Contents

[Chapter 01 : Meeting Camel 2](#_Toc160993677)

[1.1 Introducing Camel 2](#_Toc160993678)

[1.1.1 What is camel? 2](#_Toc160993679)

[1.1.2 Why Use Camel? 3](#_Toc160993680)

[1.2 Getting Started 4](#_Toc160993681)

[1.2.1 Getting Camel 4](#_Toc160993682)

[1.2.2 Your First Camel Ride 4](#_Toc160993683)

[1.3 Camel’s Message Model 5](#_Toc160993684)

[1.3.1 Message 5](#_Toc160993685)

[1.3.2 Exchange 6](#_Toc160993686)

[1.4 Camel’s Architecture 7](#_Toc160993687)

[1.4.1 Architecture from 10,000 Feet 7](#_Toc160993688)

[1.4.2 Camel Concepts 8](#_Toc160993689)

[1.5 Your First Camel ride, revisited. 10](#_Toc160993690)

[1.6 Summary 11](#_Toc160993691)

[Chapter 02 : Routing with Camel 11](#_Toc160993692)

[2.1 Introducing Rider Auto Parts 12](#_Toc160993693)

[2.2 Understanding Endpoints 13](#_Toc160993694)

[2.2.1 Working with Files over FTP 13](#_Toc160993695)

[2.2.2 Sending to a JMS Queue 14](#_Toc160993696)

[Chapter 07 - Understanding Components: **Completed** 15](#_Toc160993697)

[7.1 Overview of the Camel Components 15](#_Toc160993698)

[7.1.1 Manually Adding Components 15](#_Toc160993699)

[7.1.2 Autodiscovering Components (Skipped) 16](#_Toc160993700)

[7.2 Working with Files (File and FTP Components) 16](#_Toc160993701)

[7.2.1 Reading and Writing a file with File Component 16](#_Toc160993702)

[7.2.2 Accessing Remote Files with the FTP Component 17](#_Toc160993703)

[7.3 Asynchronous Messaging (JMS Component) 17](#_Toc160993704)

[7.3.1 Sending and Receiving Messages 18](#_Toc160993705)

[7.3.2 Request-Reply Messaging 18](#_Toc160993706)

[7.3.3 Message Mappings 19](#_Toc160993707)

[Body Mapping 20](#_Toc160993708)

[Header Mapping 21](#_Toc160993709)

[7.4 Web Services (CXF Component) : Skipped 21](#_Toc160993710)

[7.5 Networking (MINA Component): Skipped 22](#_Toc160993711)

[7.6 Working With Database (JDBC and JPA Components) 22](#_Toc160993712)

[7.7 In-Memory Messaging (Direct, SEDA, and VM Components) 22](#_Toc160993713)

[7.7.1 Synchronous Messaging with the Direct Component 22](#_Toc160993714)

[7.7.2 Asynchronous Messaging with SEDA and VM 23](#_Toc160993715)

[7.8 Automating Tasks (Timer and Quartz Component): Skipped 24](#_Toc160993716)

[7.9 Summary and Best Practices 24](#_Toc160993717)

# Chapter 01 : Meeting Camel

1. A white background with red text

   Description automatically generated
2. Building **complex systems** from scratch is very **costly endeavor**.   
   **Effective and less risky alternative** is to use **existing proven components** like jigsaw puzzle.
3. Jigsaw puzzle pieces are made to plug into each other but systems are not like this.  
   So integration frameworks fill the gap.
4. A good integration framework provides simple and manageable integration for the systems to be integrated.  
   **Apache Camel is such Integration Framework.**

## 1.1 Introducing Camel

1. Camel is integration framework making integration projects productive and simple.
2. Started in 2007, Open-Source under liberal Apache 2 license with Strong Community.
3. It was named camel.
   1. Because the name is short and easy to remember.
   2. **Rumor**: One of the founders once smoked Camel Cigarettes.
   3. <https://camel.apache.org/manual/faq/why-the-name-camel.html>

### 1.1.1 What is camel?

1. At the core of Camel Framework is **Routing Engine or more precisely Routing Engine Builder**.
2. A Routing Engine allows us to define our own **Routing Rules**.
   1. From which sources to accept messages.
   2. How to process messages.
   3. To which destination, the processed msgs to be sent.
3. Camel uses integration language which allow us to define rules.
4. **No Canonical format**. Any systems with any kind of data can be integrated.
5. Because of High level of abstraction, same api regardless of the **protocol or data types** the systems are using.  
   Components in Camel provide **API Implementations** targeting different protocols and data types.  
   Camel supports >**80 Protocols and Data Types**.
6. Its **extensible and modular architecture** allows us to implement and plug in support for our own protocols.
7. Camel is not ESB (Enterprise Service Bus).

### 1.1.2 Why Use Camel?

1. A close-up of a person's face

   Description automatically generated

#### Routing and Mediation Engine

1. Routing Engine will move a message based on routing configuration.
2. In Camel, route configuration is done with EIP (Enterprise Integration Patterns) and DSL (Domain Specific Language).

#### Enterprise Integration Patterns

1. Although integration problems are diverse, but still Gregor and Bobby (EIP Book Authors) noticed that many problems and their solutions are quite similar.  
   They catalog in their book (EIP: Enterprise Integration Pattern).
2. EIP not only provides proven solution but also helps define and communicate problem itself.

It is like using sign language or spoken language.

1. Camel is highly based on EIP.
2. There is one-to-one relationship between the patterns in EIP and The Camel DSL.

#### Domain Specific Language

1. Camel offers multiple DSLs in regular programming languages such as Java, Scala, Groovy and it also allows routing rules in XML.
2. The purpose of DSL is to allow developers to focus on the integration problems rather than on the tool - Programming Language.

A white background with black text

Description automatically generated

As these are real programming language, so we can use respective tools such as Code Completion and Compile Error Handling.

#### Extensive Component Library

1. Library of more than 80 components enabling camel to connect over transports, use API’s and understand Data Formats.

#### Payload Agnostic Router

1. Camel can route any kind of data.   
   So, no need to transform our payload into a canonical format to facilitate routing.

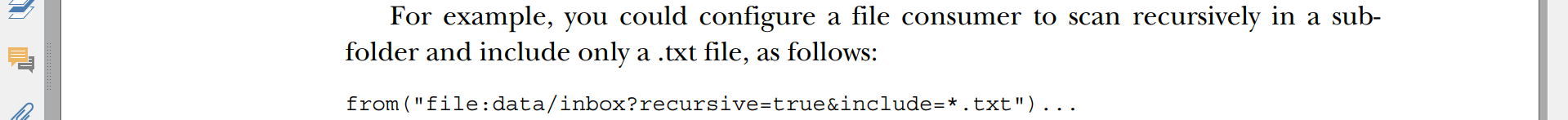
#### Modular and Pluggable Architecture

1. As Camel has a modular architecture which allows us to load any component (Camel Component, Custom Component or 3rd Party Component).

#### POJO Model

1. Beans (POJO) are first class citizen in Camel and we can extend the Camel’s functionality with our custom code using Beans.

#### Easy Configuration

1. The *Convention Over Configuration* Paradigm is applied whenever possible.
2. In order to configure endpoints directly in routes, Camel uses easy and intuitive URI format. 

#### Automatic Type Converters

1. More than 150 Type Converters.
2. No need to configure them.
3. If not supported, we can create our own.
4. The Camel Component also leverages this feature.

#### Lightweight Core

1. Camel Core lightweight which is up to 1.5 MB.
2. It has dependency only on Apache Commons Logging & Fuse-Source Commons Management which makes camel to easily embed or deploy anywhere we like such as standalone app, web app, Spring App, Java EE App, JBI Container, OSGi bundle, Java Web Start or Google App Engine.
3. Camel was designed not to be Servers or ESB but instead to be embedded in whatever platform we choose.

