JSON and back using Gson
JSON Jackson	camel-jackson-starter	Marshal POJOs to JSON and back using Jackson
Protobuf Jackson	camel-jackson-protobuf-starter	Marshal POJOs to Protobuf and back using Jackson.
SOAP	camel-soap-starter	Marshal Java objects to SOAP messages and back.

Component	Artifact	Description
Zip File	camel-zipfile-starter	Compression and decompress streams using java.util.zip.ZipStream.

Table 1.3. Camel Languages

Language	Artifact	Description
Constant	camel-core-starter	A fixed value set only once during the route startup.
CSimple	camel-core-starter	Evaluate a compiled simple expression.
ExchangeProperty	camel-core-starter	Gets a property from the Exchange.
File	camel-core-starter	File related capabilities for the Simple language.
Header	camel-core-starter	Gets a header from the Exchange.
JQ	camel-jq-starter	Evaluates a JQ expression against a JSON message body.
JSONPath	camel-jsonpath-starter	Evaluates a JSONPath expression against a JSON message body.
Ref	camel-core-starter	Uses an existing expression from the registry.
Simple	camel-core-starter	Evaluates a Camel simple expression.
Tokenize	camel-core-starter	Tokenize text payloads using delimiter patterns.
XML Tokenize	camel-xml-jaxp-starter	Tokenize XML payloads.
XPath	camel-xpath-starter	Evaluates an XPath expression against an XML payload.

Language	Artifact	Description
XQuery	camel-saxon-starter	Query and/or transform XML payloads using XQuery and Saxon.

Table 1.4. Miscellaneous Extensions

Extensions	Artifact	Description
Kamelet Main	camel-kamelet-main-starter	Main to run Kamelet standalone
Openapi Java	camel-openapi-java-starter	Rest-dsl support for using openapi doc
OpenTelemetry	camel-opentelemetry-starter	Distributed tracing using OpenTelemetry
Spring Security	camel-spring-security-starter	Security using Spring Security
YAML DSL	camel-yaml-dsl-starter	Camel DSL with YAML

1.4. STARTER CONFIGURATION

Clear and accessible configuration is a crucial part of any application. Camel starters fully support Spring Boot's external configuration mechanism. You can also configure them through Spring Beans for more complex use cases.

1.4.1. Using External Configuration

Internally, every starter is configured through Spring Boot's ConfigurationProperties. Each configuration parameter can be set in various ways (**application.[properties|json|yaml]** files, command line arguments, environments variables etc.). Parameters have the form of **camel.[component|language|dataformat].[name].[parameter]**

For example to configure the URL of the MQTT5 broker you can set:

```
camel.component.paho-mqtt5.broker-url=tcp://localhost:61616
```

Or to configure the **delimeter** of the CSV dataformat to be a semicolon(;) you can set:

```
camel.dataformat.csv.delimiter=;
```

Camel will use the Type Converter mechanism when setting properties to the desired type.

You can refer to beans in the Registry using the **#bean:name**:

```
camel.component.jms.transactionManager=#bean:myjtaTransactionManager
```

The **Bean** would be typically created in Java:

```
@Bean("myjtaTransactionManager")
public JmsTransactionManager myjtaTransactionManager(PooledConnectionFactory pool) {
    JmsTransactionManager manager = new JmsTransactionManager(pool);
    manager.setDefaultTimeout(45);
    return manager;
}
```

Beans can also be created in configuration files but this is not recommended for complex use cases.

1.4.2. Using Beans

Starters can also be created and configured via Spring Beans. Before creating a starter , Camel will first lookup it up in the Registry by it's name if it already exists. For example to configure a Kafka component:

```
@Bean("kafka")
public KafkaComponent kafka(KafkaConfiguration kafkaconfiguration){
    return ComponentsBuilderFactory.kafka()
                    .brokers("{{kafka.host}}:{{kafka.port}}")
                    .build();
}
```

The **Bean** name has to be equal to that of the Component, Dataformat or Language that you are configuring. If the **Bean** name isn't specified in the annotation it will be set to the method name.

