Welcome to

5. Using components, JMS databases

KEA System Integration F2020 10 ECTS

Henrik Kramselund Jereminsen hkj@zencurity.com @kramse

Slides are available as PDF, kramse@Github

5-using-components-system-integration.tex in the repo security-courses

Plan for today

- EIP book chapters
- Using Camel components
- Examples of FTP, JMS, Networking, Databases, timing and scheduling databases
- Talk about protocols SMTP, IMAP, FTP, SFTP
- Try some tools, find examples of applications

Exercises

- Running ActiveMQ
- Camel book exercises
- Discussions about application that use these components

Reading Summary

EIP Chapter 5 Message Construction

EIP Chapter 6 Simple Messaging

Camel chapter 6: Using components

Todays Agenda - approximate time plan

- 08:30 45m Book chapters 5 and 6 discussion
- 45m Chapter 5 Message Construction Chapter 6 Simple Messaging
- 10:00 Break 15m
- 10:15 45m Exercise: Active MQ
- 45m 2x 45m Camebook chapter 6: Using components
  - contains
- 11:45 Lunch Break
- 12:30 Continued Camelbook chapter 6
- 45m Exercise:
- 14:00 Break 15m
- 45m Investigate real-life examples

Goals for today

Todays goals:

- Know more integration patterns
- See that Camel can be used for these patterns
- Try ActiveMQ
- Understand difference between SMTP submission, SMTP and IMAP

Photo by Thomas Galler on Unsplash

Part 1-2: EIP Chapters 5 and 6

Chapter 5 Message Construction

Message intent

> Message intent — Messages are ultimately just bundles of data, but the sender can have different intentions for what it expects the receiver to do with the message. It can send a Command Message, specifying a function or method on the receiver that the sender wishes to invoke. The sender is telling the receiver what code to run. It can send a Document Message, enabling the sender to transmit one of its data structures to the receiver. The sender is passing the data to the receiver, but not specifying what the receiver should necessarily do with it. Or it can send an Event Message, notifying the receiver of a change in the sender. The sender is not telling the receiver how to react, just providing notification.

Source: more or less same as the book

`https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingChannelsIntro.html`

Returning a response

> Returning a response — When an application sends a message, it often expects a response confirming that the message has been processed and providing the result. This is a Request-Reply scenario. The request is usually a Command Message, and the reply is a Document Message containing a result value or an exception. The requestor should specify a Return Address in the request to tell the replier what channel to use to transmit the reply. The requestor may have multiple requests in process, so the reply should contain a Correlation Identifier that specifies which request this reply corresponds to. [JMS11, pp.27-28]

Source: more or less same as the book

`https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingChannelsIntro.html`

Continued on the web site

- Message Construction

  `https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageConstructionIntro.html`

- Last part of chapter 5 includes Format Indicator

  `https://www.enterpriseintegrationpatterns.com/patterns/messaging/FormatIndicator.html`

Chapter 6 Simple Messaging

- Simple Message Examples

  `https://www.enterpriseintegrationpatterns.com/patterns/messaging/SimpleMessagingIntro.html`

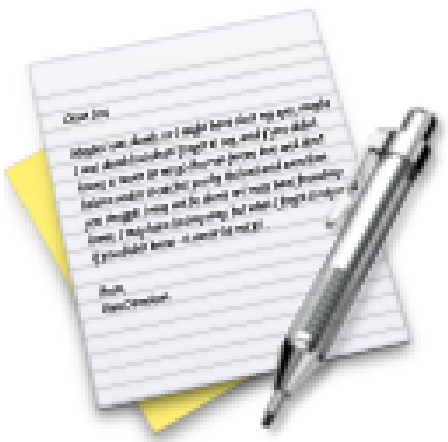
Part 3: Exercise with JMS Example ActiveMQ

Apache ActiveMQ

Apache ActiveMQ™ is the most popular open source, multi-protocol, Java-based messaging server. It supports industry standard protocols so users get the benefits of client choices across a broad range of languages and platforms. Connectivity from C, C++, Python, .Net, and more is available. Integrate your multi-platform applications using the ubiquitous AMQP protocol. Exchange messages between your web applications using STOMP over websockets. Manage your IoT devices using MQTT. Support your existing JMS infrastructure and beyond. ActiveMQ offers the power and flexibility to support any messaging use-case.