#### Test Kit

1. Camel provides Test Kit that we can use and the same was used to test Camel Components by Camel.

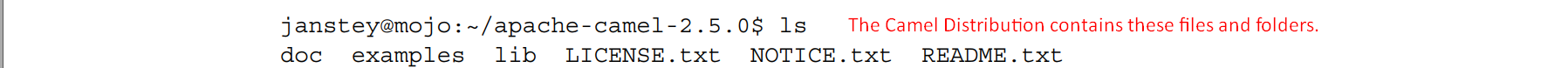
#### Vibrant Community

1. Camel has an active community in case we face any issue.  
   See appendix D.

## 1.2 Getting Started

1. We will get our hands on Camel Distribution and explain what is inside and then run an example using Apache Maven.
2. Then you will be able to run any example from this book source code.

### 1.2.1 Getting Camel

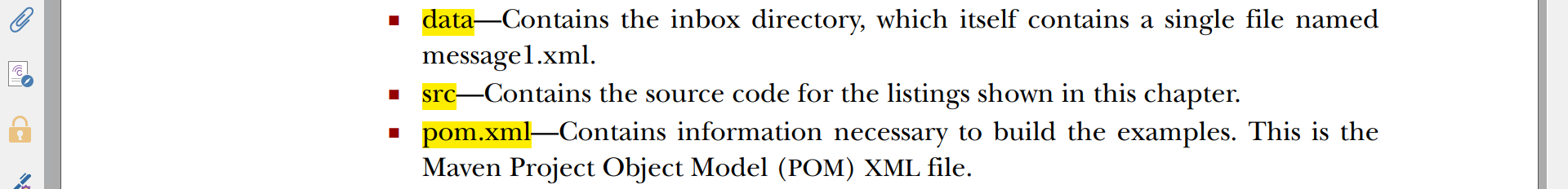
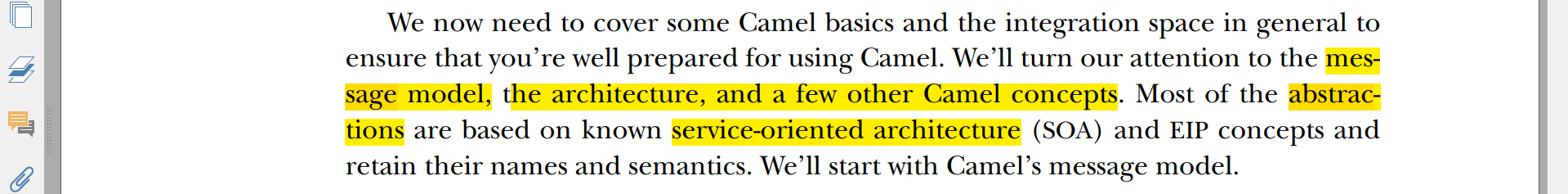
1. This site <https://camel.apache.org/download/> has all the release and we can download the latest.  
   Better use this 🡪 <https://www.apache.org/dyn/closer.cgi/camel/apache-camel/2.5.0/apache-camel-2.5.0.zip>
2. We will be using 2.5.0.   
   
3. A text on a page

   Description automatically generated

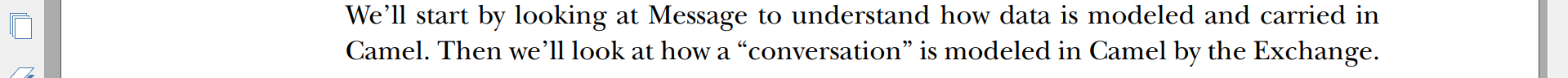
### 1.2.2 Your First Camel Ride

1. We will use Apache Maven not Camel Distribution.
2. Book Source Code : <http://manning.com/ibsen>
3. 1st example reading files (data/inbox), processing and write the result to another dir (data\outbox). A diagram of a file

   Description automatically generated  
   Jatin: I am skipping Java Source code for the same but if we write there are 34 lines of code. Need to make sure of closing resources and deal with Low Level File API. If polling (inbox/inbox) for new files, need to set up timer and keep track of already processed files.
4. A screenshot of a computer

   Description automatically generated  
   Routes are written in such a way that they flow. The above route can be read like this - consume files from “data/inbox” with **noop=true** (Telling camel to leave the files as it is) and send to “data/outbox”.  
   See, excluding the boilerplate code, we wrote file-polling route in just one line of Java Code.
5. Move to chapter1/file-copy directory of the book’s source. The listings of directory:
6.   
   Maven 2.2.1 was used during the development.
7. 

## 1.3 Camel’s Message Model

1. In camel, two abstractions for modeling msgs.
2. **org.apache.camel.Message:**
   1. The **fundamental entity** containing the data being carried and routed in Camel.
3. **org.apache.camel.Exchange:**
   1. The Camel abstraction for an exchange of message.
   2. It has **“in” message** and **“out” message** as reply.
4. 

### 1.3.1 Message

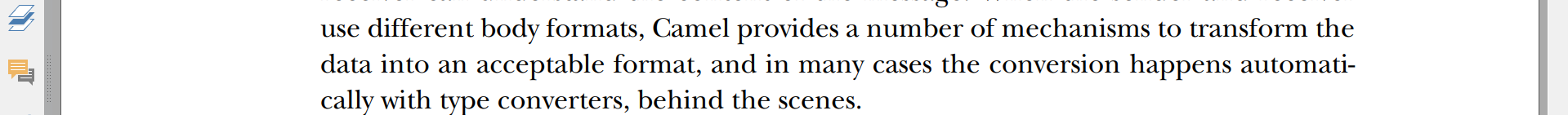
1. Systems when communicate with each other using **message channels**, they communicate using **Message Entities**.
2. Message flows from Sender to Receiver.
3. A message has   
   A white rectangular object with black text

   Description automatically generated
   1. body (Payload)
   2. Headers.
   3. Optional Attachments.
4. A Message is identified by a unique **java.lang.String** and uniqueness is guaranteed by the Message Creator.
5. As it is protocol dependent so ID format can’t be guaranteed.
6. For Protocols which don’t define a **unique message identification schema**, Camel uses its own **UID Generator**.

#### Headers and Attachments

1. **Headers** are values associated with a Message. Key-value (as Map) where key is unique and case-insensitive and value of type java.lang.Object.
   1. Sender Identifier.
   2. Hits about Encoding.
   3. Authentication Information and so on.
2. Message also has **optional attachments** for **web service and email components**.

#### Body

1. Type **java.lang.Object** which means any type of object can be stored inside Message body.
2. App Designer is responsible for the receiver to understand the msg content.
3. Not Clear:  
   

#### Fault Flag

1. Some protocols and specifications such as WSDL and JBI, distinguish between **output** and **fault** messages.  
   **Not Clear**:  
   

During Routing Messages are contained in an exchange.

### 1.3.2 Exchange

1. Exchange is message’s container during routing.
2. Exchange also provides support for various types of interactions b/w systems, also known as **MEP** (Message Exchange Pattern).
3. MEP is used to differentiate b/w one-way and request-response **messaging styles**.

A screenshot of a computer

Description automatically generatedA close-up of a text

Description automatically generated

1. **Exception**: Any Exception during routing, the exception in exception field.
2. **Properties**:

|  |  |
| --- | --- |
| Headers | Properties |
| Message Specific | Exchange Specific |
| Message related info | Global Level Info |
|  | Life Time : Exchange |
| Added by Developers | By Camel and Developers. |

1. **In-Message**: Input message and mandatory. It contains request message.
2. **Out-Message**: Optional and exists only if MEP property is to InOut then it contains reply message and used http transport for example.

Before architecture, we discussed Camel’s message model as Camel is all about routing message.

## 1.4 Camel’s Architecture

1. First high-level architecture then will go into details about specific concepts.
2. After we have read this section, you should be caught up on the integration lingo and be ready for chapter 2, where we will explore Camel’s routing capabilities.

### 1.4.1 Architecture from 10,000 Feet

1. A screenshot of a computer

   Description automatically generated
2. Routes are specified using one of the Camel’s DSLs.
3. Processors are used to transform and manipulate messages during routing and also to implement EIPs patterns using keywords in DSL.
4. Components are extension points for adding connectivity to other systems.

Let’s take a closer look at the individual concepts.