Typical Camel Spring Boot projects will use a combination of external configuration and Beans to configure an application. For more examples on how to configure your Camel Spring Boot project, please see the example repository.

1.5. GENERATING A CAMEL FOR SPRING BOOT APPLICATION USING MAVEN

You can generate a Red Hat build of Apache Camel for Spring Boot application using the Maven archetype **org.apache.camel.archetypes:camel-archetype-spring-boot:4.0.0.redhat-00036**.

Procedure

1. Run the following command:

```
mvn archetype:generate \
 -DarchetypeGroupId=org.apache.camel.archetypes \
 -DarchetypeArtifactId=camel-archetype-spring-boot \
 -DarchetypeVersion=4.0.0.redhat-00036 \
 -DgroupId=com.redhat \
 -DartifactId=csb-app \
 -Dversion=1.0-SNAPSHOT \
 -DinteractiveMode=false
```

2. Build the application:

```
mvn package -f csb-app/pom.xml
```

3. Run the application:

```
java -jar csb-app/target/csb-app-1.0-SNAPSHOT.jar
```

4. Verify that the application is running by examining the console log for the *Hello World* output which is generated by the application.

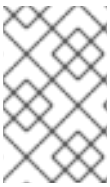
```
com.redhat.MySpringBootApplication      : Started MySpringBootApplication in 3.514
seconds (JVM running for 4.006)
Hello World
Hello World
```

1.6. DEPLOYING A CAMEL SPRING BOOT APPLICATION TO OPENSHIFT

This guide demonstrates how to deploy a Camel Spring Boot application to OpenShift.

Prerequisites

- You have access to the OpenShift cluster.
- The OpenShift **oc** CLI client is installed or you have access to the OpenShift Container Platform web console.



NOTE

The certified OpenShift Container platforms are listed in the Camel for Spring Boot Supported Configurations. The Red Hat OpenJDK 11 (ubi8/openjdk-11) container image is used in the following example.

Procedure

1. Generate a Camel for Spring Boot application using Maven by following the instructions in section 1.5 Generating a Camel for Spring Boot application using Maven of this guide.
2. Under the directory which the modified pom.xml exists, execute the following command.

```
mvn clean -DskipTests oc:deploy -Popenshift
```

3. Verify that the CSB application is running on the pod.

```
oc logs -f dc/csb-app
```

1.7. APPLYING PATCH TO RED HAT BUILD OF APACHE CAMEL FOR SPRING BOOT

Using the new **patch-maven-plugin** mechanism, you can apply a patch to your Red Hat Red Hat build of Apache Camel for Spring Boot application. This mechanism allows you to change the individual versions provided by different Red Hat application BOMS, for example, **camel-spring-boot-bom**.

The purpose of the **patch-maven-plugin** is to update the versions of the dependencies listed in the Camel on Spring Boot BOM to the versions specified in the patch metadata that you wish to apply to your applications.

The patch-maven-plugin performs the following operations:

- Retrieve the patch metadata related to current Red Hat application BOMs.
- Apply the version changes to <dependencyManagement> imported from the BOMs.

After the **patch-maven-plugin** fetches the metadata, it iterates through all managed and direct dependencies of the project where the plugin was declared and replaces the dependency versions (if they match) using CVE/patch metadata. After the versions are replaced, the Maven build continues and progresses through standard Maven project stages.

Procedure

The following procedure explains how to apply the patch to your application.

1. Add **patch-maven-plugin** to your project's **pom.xml** file. The version of the **patch-maven-plugin** must be the same as the version of the Camel on Spring Boot BOM.

```
<build>
  <plugins>
    <<plugin>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>patch-maven-plugin</artifactId>
      <version>${camel-spring-boot-version}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

2. When you run any of the **mvn clean deploy**, **mvn validate**, or **mvn dependency:tree** commands, the plugin searches through the project modules to check if the modules use the Red Hat Red Hat build of Apache Camel for Spring Boot BOM. Only the following is the supported BOM:
   - **com.redhat.camel.springboot.platform:camel-spring-boot-bom**: for Red Hat build of Apache Camel for Spring Boot BOM
3. If the plugin does not find the above BOM, the plugin displays the following messages:

```
$ mvn clean install

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] No project in the reactor uses Camel on Spring Boot product BOM. Skipping patch processing.
[INFO] [PATCH] Done in 7ms

============================================
```

4. If the correct BOM is used, the patch metadata is found, but without any patches.