`https://activemq.apache.org/`

Exercise



Now lets do the exercise

Runing ActiveMQ

which is number **7** in the exercise PDF.

Part 4-5: Camel book Chapter 6 Using components

Camel Components

> Components are the primary extension point in Camel. Over the years since Camel's inception, the list of components has grown. As of version 2.20.1, Camel ships with more than 280 components, and dozens more are available separately from other com- munity sites.

Source: *Camel in action*, Claus Ibsen and Jonathan Anstey, 2018

Components chapter 6

Table 6.1 Components covered in this chapter

Component function	Component	Camel documentation reference
File I/O	File	http://camel.apache.org/file2.html
	FTP	http://camel.apache.org/ftp2.html
Asynchronous messaging	JMS	http://camel.apache.org/jms.html
Networking	Netty4	http://camel.apache.org/netty4.html
Working with databases	JDBC	http://camel.apache.org/jdbc.html
	JPA	http://camel.apache.org/jpa.html
In-memory messaging	Direct	http://camel.apache.org/direct.html
	Direct-VM	http://camel.apache.org/direct-vm.html
	SEDA	http://camel.apache.org/seda.html
	VM	http://camel.apache.org/vm.html
Automating tasks	Scheduler	http://camel.apache.org/scheduler.html
	Quartz2	http://camel.apache.org/quartz2.html

6.1 Overview of Camel components

From an API point of view, a Camel component is simple, consisting of a class implementing the `Component` interface, shown here:

```
public interface Component extends CamelContextAware {
    Endpoint createEndpoint(String uri) throws Exception;
    boolean useRawUri();
}
```

The main responsibility of a component is to be a factory for endpoints. To do this, a component also needs to extend `CamelContextAware`, which means it holds a reference to `CamelContext`. `CamelContext` provides access to Camel's common facilities, such as the registry, class loader, and type converters. This relationship is shown in figure 6.1.

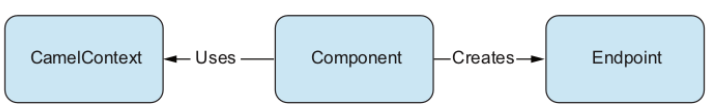


Figure 6.1 A component creates endpoints and may use the `CamelContext`'s facilities to accomplish this.

- Manually adding components
- Autodiscovering components

Autodiscovering components

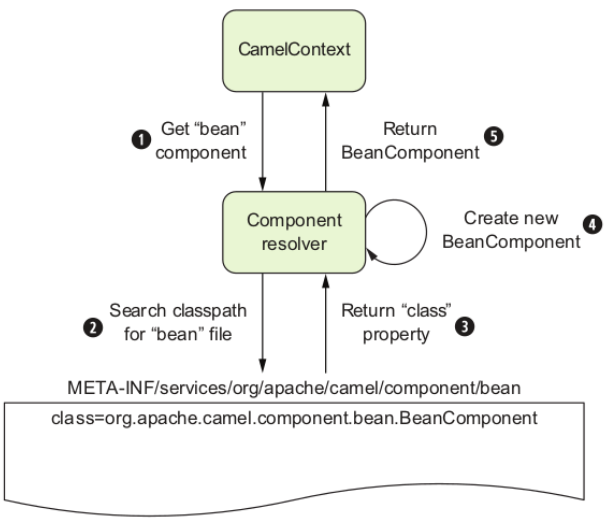


Figure 6.2 To autodiscover a component named "bean," the component resolver searches for a file named "bean" in a specific directory on the classpath. This file specifies that the component class that will be created is `BeanComponent`.