### 1.4.2 Camel Concepts

#### Camel Context

1. It is runtime for Camel and gluing all pieces together.
2. A screenshot of a computer

   Description automatically generated
3. A screenshot of a computer

   Description automatically generated A screenshot of a computer

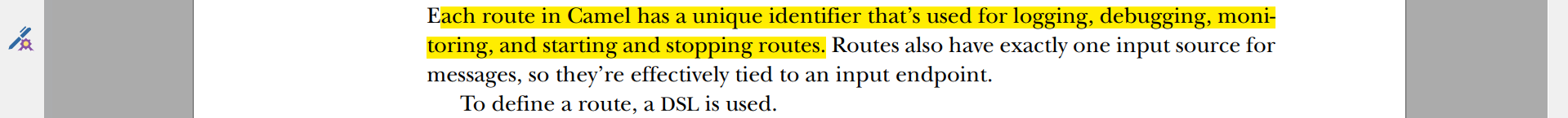
   Description automatically generated

#### Routing Engine

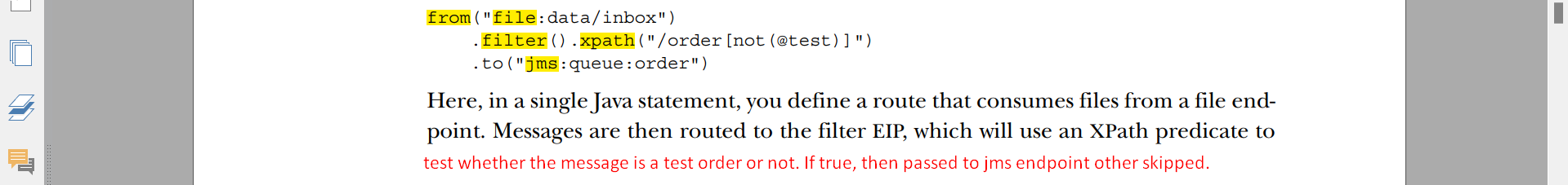
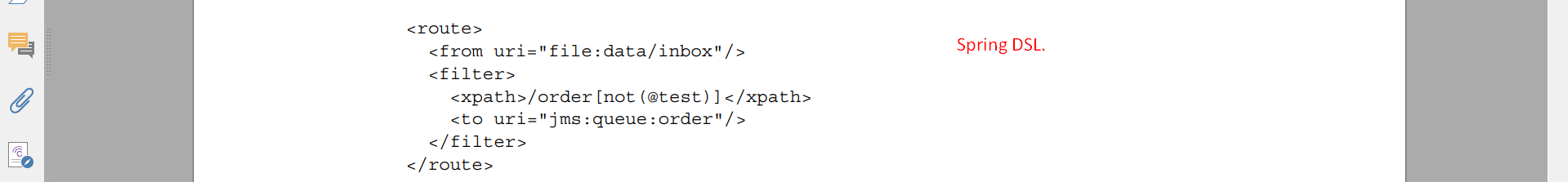
1. Camel’s Routing engine actually moves the message under the hood.
2. Not exposed to developer but there doing all the heavy lifting and making sure proper msg routing.

#### Routes

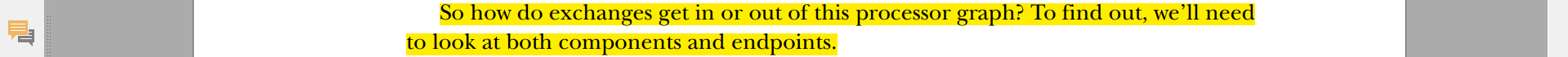
1. Routes are obviously a core abstraction for Camel.
2. The simplest way of defining a route is as a chain of processors.
3. Note Clear:  
   A close-up of a text

   Description automatically generated
4. 

#### Domain-Specific Language (DSL)

1. Camel defines DSL using which we wire endpoints and processors together to form routers.
2. In Camel, DSL means a **Fluent Java API** that contains methods named for EIP Patterns. Same route can be defined using Spring DSL.   
   So, DSL provides a nice abstraction for routing msg.  
   Under **the hood, a route is actually composed of a group of processors.**

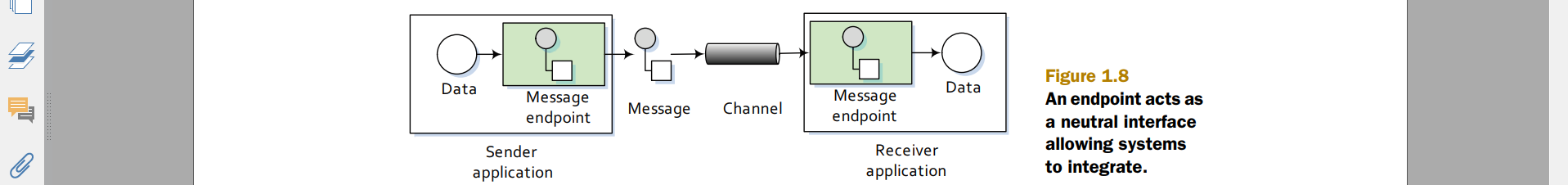
#### Processors

1. Core Camel concept that represents a node (where msg stops) and capable of using, creating or modifying incoming exchange.
2. During routing, exchange moves from one processor to another.  
   So **route** = **graph** where   
   **nodes** = processors and   
   **lines** = Output of one processor as input of another processor.
3. Many processors are implementation of EIPs and we can create custom processors.
4. 

#### Component

1. Components are the main extension points in Camel.
2. To date, more than 80 components that range in function from data transports, to DSLs, Data formats, and so on.
3. We can create our own components (Chapter 11)
4. Programming point of view, components are associated with a name that is used in URI and a component acts as a factory for endpoint.  
   **For Example**: A **FileComponent** is referred to as **file** in **URI** and creates **FileEndpoints**.  
   from(“**file**:myFileSource”).to(“**file**:myFileDestination”); Creating two endpoints one source and another destination.

#### Endpoint

1. An endpoint is the Camel Abstraction that models the end of a Channel through which a system sends or receives messages. 
2. Endpoint can be configured using URIs 🡺 file:data/inbox?delay=5000.
3. Endpoint = URI is divided into 3 parts. A screenshot of a computer

   Description automatically generated
4. **Schema**: Schema denotes which Camel component handles that type of endpoint.   
   Above **schema of file** selects **FileComponent.**
5. **FileComponent**: It then works as a factory creating **FileEndpoint** based on the remaining part of the URI -> data/inbox?delay=5000
6. **Context Path (data/index)**: It tells FileComponent that the starting folder is data/index.  
   The option ?delay=5000 indicates that the files should be polled at a 5 second intervals.
7. **Endpoint**: Endpoint acts as factory creating consumer and producer that are **capable of sending and receiving msgs to a particular endpoint**.

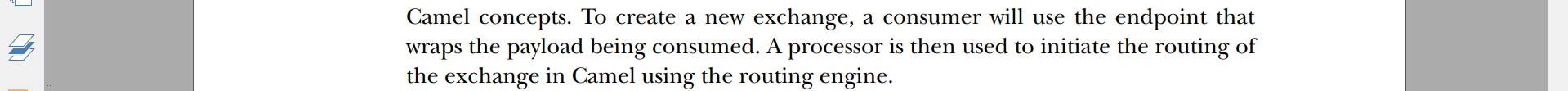
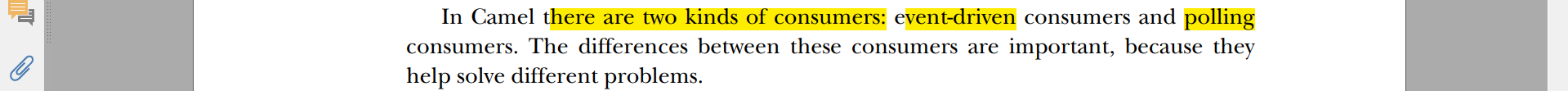
#### PRODUCER

1. A producer camel is an abstraction that refers to an entity capable of creating and sending a message to an endpoint.  
   **to**(“kafka:topicName”);
2. Whenever a msg needs to be sent, the producer will create an exchange and will populate it with the data that is compatible with that particular endpoint.
3. **For Example**:

|  |  |
| --- | --- |
| **FileProducer** | Will write message body to a file. |
| **JmsProducer** | Will write message to javax.jms.Message before sending to jms endpoint. |

This important feature of Camel hides the complexity of interacting with particular transports.  
Just route the message to an endpoint and producer does heavy lifting.