```
$ mvn clean install

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - redhat-ga-repository: http://maven.repository.redhat.com/ga/
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
Downloading from redhat-ga-repository:
http://maven.repository.redhat.com/ga/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/maven-metadata.xml
Downloading from central:
https://repo.maven.apache.org/maven2/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/maven-metadata.xml
[INFO] [PATCH] Resolved patch descriptor:
/path/to/.m2/repository/com/redhat/camel/springboot/platform/redhat-camel-spring-boot-patch-metadata/3.20.1.redhat-00043/redhat-camel-spring-boot-patch-metadata-3.20.1.redhat-00043.xml
[INFO] [PATCH] Patch metadata found for com.redhat.camel.springboot.platform/camel-spring-boot-bom/[3.20,3.21)
[INFO] [PATCH] Done in 938ms

================================================
```

5. The **patch-maven-plugin** attempts to fetch this Maven metadata.

   - For the projects with Camel Spring Boot BOM, the **com.redhat.camel.springboot.platform:redhat-camel-spring-boot-patch-metadata/maven-metadata.xml** is resolved. This XML data is the metadata for the artifact with the **com.redhat.camel.springboot.platform:redhat-camel-spring-boot-patch-metadata:RELEASE** coordinates.

     Example metadata generated by Maven

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <groupId>com.redhat.camel.springboot.platform</groupId>
  <artifactId>redhat-camel-spring-boot-patch-metadata</artifactId>
  <versioning>
    <release>3.20.1.redhat-00041</release>
    <versions>
      <version>3.20.1.redhat-00041</version>
    </versions>
    <lastUpdated>20230322103858</lastUpdated>
  </versioning>
</metadata>
```

6. The **patch-maven-plugin** parses the metadata to select the version which applies to the current project. This action is possible only for the Maven projects using Camel on Spring Boot BOM with the specific version. Only the metadata that matches the version range or later is applicable, and it fetches only the latest version of the metadata.

7. The **patch-maven-plugin** collects a list of remote Maven repositories for downloading the patch metadata identified by **groupId**, **artifactId**, and **version** found in previous steps. These Maven repositories are listed in the project's **<repositories>** elements in the active profiles, and also the repositories from the **settings.xml** file.

```
$ mvn clean install
[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 2 project repositories
[INFO] [PATCH]  - MRRC-GA: https://maven.repository.redhat.com/ga
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
```

8. Whether the metadata comes from a remote repository, local repository, or ZIP file, it is analyzed by the **patch-maven-plugin**. The fetched metadata contains a list of CVEs, and for each CVE, we have a list of the affected Maven artifacts (specified by glob patterns and version ranges) together with a version that contains a fix for a given CVE. For example,

```
<?xml version="1.0" encoding="UTF-8" ?>

<<metadata xmlns="urn:redhat:patch-metadata:1">
   <product-bom groupId="com.redhat.camel.springboot.platform" artifactId="camel-spring-
boot-bom" versions="[3.20,3.21)" />
   <cves>
   </cves>
   <fixes>
      <fix id="HF0-1" description="logback-classic (Example) - Version Bump">
         <affects groupId="ch.qos.logback" artifactId="logback-classic" versions="[1.0,1.3.0)"
fix="1.3.0" />
      </fix>
   </fixes>
</metadata>
```

