Professional Expertise Distilled

Mastering Apache Camel

An advanced guide to Enterprise Integration using Apache Camel

Jean-Baptiste Onofré

[PACKT] PUBLISHING enterprise professional expertise distilled

Mastering Apache Camel

An advanced guide to Enterprise Integration using Apache Camel

Jean-Baptiste Onofré

[PACKT] PUBLISHING enterprise professional expertise distilled

BIRMINGHAM - MUMBAI

Mastering Apache Camel

First published: June 2015

Production reference: 1250615

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78217-315-1

www.packtpub.com

Credits

Author
Jean-Baptiste Onofré

Reviewers
Volker Kueffel
Carsten Ringe
Phil Wilkins

Commissioning Editor
Amarabha Banerjee

Acquisition Editor
Meeta Rajani

Content Development Editor
Anand Singh

Technical Editors
Namrata Patil
Deepti Tuscano

Copy Editors
Merilyn Pereira
Laxmi Subramanian

Project Coordinator
Vijay Kushlani

Proofreader
Safis Editing

Indexer
Rekha Nair

Production Coordinator
Melwyn D'sa

Cover Work
Melwyn D'sa

About the Author

Jean-Baptiste Onofré is a member of the Apache Software Foundation, and he has been involved in Apache projects for about 10 years. He's the PMC chair of Apache Karaf and its subprojects, including Cellar, Cave, and EIK.

He's also a PMC member of Apache ACE, Apache ServiceMix, and Apache Syncope, and he is a committer for Apache ActiveMQ, Apache Archiva, Apache Aries, Apache Camel, and Apache jClouds.

He's currently working at Talend (`http://www.talend.com`) as a software architect and is a member of the Talend Apache team.

He has provided articles on Java technologies for GNU/Linux magazine France and has worked as an author and a reviewer on different books, such as *Learning Karaf Cellar* and *Apache Karaf Cookbook,* both by Packt Publishing.

He has also given talks on Apache projects, such as Karaf and Camel, at different conferences, especially ApacheCon NA and Europe, CamelOne, and so on.

I would like to thank the whole Camel and Karaf team, especially Guillaume Nodet, Achim Nierbeck, Jamie Goodyear, Ioannis Canellos, Claus Ibsen, and all the others. We are a great team, and you do a great job.

I would also like to thank my wife, Lucile, who accepted that I spent some nights on this book.

About the Reviewers

Volker Kueffel has been a software engineer and architect for almost two decades and has been developing software since he was a teenager. A physicist by trade, he has worked on large-scale data systems in various verticals of the software industry, spanning from online travel, mobile, and enterprise applications to online advertising. He introduced Apache Camel into one of his projects where it has successfully served as a major system component for several years. Volker is a native of Germany and currently lives with his family in San Francisco, California.

Carsten Ringe is a software developer by heart and has been working in different industries, from defense to agriculture and logistics, in the last 10 years. Over the last couple of years, he has spent his time with Apache Camel building a scalable integration platform for a large logistics enterprise.

Phil Wilkins has spent over 25 years in the software industry, working for both multinationals and software startups. He started out as a developer and has worked his way up through technical and development leadership roles, primarily in Java-based environments. He now works as an enterprise technical architect within the IT group for a global optical healthcare manufacturer and retailer using Oracle Middleware, Cloud and RedHat JBoss technologies.

Outside of his work commitments, he has contributed his technical capabilities to supporting others in a wide range of activities from the development of community websites, to providing input and support to people authoring books and developing software ideas and businesses, including reviewing a range of technical books for Packt and other publishers. He is also a blogger and a participant in the Oracle middleware community.

When not immersed in work and technology, he spends his time pursuing his passion for music and with his wife and two boys.

I'd like to take this opportunity to thank my wife, Catherine, and our two sons, Christopher and Aaron, for their tolerance and for the innumerable hours that I've spent in front of a computer, contributing to activities for both, my employer and many other IT-related activities that I've supported over the years.

---

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Instant updates on new Packt books

Get notified! Find out when new books are published by following `@PacktEnterprise` on Twitter or the *Packt Enterprise* Facebook page.

Table of Contents

Preface

Apache Camel has slowly emerged as the main framework for integration. It provides a very flexible and efficient way to integrate applications and systems all together.

Camel provides a complete set of features, based on simple but powerful concepts, allowing you to easily implement very rich integration logic.

Using this book, you will have a detailed understanding, with how to steps to implement integration logics.

What this book covers

Chapter 1, *Key Features*, introduces what Camel is and the provided key features.