#### Consumer

1. A consumer is the service that receives messages produced by a producer.
2. 
3. 

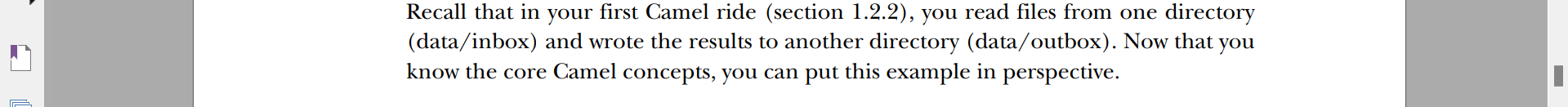
#### Event-Driven Consumer

1. Skipped

#### Polling Consumer

1. Skipped

## 1.5 Your First Camel ride, revisited.

1. 
2. A screenshot of a computer

   Description automatically generated
3. A close up of text

   Description automatically generated

## 1.6 Summary

1. Camel simplifies the integration by relying on EIPs.
2. We saw Camel DSL making Camel Code Self Documenting and keeps developers focused on what the glue code does now how it works.
3. Camel provides API that works over a large range of protocols and data formats.
4. In the rest of the book, we will explore Camel’s features and give you practical solutions we can apply in everyday integration scenarios.
5. We will also **explore what is going on under the Camel’s tough skin.**
6. Next chapter is about routing.

# Chapter 02 : Routing with Camel

1. A close up of a text

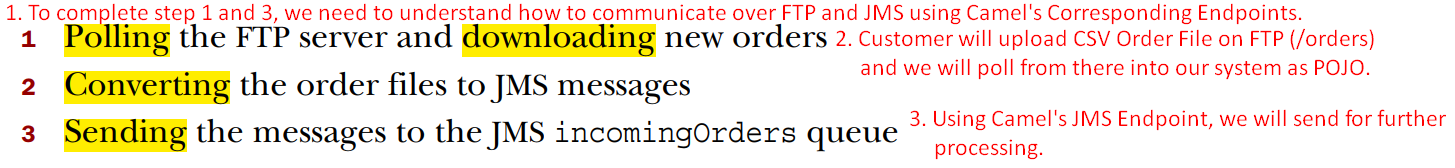
   Description automatically generated
2. When routing **Physical mail**, it may be **routed through several cities**.  
   When **routing email**, it may be routed through **several computer network systems**.  
   In all cases, the router’s function is to selectively move the message forward.
3. In the context of Enterprise Messaging System, routing is the process by which a message is taken from one input queue and based a set of conditions, send to one of several output queues, as shown in figure 2.1.  
   Consumer and producer are unaware of these conditions.  
   A diagram of a diagram

   Description automatically generated
4. **In Apache Camel, Routing** = Origin Endpoint (Consumer) + A Processing Component + Target Endpoint (Producer).
   1. Routing is step by step movement of message, which originates from an endpoint in the role of consumer.
   2. **Consumer could be receiving message from:**
      1. External Service.
      2. Polling for the message on some system.
      3. Or creating the msg itself.
   3. **Then the message flows through a processing component**: With zero or more processing component. If zero, then simple pipeline.
      1. Enterprise Integration Pattern.
      2. A Processor.
      3. An interceptor or
      4. Some other custom creation.
   4. Finally, the message is sent to the target endpoint in the role of producer.
5. In this chapter, we’ll first introduce the financial company that we will use as the running example through the book.
6. A text on a white background

   Description automatically generated

## 2.1 Introducing Rider Auto Parts

1. Our fictional motorcycle parts business, **Rider Auto Parts**, supplies to **motorcycle manufacturers**.
2. Over the years, they’ve changed the way they receive orders several times.
3. **Earlier**: CSV over FTP. A User uploads Part order on FTP Server in CSV Format from where the order (CSV) is transferred over FTP to Rider Order Frontend.
4. **New Way**: XML over HTTP by Rider Auto Parts Web Store. A user uses web store and the order as XML is transferred over HTTP as XML to Rider Order Frontend.
5. No matter what, once order comes inside Rider Order Frontend, it is converted POJO and then it is transferred to Rider Order Backend by JMS.
6. A diagram of a diagram

   Description automatically generated  
   **Figure 2.2**
7. **Problem**: Rider Auto Parts acquires Software Baggage int the form of transports (FTP, HTTP, JMS) and Data formats (CSV, HTTP).  
   **But no problem** with Integration Framework like Camel and we will implement the current requirements and new functionality using CAMEL.
8. As a first assignment, we will need to implement the FTP module in the Rider Order Frontend system.  
   Later on, we will see how backend services are implemented too.
9. Implementing the FTP module will involve the following Steps:
   1. 

## 2.2 Understanding Endpoints

1. Endpoint is the abstraction that models the endpoint of a **message channel** through which a system can send or read messages.
2. Let’s see how to configure the camel to communicate over FTP and JMS.

### 2.2.1 Working with Files over FTP

1. One of the things that make Camel easy to use is the **Endpoint URI**.
2. **URI identifies**:
   1. A component to be used (Like for each kind of transport (FTP, HTTP, Kafka etc)
   2. And how that component is configured.  
      See the following snapshot where URI is defined which will identify Camel Component and its configuration.
3. Once the URI is defined, we can either send or receive a message to the component configured by the URI.
4. To download new orders from the FTP server, we need the following.
   1. A close up of a sign

      Description automatically generated
5. See, in the following snapshot by configuring Camel with URI, we are saying to do all the above.   
   A screenshot of a computer

   Description automatically generated  
   **Figure 2.3**
6. **Scheme**:
   1. **Scheme** tells Camel which component.
   2. Camel will look for the **Scheme** “ftp” in the **Camel Registry** and it will resolve to **FileComponent.**
7. **Context Path + Options**: Based on these values, **FtpComponent** will work as factory to create **FtpEndpoint.**
8. **Context Path = rider.com/orders** : Tells **FileComponent** to log into the **FTP Server** at rider.com on the default port 21 and change the director to “orders”.
9. **Options = username=rider$password=secret** : Used **to log in to** the **FTP Server**.
10. Username and password can be specified in the context path itself without changing its earlier meaning.  
    [ftp://**rider:secret**@rider.com/orders](ftp://rider:secret@rider.com/orders)
11. **Add the following dependency as FTP Component is not part of the camel-core**.  
    
12. The above URI will work fine for consumer and producer, but we will use it to download order files CSVs from FTP.  
    So, use it in from() node in the **Camel’s DSL**.  
    
13. Recall from figure 2.2, the downloaded orders from FTP need to be sent to JMS Queue.  
    This process requires a little more setup but easy.

### 2.2.2 Sending to a JMS Queue

1. Camel provides extensive support for connecting to **JMS-Enabled Providers**. (We will cover that in detail in Chapter 7).

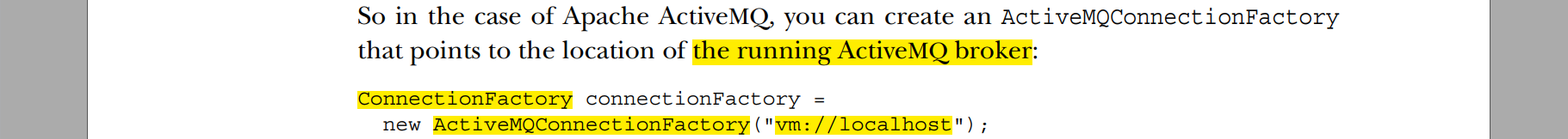
#### what is JSM

1. JMS is a Java API to create, send, receive, and read messages. (De-Facto Messaging Solution in Java Community)
2. It also mandates that messaging is asynchronous and has specific elements of reliability like guaranteed and once-and-only-once delivery.
3. Consumers and producers talk to each other through an intermediary - **JMS Destination** which can be a queue or topic.  
   A diagram of a flowchart

   Description automatically generated
4. JMS also provides **a ConnectionFactory** which a client (like Camel) can use to create a connection with a JMS Provider.
5. JMS Providers are usually called Brokers as they manage communication b/w a producer and a consumer.