9. Finally a list of fixes specified in patch metadata is consulted when iterating over all managed dependencies in the current project. These dependencies (and managed dependencies) that match are changed to fixed versions. For example:

```
$ mvn dependency:tree

[INFO] Scanning for projects...
[INFO]

========== Red Hat Maven patching ==========

[INFO] [PATCH] Reading patch metadata and artifacts from 3 project repositories
[INFO] [PATCH]  - redhat-ga-repository: http://maven.repository.redhat.com/ga/
[INFO] [PATCH]  - local: file:///path/to/.m2/repository
[INFO] [PATCH]  - central: https://repo.maven.apache.org/maven2
[INFO] [PATCH] Resolved patch
descriptor:/path/to/.m2/repository/com/redhat/camel/springboot/platform/redhat-camel-spring-
boot-patch-metadata/3.20.1.redhat-00043/redhat-camel-spring-boot-patch-metadata-
3.20.1.redhat-00043.xml
[INFO] [PATCH] Patch metadata found for com.redhat.camel.springboot.platform/camel-
```

```
spring-boot-bom/[3.20,3.21)
[INFO] [PATCH]  - patch contains 1 patch fix
[INFO] [PATCH] Processing managed dependencies to apply patch fixes...
[INFO] [PATCH] - HF0-1: logback-classic (Example) - Version Bump
[INFO] [PATCH]   Applying change ch.qos.logback/logback-classic/[1.0,1.3.0) -> 1.3.0
[INFO] [PATCH]   Project com.test:yaml-routes
[INFO] [PATCH]    - managed dependency: ch.qos.logback/logback-classic/1.2.11 -> 1.3.0
[INFO] [PATCH] Done in 39ms

================================================
```

Skipping the patch

If you do not wish to apply a specific patch to your project, the **patch-maven-plugin** provides a **skip** option. Assuming that you have already added the **patch-maven-plugin** to the project's **pom.xml** file, and you do not wish to alter the versions, you can use one of the following method to skip the patch.

- Add the skip option to your project's **pom.xml** file as follows.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.redhat.camel.springboot.platform</groupId>
      <artifactId>patch-maven-plugin</artifactId>
      <version>${camel-spring-boot-version}</version>
      <extensions>true</extensions>
      <configuration>
        <skip>true</skip>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Or use the **-DskipPatch** option when running the **mvn** command as follows.

```
$ mvn clean install -DskipPatch
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.example:test-csb >-------------------------
[INFO] Building A Camel Spring Boot Route 1.0-SNAPSHOT
...
```

As shown in the above output, the **patch-maven-plugin** was not invoked, which resulted in the patch not being applied to the application.

1.8. CAMEL REST DSL OPENAPI MAVEN PLUGIN

The Camel REST DSL OpenApi Maven Plugin supports the following goals.

- camel-restdsl-openapi:generate - To generate consumer REST DSL RouteBuilder source code from OpenApi specification

- camel-restdsl-openapi:generate-with-dto - To generate consumer REST DSL RouteBuilder source code from OpenApi specification and with DTO model classes generated via the swagger-codegen-maven-plugin.

- camel-restdsl-openapi:generate-xml - To generate consumer REST DSL XML source code from OpenApi specification
- camel-restdsl-openapi:generate-xml-with-dto - To generate consumer REST DSL XML source code from OpenApi specification and with DTO model classes generated via the swagger-codegen-maven-plugin.
- camel-restdsl-openapi:generate-yaml - To generate consumer REST DSL YAML source code from OpenApi specification
- camel-restdsl-openapi:generate-yaml-with-dto - To generate consumer REST DSL YAML source code from OpenApi specification and with DTO model classes generated via the swagger-codegen-maven-plugin.

1.8.1. Adding plugin to Maven pom.xml

This plugin can be added to your Maven **pom.xml** file by adding it to the **plugins** section, for example in a Spring Boot application:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>

    <plugin>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-restdsl-openapi-plugin</artifactId>
      <version>{CamelCommunityVersion}</version>
    </plugin>

  </plugins>
</build>
```

The plugin can then be executed using its prefix **camel-restdsl-openapi** as shown below.

