Apache Camel





This work is licensed under the Creative Commons Attribution 2.0 Belgium License.

To view a copy of this license, visit http://creativecommons.org/licenses/by/2.0/be/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action

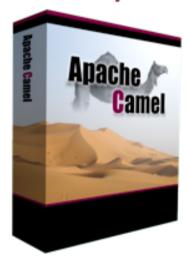
Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



Introduction

Download it Today!



Apache Camel is a powerful open source integration framework based on known Enterprise Integration Patterns with powerful Bean Integration.

Camel lets you create the Enterprise Integration Patterns to implement routing and mediation rules in either a Java based Domain Specific Language (or Fluent API), via Spring based Xml Configuration files or via the Scala DSL. This means you get smart completion of routing rules in your IDE whether in your Java, Scala or XML editor.

Apache Camel uses URIs so that it can easily work directly with any kind of Transport or messaging model such as HTTP, ActiveMQ, JMS, JBI, SCA, MINA or CXF Bus API together with working with pluggable Data Format options. Apache Camel is a small library which has minimal dependencies for easy embedding in any Java application. Apache Camel lets you work with the same API regardless which kind of Transport used, so learn the API once and you will be able to interact with all the Components that is provided out-of-the-box.

Apache Camel has powerful Bean Binding and integrated seamless with popular frameworks such as Spring and Guice.

Apache Camel has extensive Testing support allowing you to easily unit test your routes.

Open-Source

- Open-Source
 - Apache Software License
 - very liberal open-source license
 - Very active community
 - mailing lists
 - development

Enterprise Integration Patterns

- Enterprise Integration Patterns
 - Based on book by G. Hohpe & B. Woolf
 - Identifies common patterns in EAI solutions
 - Examples:
 - wiretap
 - content-based router
 - normalizer
 - Camel makes it easy to implement these patterns





Domain-specific language

- Routing can be defined in DSL
 - Java DSL
 Fluent API that allows code completion by your IDE
 - Spring DSL
 Custom namespace with XML tags for routing
 - Scala DSL

Adding some Scala features on top of the Java DSL

Domain-specific language

- Examples
 - in Java DSL

```
from("timer:test").to("log:test");
```

in Spring DSL

in Scala DSL

```
"timer:test" -> "log:test"
"timer:test" to "log:test"
```



URIs

- URIs to specify endpoints
 - camel-core provides

```
direct:, seda:, vm:timer:log:
```

- components can add others
 - camel-gae provides ghttp:, gtask:, gmail: ...
 - camel-mina provides tcp: and udp:
 - ... and over 80 others

(cc) BY

URIs

- URI parameters to configure endpoints
 - syntax : ?param1=value¶m2=value
 - caveat: & becomes & amp; in XML
- Example log: endpoint "log:example?level=DEBUG"

```
"log:example?level=DEBUG&showAll=true"
```

- "log:another-log?showBody=false&showBodyType=true&multiline=true"
- Example timer: endpoint in XML



Message Data

- Camel supports any kind of payload
- Pluggable handlers
 - TypeConverters allow you to easily convert
 - File to String, Stream to bytes, XML to String, ...
 - Data Formats support marshalling/unmarshalling
 - JAXB, Google Prototype Buffers, HL7, ...

Beans

- Camel supports the use of beans
 - for message processing
 - for defining expressions predicates
 - for routing
- Features
 - Intelligent bean binding
 - Several annotations to ease building beans for use in Camel





Testing

- Support for testing routes
 - camel-test for building unit tests
 - annotations and base classes
 - Testing endpoints
 - mock:
 - dataset:
 - test:
 - Use direct: or seda: to replace transports

Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



Getting started with Camel

- CamelContext
- Defining routes
- Camel API basics

CamelContext

- Runtime execution context
 - 'glue' for everything else
 - Registry, TypeConverters, Components, Endpoints
 - Languages and Data Formats
- Routes are added to the context as well
 - context allows for runtime control over routes with start() and stop() methods