#### How to configure Camel to Use A JMS Provider

1. Configure Camel JSM Component with an appropriate **ConnectionFactory**.
2. We will use ApacheMQ JMS Provider which is the most popular and the same is used by Camel team to test Camel’s JMS Components.  
   Read ActiveMQ in Action by Bruce Snyder.
3. So, we will use **ActiveMQConnectionFactory** that points to the location of the running ActiveMQ Broker.
4. A diagram of a diagram

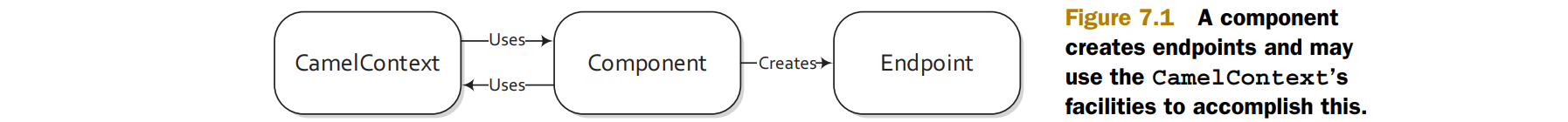
   Description automatically generated
5. 

# Chapter 07 - Understanding Components: **Completed**

1. A screenshot of a computer

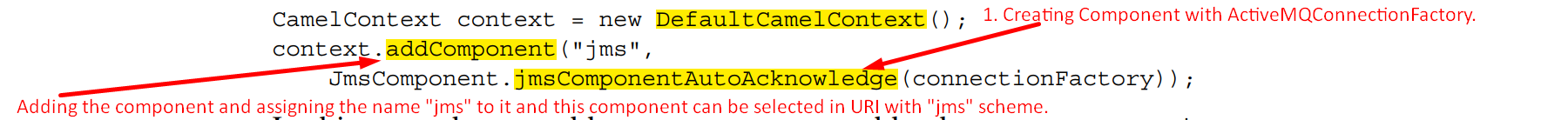
   Description automatically generated  
   We will talk about
   1. What is a Component in Camel?
   2. How are components added to Camel?

## 7.1 Overview of the Camel Components

1. Components are primary extension points in Camel.
2. As of version 2.5, Camel ships with more than 80 components and still dozens from other community sites.  
   These components allow us to bridge to many different APIs, protocols, data formats and so on.
3. If Camel routes as Highways and components to be ramps.
4. 
5. A close-up of a computer screen

   Description automatically generated
6. Main responsibility of Component is to create endpoint using CamelContext and CamelContext provides other facilities like registry, class loader, & type converters.
7. **Two ways to add components to a “Camel Runtime”**.
   1. Adding manually to CamelContext and
   2. Through Autodiscovery.

# 7.1.1 Manually Adding Components

1. In Chapter 02, we had to add a configured JMS component to the **CamelContext** to utilize a **ConnectionFactory**.
2. 

# 7.1.2 Autodiscovering Components (Skipped)

# 7.2 Working with Files (File and FTP Components)

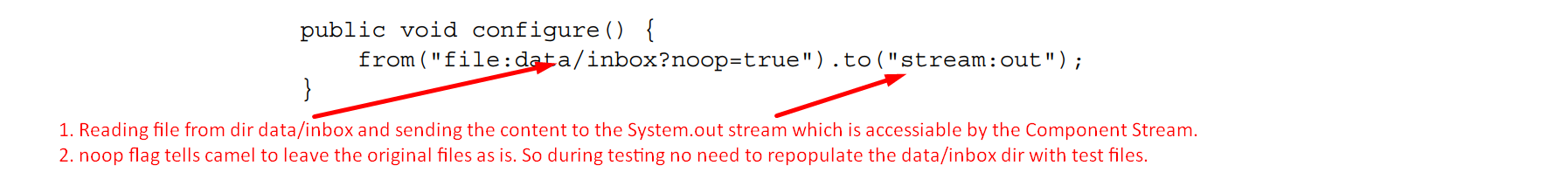
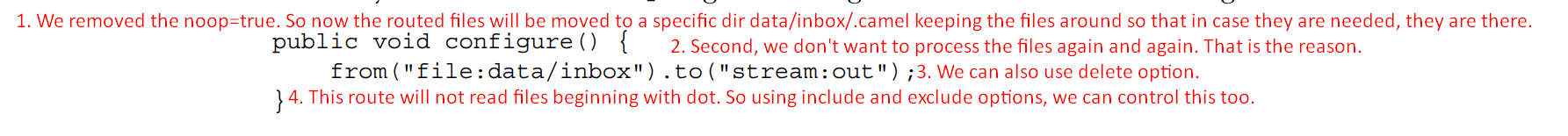
1. In any system integration, you may come across some legacy systems where we need to read a file written some other system - it could be sending a command, an order to be processed, data to be logged or anything else.
2. A close-up of a phone

   Description automatically generated
3. In this section, we will look at.
   1. How to use File Component to read a file and write to a local file system.
   2. Some advanced file processing options and
   3. How to access remote files with the FTP Component.

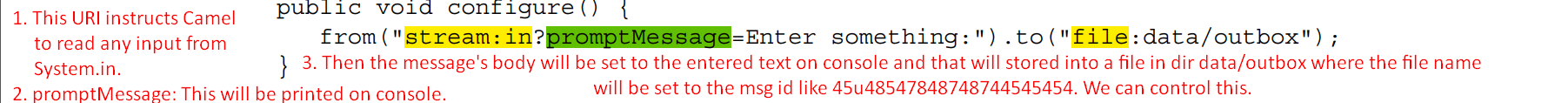
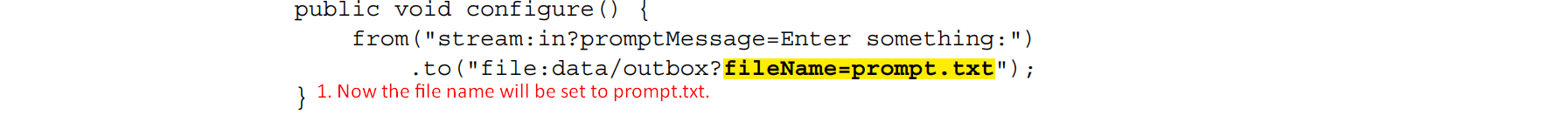
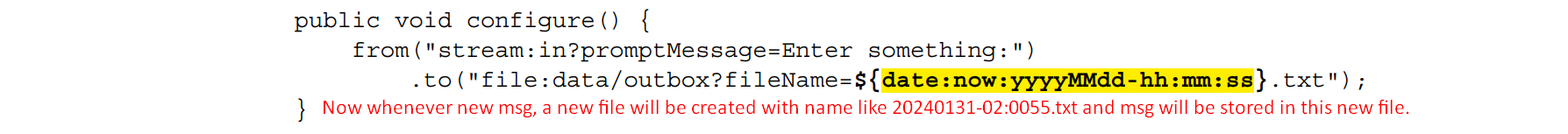
### 7.2.1 Reading and Writing a file with File Component

1. File Component is configured through URI Options.

#### Reading Files

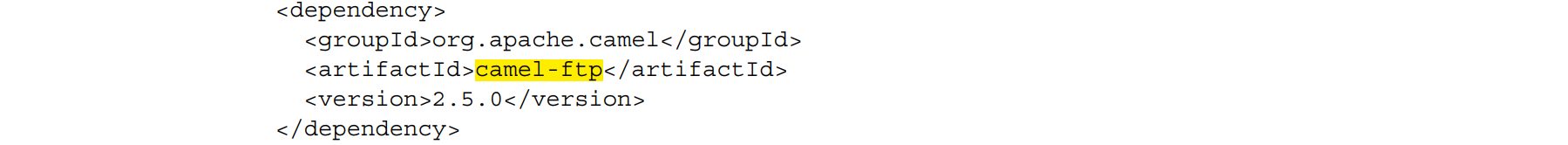
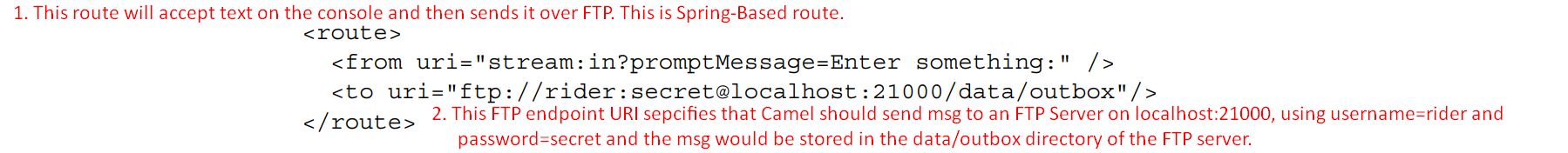
1. 
2. 
3. By default, Camel will lock the files being routed. Once route is completed, lock will be released.