```
$mvn camel-restdsl-openapi:generate
```

1.8.2. camel-restdsl-openapi:generate

The goal of the Camel REST DSL OpenApi Maven Plugin is used to generate REST DSL RouteBuilder implementation source code from Maven.

1.8.3. Options

The plugin supports the following options which can be configured from the command line (use **-D** syntax), or defined in the **pom.xml** file in the **configuration** tag.

Parameter	Default Value	Description
skip	**false**	Set to **true** to skip code generation

Parameter	Default Value	Description
filterOperation		Used for including only the operation ids specified. Multiple ids can be separated by comma. Wildcards can be used, eg **find*** to include all operations starting with **find**.
specificationUri	**src/spec/openapi.json**	URI of the OpenApi specification, supports filesystem paths, HTTP and classpath resources, by default **src/spec/openapi.json** within the project directory. Supports JSON and YAML.
auth		Adds authorization headers when fetching the OpenApi specification definitions remotely. Pass in a URL-encoded string of name:header with a comma separating multiple values.
className	from **title** or **RestDslRoute**	Name of the generated class, taken from the OpenApi specification title or set to **RestDslRoute** by default
packageName	from **host** or **rest.dsl.generated**	Name of the package for the generated class, taken from the OpenApi specification host value or **rest.dsl.generated** by default
indent	**" "**	Which indenting character(s) to use, by default four spaces, you can use **\t** to signify tab character
outputDirectory	**generated-sources/restdsl-openapi**	Where to place the generated source file, by default **generated-sources/restdsl-openapi** within the project directory
destinationGenerator		Fully qualified class name of the class that implements **org.apache.camel.generator.openapi.DestinationGenerator** interface for customizing destination endpoint

Parameter	Default Value	Description
destinationToSyntax	**direct:${operationId}**	The default to syntax for the to uri, which is to use the direct component.
restConfiguration	**true**	Whether to include generation of the rest configuration with detected rest component to be used.
apiContextPath		Define openapi endpoint path if **restConfiguration** is set to true.
clientRequestValidation	**false**	Whether to enable request validation.
basePath		Overrides the api base path as defined in the OpenAPI specification.
requestMappingValues	/**	Allows generation of custom **RequestMapping** mapping values. Multiple mapping values can be passed as: **<requestMappingValues> <param>/my-api-path/</param> <param>/my-other-path/</param> </requestMappingValues>**

1.8.4. Spring Boot Project with Servlet component

If the Maven project is a Spring Boot project and **restConfiguration** is enabled and the servlet component is being used as REST component, then this plugin will autodetect the package name (if packageName has not been explicitly configured) where the **@SpringBootApplication** main class is located, and use the same package name for generating Rest DSL source code and a needed **CamelRestController** support class.

1.8.5. camel-restdsl-openapi:generate-with-dto

Works as **generate** goal but also generates DTO model classes by automatic executing the swagger-codegen-maven-plugin to generate java source code of the DTO model classes from the OpenApi specification.

This plugin has been scoped and limited to only support a good effort set of defaults for using the swagger-codegen-maven-plugin to generate the model DTOs. If you need more power and flexibility then use the Swagger Codegen Maven Plugin directly to generate the DTO and not this plugin.

The DTO classes may require additional dependencies such as:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
</dependency>
<dependency>
  <groupId>org.threeten</groupId>
  <artifactId>threetenbp</artifactId>
  <version>1.6.8</version>
</dependency>
```

1.8.6. Options

The plugin supports the following **additional** options

Parameter	Default Value	Description
swaggerCodegenMavenPluginVersion	3.0.36	The version of the **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven plugin to be used.
modelOutput		Target output path (default is ${project.build.directory}/generated-sources/openapi)
modelPackage	**io.swagger.client.model**	The package to use for generated model objects/classes
modelNamePrefix		Sets the pre- or suffix for model classes and enums
modelNameSuffix		Sets the pre- or suffix for model classes and enums
modelWithXml	false	Enable XML annotations inside the generated models (only works with libraries that provide support for JSON and XML)
configOptions		Pass a map of language-specific parameters to **swagger-codegen-maven-plugin**

1.8.7. camel-restdsl-openapi:generate-xml

The **camel-restdsl-openapi:generate-xml** goal of the Camel REST DSL OpenApi Maven Plugin is used to generate REST DSL XML implementation source code from Maven.