CamelContext

- Create and start a new CamelContext
 - in Java code

```
CamelContext context = new DefaultCamelContext();
context.addRoutes(new MyRouteBuilder());
context.start();
```

CamelContext

- Create and start a new CamelContest
 - in a Spring XML file

```
<?xml version="1.0"?>
<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">
        <!-- routes can be added here -->
        </camelContext>
```

(cc)) BY

- Defining routes in a Java RouteBuilder
 - extend org.apache.camel.builder.RouteBuilder
 - implement the configure() method

```
public class MyRouteBuilder extends RouteBuilder {
    @Override
    public void configure() throws Exception {
        from("timer:test").to("log:test");
    }
}
```

- Defining routes in a Scala RouteBuilder
 - extend
 o.a.camel.scala.dsl.builder.RouteBuilder

```
package be.anova.course.camel
import org.apache.camel.scala.dsl.builder.RouteBuilder;
class MyRouteBuilder extends RouteBuilder {
   "timer:test" to "log:test"
}
```

- Defining routes in a Spring XML file
 - add <route/> elements to
 <amelContext/> ...



- Defining routes in a Spring XML file
 - ... or refer to existing Java/Scala RouteBuilders



Camel API basics

- Let's look at a few basic Camel API interfaces
 - Exchange
 - Message
 - Processor

Exchange

- Contains Messages while routing
- Consists of
 - exchange id
 - MEP
 - Properties
 - In Message
 - Out Message (also used for Faults)

Exchange

Examples

```
String id = exchange.getExchangeId();
ExchangePattern mep = exchange.getPattern();
Object property = exchange.getProperty("property.name");
String string =
     exchange.getProperty("property.name", String.class);
Message in = exchange.getIn();
if (exchange.hasOut()) {
   Message out = exchange.getOut();
if (exchange.isFailed()) {
   Exception exception = exchange.getException();
```

Message

- Used as In or Out message on Exchange
- Consists of
 - headers
 - body
 - attachments

Message

Examples

```
Object body = message.getBody();
String bodyAsString = message.getBody(String.class);
// will throw exception if conversion not possible
String mandatoryString = message.getBody(String.class);
if (message.hasAttachments()) {
   DataHandler attachment =
       message.getAttachment("attachment.name");
   message.addAttachment("attachement.new", attachment);
Object header = message.getHeader("header.name");
Integer headerAsInt =
           message.getHeader("header.name", Integer.class);
```

- Interface to create an Exchange processor
 - for consuming/handling an Exchange

```
• for message transformation
public class MyProcessor implements Processor {

private static final Log LOG = LogFactory.getLog(MyProcessor.class);

public void process(Exchange exchange) throws Exception {
    LOG.info("Handling " + exchange.getExchangeId());

    exchange.getOut().copyFrom(exchange.getIn());
    exchange.getOut().setBody("Message handled on " + new Date());
}
```

- Add a Processor in the Java DSL
 - process() method to add the Processor instance
 - processRef() to refer to Processor in the Registry

- Add a Processor in the Java DSL
 - with an anonymous inner class

```
public void configure() throws Exception {
   from("timer:test")
        .process(new Processor() {
        public void process(Exchange exchange) throws Exception {
            LOG.info("Handled " + exchange.getExchangeId());
        }
    }).to("log:test");
}
```

- Add a Processor in the Spring DSL
 - register Processor as a Spring bean
 - process/> to refer to Spring bean

Exercise: Getting started

- Create a CamelContext in Java code
 - create a route that writes a message to the log every 5 seconds
 - add a Processor to add a timestamp to the message body
- Create a CamelContext in Spring XML
 - copy file from one directory to another
 - add a Processor to show the file contents in the log





Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



Enterprise Integration Patterns

- Look at some of the EIPs available in Camel
 - Goals and use cases
 - Implementation in the Java DSL
 - Implementation in the Spring XML DSL
- Exercise



EIP: Pipeline

- How can you build complex processing from independent services?
 - use explicit <pipeline/> / pipeline()
 - but also happens by default in a route
- Example: order processing requires validation, stock checking and backordering