#### Writing Files

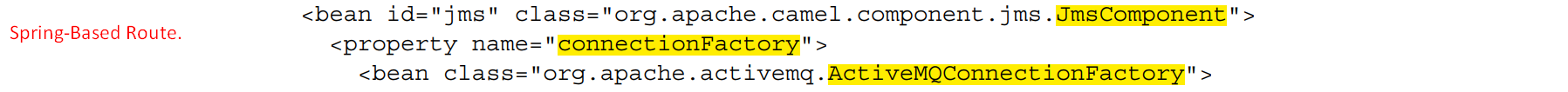
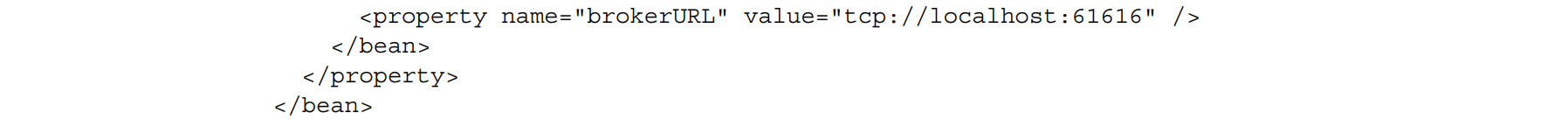
1. 
2.   
   Each time there is a msg to be stored into the file, the previous file will be replaced with the file having new msg.
3. 
4. All that we discussed about file reading and writing is enough for most of the cases whereas for the trickier cases, there are a plethora of configurations in doc.

# 7.2.2 Accessing Remote Files with the FTP Component

1. A white background with black text

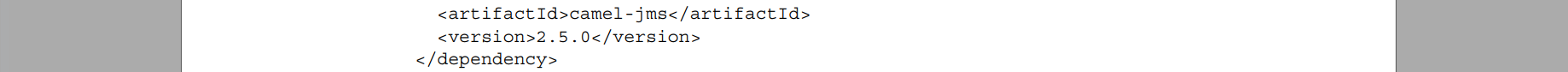
   Description automatically generated
2. FTP Component inherits all the features and options of File Component and many more.
3. As FTP not part of camel-core, need to add its dependency.  
   
4. 
5. The FTP Root directory is set to the current directory of this app.

# 7.3 Asynchronous Messaging (JMS Component)

1. JMS Messaging is an incredibly useful integration technology.
   1. It promotes loose coupling in app design.
   2. Built-in support for reliable messaging.
   3. By nature, asynchronous.
2. Came doesn’t ship with JMS Provider.   
   So, we need to configure a specific JMS Provider by passing in a **Jms.ConnectionFactory** instance.
3. **For example**, to connect to an Apache ActiveMQ broker listener listening on port 61616 of the local host, we could JMS Component like this:  
     
     
   As we are using ActiveMQConnectionFactory, so the tcp://localhost:61616 will be parsed by ActiveMQ.  
   This URI tells ActiveMQ to connect to a broker using TCP on port 61616 of the localhost.  
   ActiveMQ also supports connection over VM, SSL, UDP, multicast and so on.
4. Through this section, we will be demonstrating JMS Concepts using ActiveMQ as the JMS Provider but any provider could have been used here.
5. A screenshot of a computer

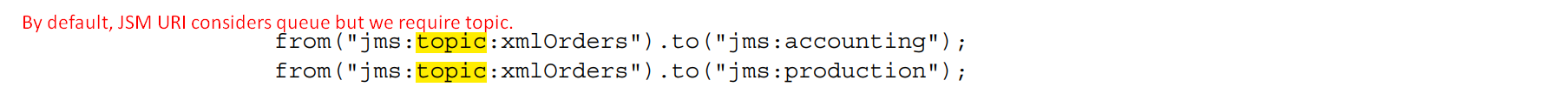
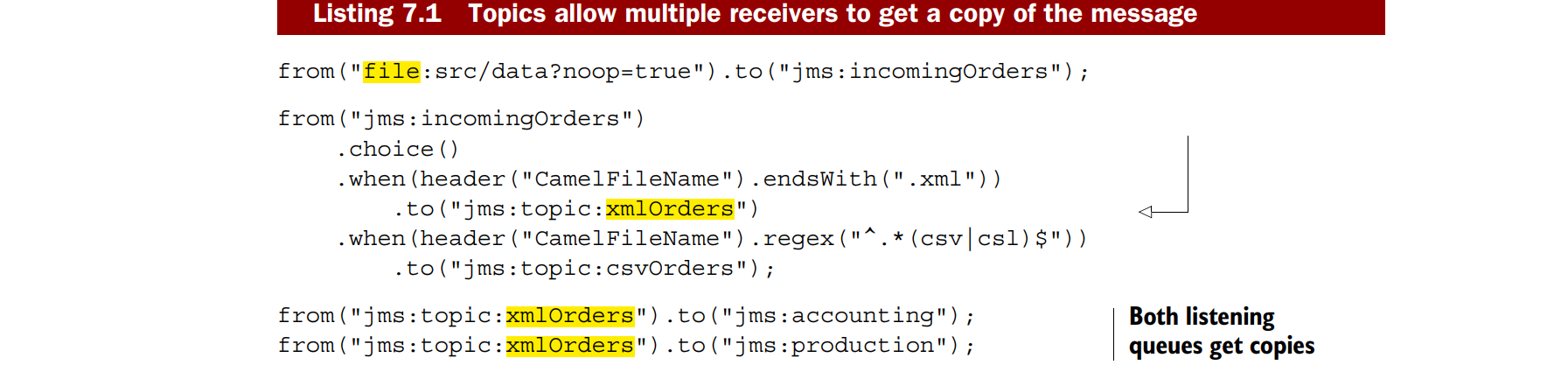
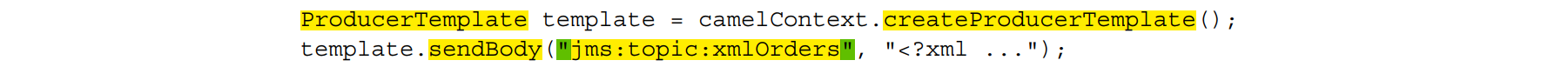
   Description automatically generated

In above snapshot, we discussed about **ActiveMQComponent** instead of **JmsComponent** which provides connections pooling.