1.8.8. Options

The plugin supports the following options which can be configured from the command line (use **-D** syntax), or defined in the **pom.xml** file in the **<configuration>** tag.

Parameter	Default Value	Description
skip	**false**	Set to **true** to skip code generation.
filterOperation		Used for including only the operation ids specified. Multiple ids can be separated by comma. Wildcards can be used, eg **find*** to include all operations starting with **find**.
specificationUri	**src/spec/openapi.json**	URI of the OpenApi specification, supports filesystem paths, HTTP and classpath resources, by default **src/spec/openapi.json** within the project directory. Supports JSON and YAML.
auth		Adds authorization headers when fetching the OpenApi specification definitions remotely. Pass in a URL-encoded string of name:header with a comma separating multiple values.
outputDirectory	**generated-sources/restdsl-openapi**	Where to place the generated source file, by default **generated-sources/restdsl-openapi** within the project directory
fileName	**camel-rest.xml**	The name of the XML file as output.
blueprint	**false**	If enabled generates OSGi Blueprint XML instead of Spring XML.

Parameter	Default Value	Description
destinationGenerator		Fully qualified class name of the class that implements **org.apache.camel.generator.openapi.DestinationGenerator** interface for customizing destination endpoint
destinationToSyntax	**direct:${operationId}**	The default to syntax for the to uri, which is to use the direct component.
	restConfiguration	**true**
Whether to include generation of the rest configuration with detected rest component to be used.	**apiContextPath**	
Define openapi endpoint path if **restConfiguration** is set to **true**.	**clientRequestValidation**	**false**
Whether to enable request validation.	**basePath**	
Overrides the api base path as defined in the OpenAPI specification.	**requestMappingValues**	/**

1.8.9. camel-restdsl-openapi:generate-xml-with-dto

Works as **generate-xml** goal but also generates DTO model classes by automatic executing the swagger-codegen-maven-plugin to generate java source code of the DTO model classes from the OpenApi specification.

This plugin has been scoped and limited to only support a good effort set of defaults for using the swagger-codegen-maven-plugin to generate the model DTOs. If you need more power and flexibility then use the Swagger Codegen Maven Plugin directly to generate the DTO and not this plugin.

The DTO classes may require additional dependencies such as:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
```

```
</dependency>
<dependency>
  <groupId>org.threeten</groupId>
  <artifactId>threetenbp</artifactId>
  <version>1.6.8</version>
</dependency>
```

1.8.10. Options

The plugin supports the following **additional** options

Parameter	Default Value	Description
swaggerCodegenMavenPluginVersion	3.0.36	The version of the **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven plugin to be used.
modelOutput		Target output path (default is ${project.build.directory}/generated-sources/openapi)
modelPackage	**io.swagger.client.model**	The package to use for generated model objects/classes
modelNamePrefix		Sets the pre- or suffix for model classes and enums
modelNameSuffix		Sets the pre- or suffix for model classes and enums
modelWithXml	false	Enable XML annotations inside the generated models (only works with libraries that provide support for JSON and XML)
configOptions		Pass a map of language-specific parameters to **swagger-codegen-maven-plugin**

1.8.11. camel-restdsl-openapi:generate-yaml

The **camel-restdsl-openapi:generate-yaml** goal of the Camel REST DSL OpenApi Maven Plugin is used to generate REST DSL YAML implementation source code from Maven.

1.8.12. Options

The plugin supports the following options which can be configured from the command line (use **-D** syntax), or defined in the **pom.xml** file in the **<configuration>** tag.

Parameter	Default Value	Description
skip	**false**	Set to **true** to skip code generation.
filterOperation		Used for including only the operation ids specified. Multiple ids can be separated by comma. Wildcards can be used, eg **find*** to include all operations starting with **find**.
specificationUri	**src/spec/openapi.json**	URI of the OpenApi specification, supports filesystem paths, HTTP and classpath resources, by default **src/spec/openapi.json** within the project directory. Supports JSON and YAML.
auth		Adds authorization headers when fetching the OpenApi specification definitions remotely. Pass in a URL-encoded string of name:header with a comma separating multiple values.
outputDirectory	**generated-sources/restdsl-openapi**	Where to place the generated source file, by default **generated-sources/restdsl-openapi** within the project directory
fileName	**camel-rest.xml**	The name of the XML file as output.
destinationGenerator		Fully qualified class name of the class that implements **org.apache.camel.generator.openapi.DestinationGenerator** interface for customizing destination endpoint
destinationToSyntax	**direct:${operationId}**	The default to syntax for the to uri, which is to use the direct component.
	restConfiguration	**true**

Parameter	Default Value	Description
Whether to include generation of the rest configuration with detected rest component to be used.	**apiContextPath**	
Define openapi endpoint path if **restConfiguration** is set to **true**.	**clientRequestValidation**	**false**
Whether to enable request validation.	**basePath**	
Overrides the api base path as defined in the OpenAPI specification.	**requestMappingValues**	/**

1.8.13. camel-restdsl-openapi:generate-yaml-with-dto

Works as **generate-yaml** goal but also generates DTO model classes by automatic executing the swagger-codegen-maven-plugin to generate java source code of the DTO model classes from the OpenApi specification.

This plugin has been scoped and limited to only support a good effort set of defaults for using the **swagger-codegen-maven-plugin** to generate the model DTOs. If you need more power and flexibility then use the Swagger Codegen Maven Plugin directly to generate the DTO and not this plugin.

The DTO classes may require additional dependencies such as:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
</dependency>
<dependency>
  <groupId>org.threeten</groupId>
  <artifactId>threetenbp</artifactId>
  <version>1.6.8</version>
</dependency>
```