Chapter 2, *Core Concepts*, introduces the basis of all the functionalities provided by Camel.

Chapter 3, *Routing and Processors*, introduces Camel routing and the usage of processors.

Chapter 4, *Beans*, explains how to use beans in Camel routes and the different registries in which the beans live.

Chapter 5, *Enterprise Integration Patterns*, introduces one of the most interesting features of Camel—the ready-to-use patterns, which serve as an answer to classic integration problems.

Chapter 6, *Components and Endpoints*, introduces Camel components and endpoints, both how to use them and implement your own.

Chapter 7, *Error Handling*, introduces how to deal with errors in Camel routes.

Chapter 8, *Testing*, introduces how to implement both unit tests and integration tests on your Camel routes.

What you need for this book

For this book, the software required will be as follows:

- Operating systems (any system supporting Java):
    - Windows 7 or superior
    - Unix (Linux)
- Java DK 1.7
- Apache Karaf 3.0.3

Who this book is for

This book is for developers who want to implement integration logic using Apache Camel. They will get details about Camel, from basic usage, up to the custom development of their own components.

Thanks to the first few chapters, even beginners unfamiliar with Camel will receive a comprehensive look into Camel before jumping into the details.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: " A message is described in the `org.apache.camel.Message` interface."

A block of code is set as follows:

```
public class MyProcessor implements Processor {

  public void process(Exchange exchange) {
System.out.println("Hello " +
exchange.getIn().getBody(String.class));
  }

}
```

Any command-line input or output is written as follows:

```
$ mvn clean install
```

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Key Features

After a quick introduction about what Apache Camel is, this chapter will introduce the key features provided by Camel. It provides just an overview of these features; the details will come in dedicated chapters.

In an enterprise, you see a lot of different software and systems in the IT ecosystem. In order to consolidate the data and sync the systems, the enterprise would want to implement communication and integration of these systems. This communication or integration is not so easy, as we have to deal with the specifications on each system the protocol and the message's data format are different most of the time, so we have to transform and adapt to each system.

Using point-to-point communication is one option. However, the problem with this approach is that we tighten the integration of a couple of systems. Changing to other systems or protocols requires refactoring of the implementation. Moreover, dealing with multiple systems is not so easy with point-to-point.

So, instead of point-to-point, we use mediation. Mediation reduces complexity and provides a more flexible approach by adding and using a tier between the systems (man in the middle). The purpose is to facilitate the information flow and integration of the systems.

Apache Camel is a mediation framework.

What is Apache Camel?

Apache Camel originated in Apache ServiceMix. Apache ServiceMix 3 was powered by the Spring framework and implemented in the JBI specification. The **Java Business Integration** (**JBI**) specification proposed a Plug and Play approach for integration problems. JBI was based on WebService concepts and standards. For instance, it directly reuses the **Message Exchange Patterns** (**MEP**) concept that comes from **WebService Description Language** (**WSDL**).

Camel reuses some of these concepts, for instance, you will see that we have the concept of MEP in Camel.

However, JBI suffered mostly from two issues:

- In JBI, all messages between endpoints are transported in the **Normalized Messages Router** (**NMR**).

  In the NMR, a message has a standard XML format. As all messages in the NMR have the same format, it's easy to audit messages and the format is predictable.

  However, the JBI XML format has an important drawback for performances: it needs to marshall and unmarshall the messages. Some protocols (such as REST or RMI) are not easy to describe in XML.

  For instance, REST can work in stream mode. It doesn't make sense to marshall streams in XML.

  Camel is payload-agnostic. This means that you can transport any kind of messages with Camel (not necessary XML formatted).

- JBI describes a packaging. We distinguish the binding components (responsible for the interaction with the system outside of the NMR and the handling of the messages in the NMR), and the service engines (responsible for transforming the messages inside the NMR).

  However, it's not possible to directly deploy the endpoints based on these components. JBI requires a service unit (a ZIP file) per endpoint, and for each package in a service assembly (another ZIP file). JBI also splits the description of the endpoint from its configuration.

  It does not result in a very flexible packaging: with definitions and configurations scattered in different files, not easy to maintain. In Camel, the configuration and definition of the endpoints are gathered in a simple URI. It's easier to read.

  Moreover, Camel doesn't force any packaging; the same definition can be packaged in a simple XML file, OSGi bundle, and regular JAR file.

In addition to JBI, another foundation of Camel is the book *Enterprise Integration Patterns by Gregor Hohpe and Bobby Woolf*.

This book describes design patterns answering classical problems while dealing with enterprise application integration and message oriented middleware.