EIP: Pipeline

Example:

```
from("file:orders")
                                   from("file:orders")
  .to("direct:validation")
                                     .pipeline()
                                       .to("direct:validation")
  .to("direct:stock")
                                       .to("direct:stock")
  .to("direct:backorder");
                                       .to("direct:backorder");
<route>
                                     <route>
  <from uri="file:orders"/>
                                       <from uri="file:orders"/>
  <to uri="direct:validation"/>
                                       <pipeline>
  <to uri="direct:stock"/>
                                         <to uri="direct:validation"/>
  <to uri="direct:backorder"/>
                                         <to uri="direct:stock"/>
</route>
                                         <to uri="direct:backorder"/>
                                       </pipeline>
                                     </route>
```

EIP: Message Filter

- How can you avoid receiving uninteresting messages?
- Example: only copy XML files from dir/a to dir/b

Sidetrack: Languages

- EIPs often require an expression/predicate, which can be expressed in several Languages
 - header, constant,
 - simple
 - xpath, xquery
 - OGNL
 - JSP EL
 - Scripting languages like Ruby, Groovy, ...

EIP: Content-based router

- How to route a message to the correct recipient based on the message content?
- Example: send orders to specific order folder per customer language

```
from("file:orders")
   .choice()
   .when(xpath("/order/customer/@language = 'gd'"))
        .to("file:orders/gd")
   .when(xpath("/order/customer/@language = 'cy'"))
        .to("file:orders/cy");
```

EIP: Content-based Router

 Example: send orders to specific order folder per customer language

EIP: Multicast

- How to route the same message to multiple recipients?
- Example: order information is sent to accounting as well as fulfillment departement

EIP: Multicast

Example:

EIP: Wire Tap

- How to inspect message that travel from endpoint to the other?
- Example: we want to keep a copy of every order

```
from("file:orders")
   .wireTap("file:audit/orders")
   .to("direct:order");

<route>
   <from uri="file:orders"/>
   <wireTap uri="file:audit/orders"/>
   <to uri="direct:order"/>
</route>
```

EIP: Recipient List

- How to route a message to a list of dynamically specified recipients?
- This EIP can also be used to build a Dynamic Router

EIP: Recipient List

 Example: send orders to specific order folder per customer language



EIP: Splitter

- How can we process a message that contains multiple elements?
- Example: a customer sends a single file containing multiple orders

EIP: Aggregator

- How to combine several, related messages into a single message?
- Example: our customer now wants a single order confirmation file for the order file she sent

EIP: Aggregator

- AggregationStrategy required to combine the exchanges
 - also used for multicast and recipient list

```
public class MyAggregationStrategy implements AggregationStrategy {
   public Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
      // oldExchange contains the current aggregation result
      // newExchange is the new exchange to be added to the result
      // this would return the last Exchange as the final result
      return newExchange;
   }
}
```

Enterprise integration patterns

- ...and the list goes on
 - delay
 - enricher
 - idempotent consumer
 - loadbalance
 - loop
 - threads
 - throttle
 - •



Exercise

- Use the EIPs to implement
 - read orders files from directory
 - split the orders file in separate orders
 - make sure that we also have a copy of the original file in an audit folder
 - drop any orders with a total quantity of 0
 - organize orders in separate folders by customer country

Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



Components in action

- Introduction
- Using components
- A few examples

Introduction

DataSet	GTask	FTPS	Bean Validation	RNC
Direct	GMail	GAuth	SNMP	RNG
<u>Esper</u>	HDFS	iBATIS	UDP	<u>Nagios</u>
Event	<u>Hibernate</u>	GHttp	Spring-	RSS
<u>Exec</u>	HL7	ActiveMQ	Integration	<u>Scalate</u>
Browse	HTTP	Journal	SQL	MSV
Cache	IMap	AMQP	StringTemplate	XMPP
FTP	IRC	Stream	TCP	Mock
<u>Cometd</u>	JCR	Atom	SERVLET	XQuery
CXF	<u>Crypto</u>	JBI	Test	MINA
CXFRS	JDBC	Bean	ActiveMQ	XSLT
GLogin	File	Jetty	Timer	Netty
JavaSpace	<u>Flatpack</u>	JMS	Validation	NMR
POP	<u>Freemarker</u>	SMPP	JT/400	SEDA
Printer	SMTP	JPA	Velocity	<u>Ref</u>
Quartz	<u>Restlet</u>	Lucene	VM	LDAP
<u>Smooks</u>	RMI	Mail	<u>Quickfix</u>	Properties
		SFTP		Log