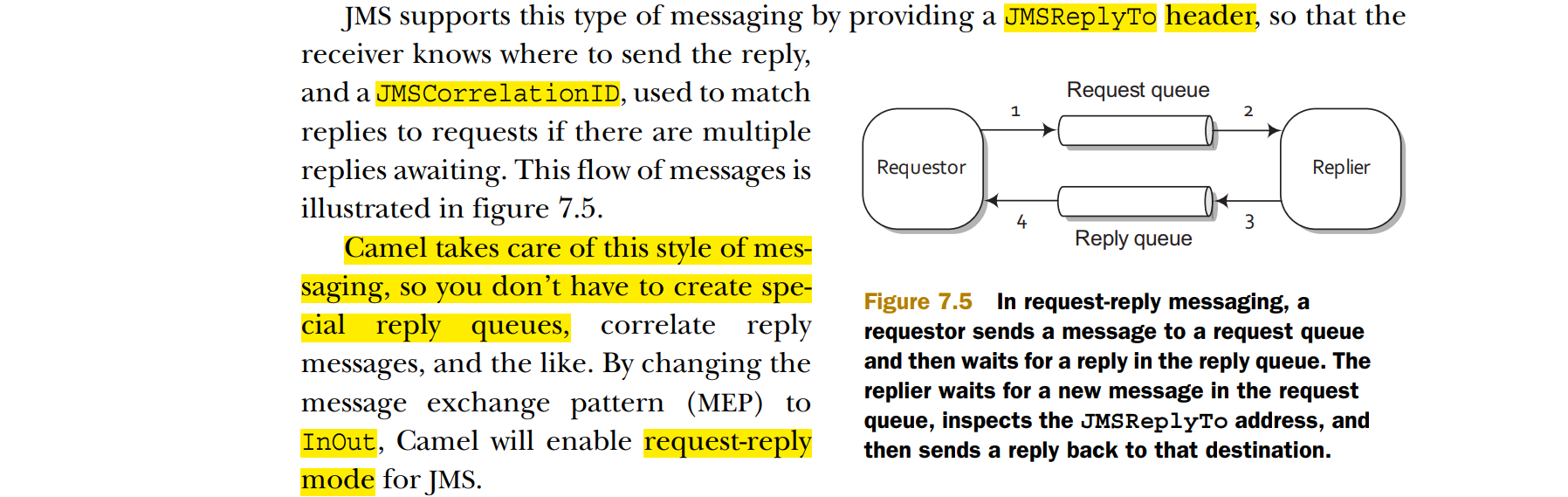
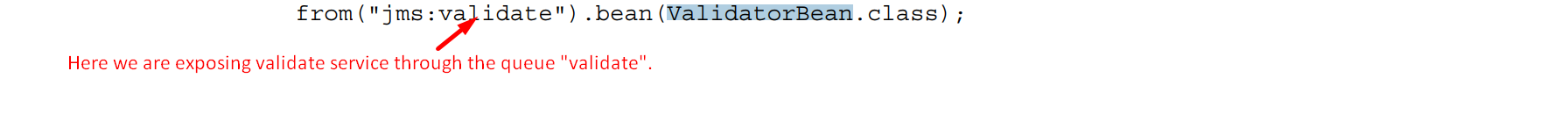
1. But to use JmsComponent instead of **ActiveMQComponent**, we need to include the following dependencies.  
    
2. Let us see how to send and receive msgs over JMS.

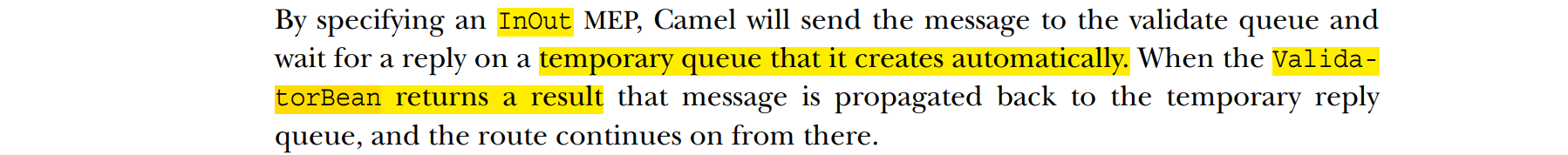
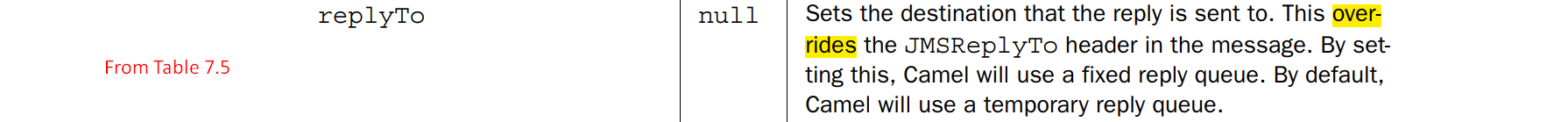
# 7.3.1 Sending and Receiving Messages

1. In Chapter 02, we sent the same msg to two endpoints using Multicast EIP was used but we can use Publish-Subscriber model using topic. A screenshot of a computer

   Description automatically generated  
   
2. 
3. Messages can also be sent “By Hand” to a JMS Destination using **ProducerTemplate**.
4. A Template class is usually a utility class which simplifies the access to the API. In this case **Producer Interface**.
5. Sending (Producing) a msg using **ProducerTemplate**

# 7.3.2 Request-Reply Messaging

1. JMS messaging with Camel (and in general) is asynchronous by default.
2. Messages are sent to the destination and the client doesn’t wait for the reply.
3. But sometimes we need to wait for the reply.
4. JSM supports such kind of messaging by providing a **JMSReply** Header, so that the receiver knows where to send the reply and a **JMSCorrelationID**, used to match replies to requests if there are multiple replies awaiting. 
5. 
6. A screenshot of a computer

   Description automatically generated
7. 
8. 
9. 
10. A close up of text

    Description automatically generated

# 7.3.3 Message Mappings

1. Camel hides a lot of details when doing JMS Messaging, so we don’t need to worry about.
2. But one detail we should be aware of is that Camel maps both bodies and headers from arbitrary types and names allowed in Camel to JMS-Specific types.

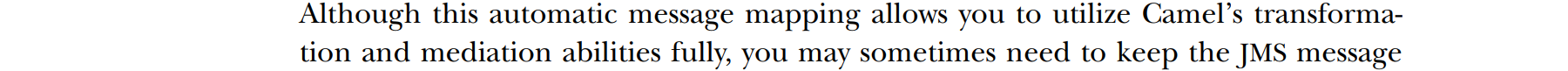
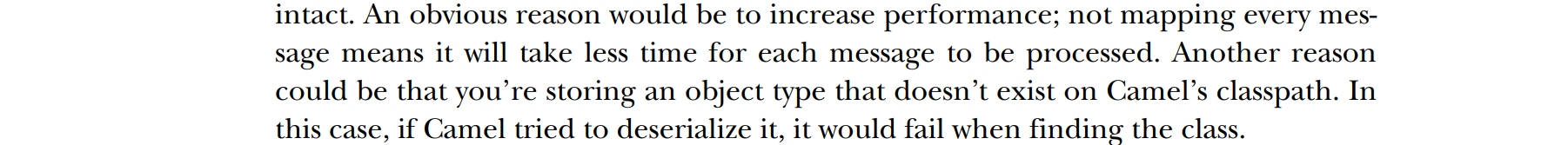
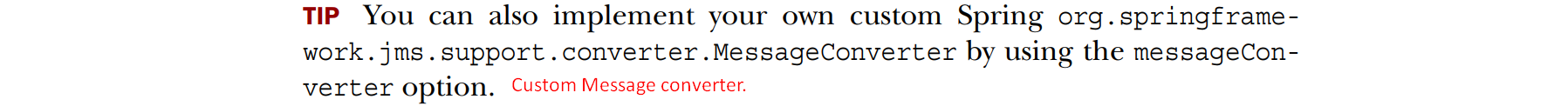
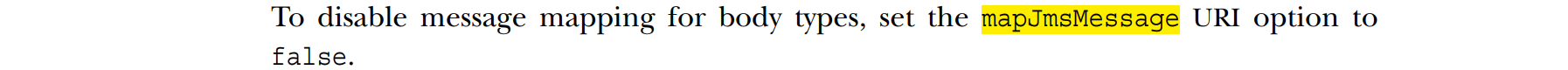
## Body Mapping

1. First Camel poses no restriction on what a message’s body contains.
2. JMS specifies different message types based on what the type of body in message in exchange.
3. A screenshot of a computer