1.8.14. Options

The plugin supports the following **additional** options

Parameter	Default Value	Description
swaggerCodegenMavenPluginVersion	3.0.36	The version of the **io.swagger.codegen.v3:swagger-codegen-maven-plugin** maven plugin to be used.
modelOutput		Target output path (default is ${project.build.directory}/generated-sources/openapi)
modelPackage	**io.swagger.client.model**	The package to use for generated model objects/classes
modelNamePrefix		Sets the pre- or suffix for model classes and enums
modelNameSuffix		Sets the pre- or suffix for model classes and enums
modelWithXml	false	Enable XML annotations inside the generated models (only works with libraries that provide support for JSON and XML)
configOptions		Pass a map of language-specific parameters to **swagger-codegen-maven-plugin**

1.9. SUPPORT FOR FIPS COMPLIANCE

You can install an OpenShift Container Platform cluster that uses FIPS Validated / Modules in Process cryptographic libraries on the x86_64 architecture.

For the Red Hat Enterprise Linux CoreOS (RHCOS) machines in your cluster, this change applies when the machines deploy based on the status of an option in the install-config.yaml file, which governs the cluster options that users can change during cluster deployment. With Red Hat Enterprise Linux (RHEL) machines, you must enable FIPS mode when installing the operating system on the machines you plan to use as worker machines. These configuration methods ensure that your cluster meets the requirements of a FIPS compliance audit. Only FIPS Validated / Modules in Process cryptography packages are enabled before the initial system boot.

Because you must enable FIPS before your cluster's operating system boots for the first time, you cannot enable FIPS after you deploy a cluster.

1.9.1. FIPS validation in OpenShift Container Platform

OpenShift Container Platform uses certain FIPS Validated / Modules in Process modules within RHEL and RHCOS for its operating system components. For example, when users SSH into OpenShift Container Platform clusters and containers, those connections are properly encrypted.

OpenShift Container Platform components are written in Go and built with Red Hat's Golang compiler. When you enable FIPS mode for your cluster, all OpenShift Container Platform components that require cryptographic signing call RHEL and RHCOS cryptographic libraries.

For more details about FIPS, see FIPS mode attributes and limitations

For details on deploying Camel Spring Boot on OpenShift, see How to deploy a Camel Spring Boot application to OpenShift?