Using components

- Add required JARs to the classpath
- Add component to CamelContext
 - Auto-discovery
 - works for most of the components
 - uses files in component's JAR META-INF folder
 - Manual
 - to have multiple instances of the same component
 - to configure the component before using it e.g. inject a JMS ConnectionFactory

Using components

- Example register amq: component
 - Register in Java code

```
JmsComponent component = new JmsComponent();
component.setConnectionFactory(
    new ActiveMQConnectionFactory("tcp://localhost:61616"));
context.addComponent("amq", component);
```

Register in Spring XML



A few examples

- File and FTP
- Jetty
- JMS and ActiveMQ
- Mail
- Quartz
- XSLT



- Reading and writing files
 - on the local file system
 - on an FTP(S) server
- Add to your project
 - file: component is part of camel-core.jar
 - ftp: component is in camel-ftp.jar
 - both use auto-discovery

- File URI format
 - file:directoryName or file://directoryName
 - e.g. file:/Users/gert/incoming
- FTP URI format
 - ftp://[user@]host[:port]/directory
 - e.g. ftp://gert@www/public_html
- Both of them have similar options

- Selecting files
 - recursive
 - include
 - exclude
 - filter refers to GenericFileFilter
 - sorter refers to Comparator<GenericFile>

```
from("file:local?recursive=true&include=.*html")
   .to("ftp://www@www.anova.be/public_html");
```



- Poller configuration
 - delete
 - move
 - moveFailed
 - noop
 - preMove

from("file:orders?move=done&moveFailed=error")



- Locking and concurrency
 - readLock: markerFile, changed, fileLock, rename, none
 - idempotent
 - idempotentRepository
 - inProgressRepository

- Writing files
 - File name
 - use message id
 - use Exchange.FILE_NAME header
 - specify on endpoint

```
from("file:orders")
    .setHeader(Exchange.FILE_NAME)
    .simple("${file:name.noext}-${date:now:hhmmss}.${file:ext}")
    .to("file:handled");

from("file:orders")
    .to("file:handled?filename=${file:name}.done");
```

- Other producer options
 - fileExist: Override, Append, Fail, Ignore
 - tempPrefix
 - tempFileName

JMS and ActiveMQ

- Two JMS components
 - org.apache.camel:camel-jms
 - org.apache.activemq:activemq-camel
- URI format
 - jms:[temp:|queue:|topic:]destinationName[? options]

```
from("jms:incoming.orders")
    .to("jms:topic:events")
    .to("jms:queue:orders");
```

JMS and ActiveMQ

- Message mapping
 - String, Node → TextMessage → String
 - Map → MapMessage → Map
 - Serializable → ObjectMessage → Object



JMS and ActiveMQ

- Some of the available options
 - transactions and acknowledgement
 - JMS priority, time to live and delivery mode
 - durable subscriptions

Mail

- camel-mail component supports
 - SMTP(S) for sending mails
 - IMAP(S)/POP(S) for receiving mails
- URI format
 - imap://[username@]host[:port][?options]
 - pop3://[username@]host[:port][?options]
 - smtp://[username@]host[:port][?options]

68/88

Mail

- Options for receiving mails
 - username/password
 - delete or mark as read
 - folderName
 - unseen=false to read all messages



Mail

- Options for sending mails
 - from, to, cc, bcc can be specified on URI or set on the message as headers

```
from("file:orders")
  .process(new Processor() {

   public void process(Exchange exchange) throws Exception {
      Map<String, Object> headers = exchange.getIn().getHeaders();
      headers.put("To", "gert.vanthienen@gmail.com");
      headers.put("Subject", "Order confirmation");
   }
}).to("smtp:relay.skynet.be?from=gert.vanthienen@skynet.be");
```