   Description automatically generated
4. A close up of text

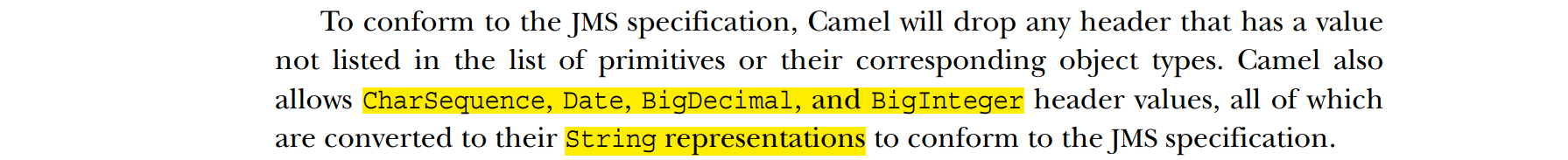
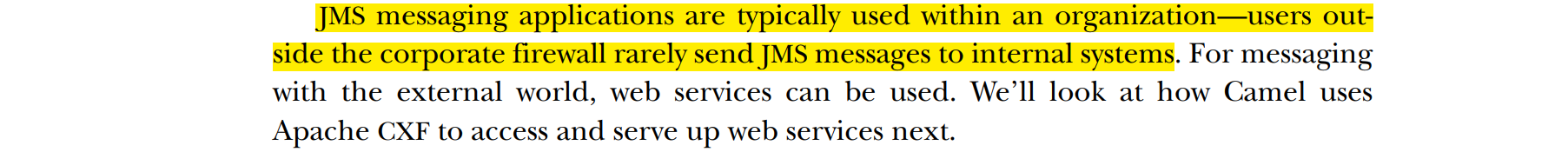
   Description automatically generated
5. A screenshot of a computer

   Description automatically generated
6. A screenshot of a computer

   Description automatically generated
7.   
   
8. 
9. 

# Header Mapping

1. Headers in JMS are even more restrictive than body types.
2. In Camel, Header Name anything, value any object.
3. It creates issue when sending and receiving to/from JMS destination.  
   A close up of text

   Description automatically generated
4. To handle these restrictions, Camel does a lot of things like.
   1. Any header name starting with JMS, will be dropped before sending to a JMS destination.
   2. Camel also attempts to convert the header names to be **JMS-compliant**. Like.
      1. Any dot will be replaced with **\_DOT\_**
      2. Any hyphen with **\_HYPHEN\_**
         1. **Example**:  
            org.apache.camel.Test-Header -> org\_DOT\_apache\_DOT\_camel\_DOT\_Test\_HYPHEN\_Header
   3. If this message is consumed by a Camel Route at some point down the line, the Header name will be converted back.  
      Jatin: Like we are sending msg to JMS destination so the above steps will be followed but before reaching the msg to the destination, if on the way somewhere msg is consumed by Camel Route, the reverse process will be followed.
   4. 
5. 

# 7.4 Web Services (CXF Component) : Skipped

1. Complete 7.4 and its subsections are skipped.

# 7.5 Networking (MINA Component): Skipped

# 7.6 Working With Database (JDBC and JPA Components)

# 7.7 In-Memory Messaging (Direct, SEDA, and VM Components)

1. Camel provides three main components in the core to handle in-memory messaging.
2. Components for in-memory messaging 🡺 Direct, SEDA, VM.
3. **Direct**d

|  |  |  |
| --- | --- | --- |
| **In-Memory Messaging** | | |
| **Direct** | **SEDA** | **VM** |
| For Synchronous Messaging | For Asynchronous Messaging | For Asynchronous Messaging |
|  | Used for communication within a single **CamelContext** | Used for communication within a JVM may contain multiple **CamelContext.** Like if we have loaded two **CamelContexts**  within a single application server, we can send msgs b/w them using the VM Component. |

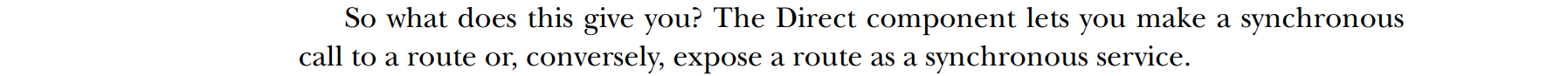
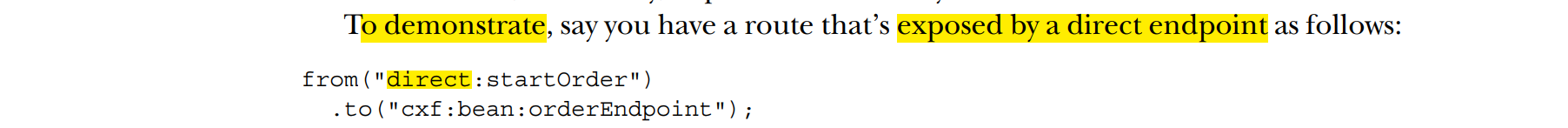
**NOTE** For more information on staged event-driven architecture (SEDA) in

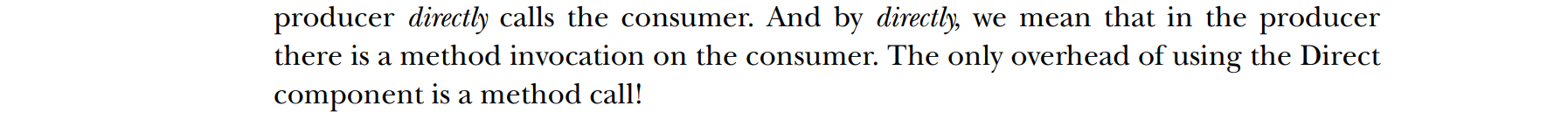
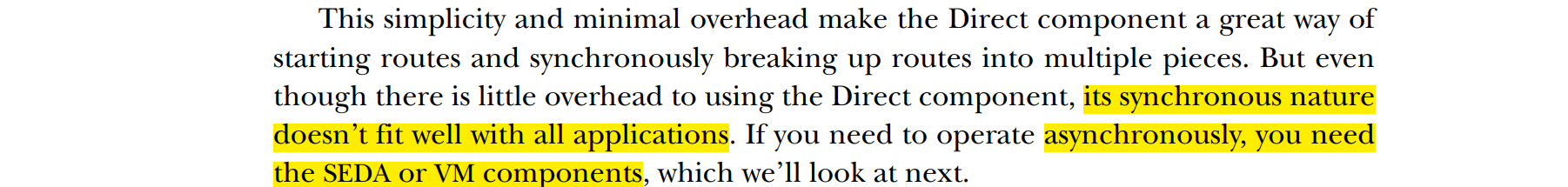
general, see Matt Welsh’s SEDA page at http://www.eecs.harvard.edu/~mdw/

proj/seda/.

Let’s look first at the Direct component.

# 7.7.1 Synchronous Messaging with the Direct Component

1. **Syntax**: direct:endpointname.
2. There is no configuration option just endpointname.
3. 
4. 
5. A text on a computer screen

   Description automatically generated
6.  

# 7.7.2 Asynchronous Messaging with SEDA and VM

1. So, if we are looking for in-memory solution but asynchronous, then we can go for SEDA and VM.
2. Ditching the JMS Specification conformance or the built-in reliability that JMS provides, the in-memory solution may be much faster.
3. Such solution doesn’t provide msg persistence as JMS provides.
4. Camel provides two in-memory queuing components 🡪 SEDA, VM.
5. They have common options.
6. One of the common uses of SEDA is to connect different routes (but within an app) together to form a routing application.
7. In the example that we took, only accounting and production departments are hosted on different machines but for everything else, we can use SEDA to connect routes in the same app to form a routing app.
8. A screenshot of a computer

   Description automatically generated
9. **Jatin**: SEDA should be used if the following conditions are true.
   1. The routes to be connected are within the same app and should be related to same CamelContext.
10. A screenshot of a computer

    Description automatically generated

A diagram of a diagram

Description automatically generated

1. The above with SEDA endpoint will behave in the same way as we discussed with JMS Endpoint.

# 7.8 Automating Tasks (Timer and Quartz Component): Skipped

# 7.9 Summary and Best Practices

1. 
2. A close-up of a text

   Description automatically generated
3. A close-up of a text

   Description automatically generated A close up of text

   Description automatically generated A close up of text

   Description automatically generated