Details about supported configurations can be found at, Camel for Spring Boot Supported Configurations

CHAPTER 2. USING CAMEL WITH SPRING XML

Spring XML

Using Camel with Spring XML files, is a way, of using XML DSL with Camel. Camel has historically been using Spring XML for a long time. The Spring framework started with XML files as a popular and common configuration for building Spring applications.

Example of Spring application

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
    ">

    <camelContext xmlns="http://camel.apache.org/schema/spring">
        <route>
            <from uri="direct:a"/>
            <choice>
                <when>
                    <xpath>$foo = 'bar'</xpath>
                    <to uri="direct:b"/>
                </when>
                <when>
                    <xpath>$foo = 'cheese'</xpath>
                    <to uri="direct:c"/>
                </when>
                <otherwise>
                    <to uri="direct:d"/>
                </otherwise>
            </choice>
        </route>
    </camelContext>

</beans>
```

2.1. SPECIFYING CAMEL ROUTES USING SPRING XML

You can use Spring XML files to specify Camel routes using XML DSL as shown:

```
<camelContext id="camel-A" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="seda:start"/>
    <to uri="mock:result"/>
  </route>
</camelContext>
```

2.2. CONFIGURING COMPONENTS AND ENDPOINTS

You can configure your Component or Endpoint instances in your Spring XML as follows in this example.

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
</camelContext>

<bean id="jmsConnectionFactory"
class="org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp:someserver:61616"/>
</bean>
<bean id="jms" class="org.apache.camel.component.jms.JmsComponent">
  <property name="connectionFactory">
    <bean class="org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp:someserver:61616"/>
      </bean>
  </property>
</bean>
```

This allows you to configure a component using any name, but its common to use the same name, for example, **jms**. Then you can refer to the component using **jms:destinationName**.

This works by the Camel fetching components from the Spring context for the scheme name you use for Endpoint URIs.

2.3. USING JAVA DSL WITH SPRING XML FILES

You can use Java Code to define your RouteBuilder implementations. These are defined as beans in spring and then referenced in your camel context, as shown:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <routeBuilder ref="myBuilder"/>
</camelContext>

<bean id="myBuilder" class="org.apache.camel.spring.example.test1.MyRouteBuilder"/>
```

2.4. USING PACKAGE SCANNING

Camel also provides a powerful feature that allows for the automatic discovery and initialization of routes in given packages. This is configured by adding tags to the camel context in your spring context definition, specifying the packages to be recursively searched for **RouteBuilder** implementations. To use this feature add a <package></package> tag specifying a comma separated list of packages that should be searched. For example,

```
<camelContext>
  <packageScan>
    <package>com.foo</package>
    <excludes>**.*Excluded*</excludes>
    <includes>**.*</includes>
  </packageScan>
</camelContext>
```

This scans for RouteBuilder classes in the **com.foo** and the sub-packages.

You can also filter the classes with includes or excludes such as:

```
<camelContext>
```

```
  <packageScan>
    <package>com.foo</package>
    <excludes>**.*Special*</excludes>
  </packageScan>
</camelContext>
```

This skips the classes that has Special in the name. Exclude patterns are applied before the include patterns. If no include or exclude patterns are defined then all the Route classes discovered in the packages are returned.

? matches one character, ***** matches zero or more characters, ****** matches zero or more segments of a fully qualified name.

2.5. USING CONTEXT SCANNING

You can allow Camel to scan the container context, for example, the Spring **ApplicationContext** for route builder instances. This allows you to use the Spring **<component-scan>** feature and have Camel pickup any RouteBuilder instances which was created by Spring in its scan process.

```
<!-- enable Spring @Component scan -->
<context:component-scan base-package="org.apache.camel.spring.issues.contextscan"/>

<camelContext xmlns="http://camel.apache.org/schema/spring">
    <!-- and then let Camel use those @Component scanned route builders -->
    <contextScan/>
</camelContext>
```

This allows you to just annotate your routes using the Spring **@Component** and have those routes included by Camel:

```
@Component
public class MyRoute extends RouteBuilder {

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

You can also use the ANT style for inclusion and exclusion, as mentioned above in the package scan section.