The book describes the problems and the patterns to solve them. Camel strives to implement the patterns described in the book to make them easy to use and let the developer concentrate on the task at hand.

This is what Camel is: an open source framework that allows you to integrate systems and that comes with a lot of connectors and **Enterprise Integration Patterns** (**EIP**) components out of the box. And if that is not enough, one can extend and implement custom components.

Components and bean support

Apache Camel ships with a wide variety of components out of the box; currently, there are more than 100 components available.

We can see:

- The connectivity components that allow exposure of endpoints for external systems or communicate with external systems. For instance, the FTP, HTTP, JMX, WebServices, JMS, and a lot more components are connectivity components. Creating an endpoint and the associated configuration for these components is easy, by directly using a URI.
- The internal components applying rules to the messages internally to Camel. These kinds of components apply validation or transformation rules to the inflight message. For instance, validation or XSLT are internal components.

Thanks to this, Camel brings a very powerful connectivity and mediation framework.

Moreover, it's pretty easy to create new custom components, allowing you to extend Camel if the default components set doesn't match your requirements.

It's also very easy to implement complex integration logic by creating your own processors and reusing your beans. Camel supports beans frameworks (IoC), such as Spring or Blueprint.

Predicates and expressions

As we will see later, most of the EIP need a rule definition to apply a routing logic to a message. The rule is described using an expression.

It means that we have to define expressions or predicates in the Enterprise Integration Patterns. An expression returns any kind of value, whereas a predicate returns true or false only.

Camel supports a lot of different languages to declare expressions or predicates. It doesn't force you to use one, it allows you to use the most appropriate one.

For instance, Camel supports xpath, mvel, ognl, python, ruby, PHP, JavaScript, SpEL (Spring Expression Language), Groovy, and so on as expression languages. It also provides native Camel prebuilt functions and languages that are easy to use such as header, constant, or simple languages.

Data format and type conversion

Camel is payload-agnostic. This means that it can support any kind of message. Depending on the endpoints, it could be required to convert from one format to another. That's why Camel supports different data formats, in a pluggable way. This means that Camel can marshall or unmarshall a message in a given format. For instance, in addition to the standard JVM serialization, Camel natively supports Avro, JSON, protobuf, JAXB, XmlBeans, XStream, JiBX, SOAP, and so on.

Depending on the endpoints and your need, you can explicitly define the data format during the processing of the message. On the other hand, Camel knows the expected format and type of endpoints. Thanks to this, Camel looks for a type converter, allowing to implicitly transform a message from one format to another.

You can also explicitly define the type converter of your choice at some points during the processing of the message. Camel provides a set of ready-to-use type converters, but, as Camel supports a pluggable model, you can extend it by providing your own type converters. It's a simple POJO to implement.

Easy configuration and URI

Camel uses a different approach based on URI. The endpoint itself and its configuration are on the URI.

The URI is human readable and provides the details of the endpoint, which is the endpoint component and the endpoint configuration.

As this URI is part of the complete configuration (which defines what we name a route, as we will see later), it's possible to have a complete overview of the integration logic and connectivity in a row. We will cover this in detail in *Chapter 2*, *Core Concepts*.

Lightweight and different deployment topologies

Camel itself is very light. The Camel core is only around 2 MB, and contains everything required to run Camel. As it's based on a pluggable architecture, all Camel components are provided as external modules, allowing you to install only what you need, without installing superfluous and needlessly heavy modules.

As we saw, Camel is based on simple POJO, which means that the Camel core doesn't depend on other frameworks: it's an atomic framework and is ready to use. All other modules (components, DSL, and so on) are built on top of this Camel core.

Moreover, Camel is not tied to one container for deployment. Camel supports a wide range of containers to run. They are as follows:

- A J2EE application server such as WebSphere, WebLogic, JBoss, and so on
- A Web container such as Apache Tomcat
- An OSGi container such as Apache Karaf
- A standalone application using frameworks such as Spring

Camel gives a lot of flexibility, allowing you to embed it into your application or to use an enterprise-ready container.

Quick prototyping and testing support

In any integration project, it's typical that we have some part of the integration logic not yet available. For instance:

- The application to integrate with has not yet been purchased or not yet ready
- The remote system to integrate with has a heavy cost, not acceptable during the development phase
- Multiple teams work in parallel, so we may have some kinds of deadlocks between the teams

As a complete integration framework, Camel provides a very easy way to prototype part of the integration logic. Even if you don't have the actual system to integrate, you can simulate this system (mock), as it allows you to implement your integration logic without waiting for dependencies. The mocking support is directly part of the Camel core and doesn't require any additional dependency.