Quartz and Timer

- Two ways of scheduling tasks
 - timer: uses a simple Java Timer
 - simple
 - part of camel-core
 - quartz: uses Quartz
 - more options, e.g. cron-like expressions

XSLT

- Process messages with an XSLT template
 - template is used to convert body
 - access to headers by using <xsl:param/>s
- URI
 - xslt:template[?options]
 - template can be on the classpath, file:// or http:// (Spring resource URI)

72/88

XSLT

Example for using header as a parameter

```
from("direct:xsl")
    .setHeader("timestamp").simple("${date:now:yyyyMMdd.hhss}")
    .to("xslt:transform.xsl")
   .to("log:result");
<xsl:stylesheet version="1.0" xmlns:xsl="...">
  <xsl:param name="timestamp"/>
  <xsl:template match="/">
    <message>
      <body>Message at <xsl:value-of select="$timestamp"/></body>
    </message>
  </xsl:template>
</xsl:stylesheet>
```

Exercise

- Build a set of camel routes to
 - Generate a message every 30 seconds
 - Write the message to a file
 - Read the file and it to the JMS queue
 - Read the message from the JMS queue and convert it to a nice HTML mail

Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



Beans in action

- Adding beans to your route
- Methods
- Routing with beans
- Exercise

- To access a bean in Camel
 - create the bean yourself in a Processor
 - create the bean in Spring
 - use @Autowired to automatically inject it
 - inject it into the router yourself in Spring
 - use the bean/beanRef DSL

- In the Java DSL
 - use bean() with class/object
 - use beanRef() with bean name

```
OrderService orderService = new OrderService();
from("direct:order")
   .bean(OrderService.class)
   .bean(orderService)
   .beanRef("orderService");
```

- In the Spring DSL
 - beanType for bean class
 - ref for bean name

```
<route>
  <from uri="direct:order"/>
  <bean ref="orderService"/>
  <bean beanType="be.anova.course.camel.OrderService"/>
</route>

<bean id="orderService" class="be.anova.course.camel.OrderService"/>
```

- Bean name is looked up in Registry
 - SimpleRegistry: for testing or GAE
 - JndiRegistry: uses JNDI (e.g. in appserver)
 - ApplicationContextRegistry: uses Spring ApplicationContext beans
 - OsgiServiceRegistry: leverages OSGi Service Registry, using the Spring DM bean name

- Select method to invoke
 - CamelBeanMethodName
 - explicit method name on bean/beanRef()
 - @Handler annotation or other Camel annotations
 - method parameters
- AmbiguousMethodCallException if unable to find a single best match

- Automatic binding occurs for
 - Exchange
 - Message
 - CamelContext
 - TypeConverter
 - Registry
 - Exception

```
public void handle(Exchange exchange, Message message, Exception exception){
   // handle the exchange/message directly
}
```

- Binding using annotations
 - @Body, @Header, @Headers
 - @OutHeader
 - @Property, @Properties

- Binding using annotations
 - Annotations for expression languages
 - @Bean, @BeanShell, @EL, @Groovy, @JavaScript, @MVEL, @OGNL, @PHP, @Python, @Ruby, @Simple, @XPath and @XQuery

Routing with beans

- Only when using Spring
 - add bean to Spring XML file
 - add @Consume annotation to method
 - Camel bean post-processing will create route

Exercise

- Wire the OrderService into your route
 - 3 methods
 - enrich() use Exchange as parameter and add timestamp header to the exchange
 - process() uses annotations to get parameter and message body
 - log() method uses @Consume to read from the direct:log endpoint

Planning

- Introduction
- Getting started with Camel
- EIPs in action
- Components in action
- Beans in action



What else is there?

- There's a lot more in Camel
 - Interceptors and Tracer
 - OnCompletion
 - Error handling
 - JMX management
 - Transaction support
 - Testing support
 - Async API and threading

•