Autodiscovery is the way the components that ship with Camel are registered. To discover new components, Camel looks in the META-INF/services/org/apache/camel/ component directory on the classpath for files. Files in this directory determine the name of a component and the fully qualified class name.

6.2 Working with files: File and FTP components

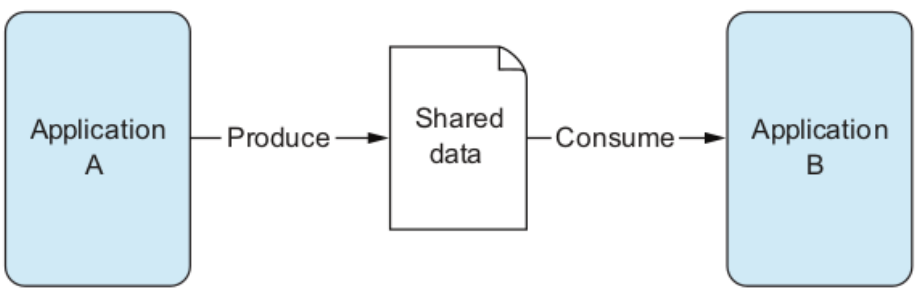


Figure 6.3 A file transfer between two applications is a common way to integrate with legacy systems.

- Reading and writing files with the File component,
- Accessing remote files with the FTP component

To see what happens firsthand, you can try the example by changing to the chapter6/file directory in the book's source code and executing the following command: `mvn camel:run`

FTP File Transfer Protocol

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network.

FTP is built on a client-server model architecture using separate control and data connections between the client and the server.[1] FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS) or **replaced with SSH File Transfer Protocol (SFTP)**.

`https://en.wikipedia.org/wiki/File_Transfer_Protocol`

Camel FTP support

Camel supports three flavors of FTP:

- Plain FTP mode transfer
- Secure FTP (SFTP) for secure transfer
- FTP Secure (FTPS) for transfer with the Transport Layer Se

The FTP component inherits all the features and options of the File component, and it adds a few more options, as shown in table 6.4. For a complete listing of options for the FTP component, see the online documentation (http://camel.apache.org/ftp2.html).

To run this example, change to the chapter6/ftp directory and run this command: `mvn camel:run`

Spring does not really want to work.

Java 8 on Debian 10/9/8

How do I Install Java 8 on Debian?. The first Oracle Java 8 stable version was released on Mar 18, 2014, and available to download and install. Oracle Java PPA for Debian systems is being maintained by Webupd8 Team. JAVA 8 is released with many of new features and security updates. This tutorial will help you to Install Java 8 on Debian 9/8/7 systems using PPA and apt-get command.

`https://tecadmin.net/install-java-8-on-debian/`

Install Java 8 on Debian

`sudo vim /etc/apt/sources.list.d/java-8-debian.list` and add following content in it:

```
deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main
deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main
```

Run the following to install

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
sudo apt-get update
sudo apt-get install oracle-java8-installer
java -version
sudo apt-get install oracle-java8-set-default
```

6.3 Asynchronous messaging: JMS component

- Sending and receiving messages
- Request-reply messaging, Message mappings

Camel with JMS

Camel doesn't ship with a JMS provider; you need to configure Camel to use a specific JMS provider by passing in a ConnectionFactory instance. For example, to connect to an Apache ActiveMQ broker listening on port 61616 of the local host, you could configure the JMS component like this:

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

The tcp://localhost:61616 URI passed in to ConnectionFactory is JMS provider- specific. In this example, you're using the ActiveMQConnectionFactory , so the URI is parsed by ActiveMQ. The URI tells ActiveMQ to connect to a broker by using TCP on port 61616 of the local host.

Connection Pooling

The ActiveMQ component
By default, a JMS ConnectionFactory doesn't pool connections to the broker, so it spins up new connections for every message. The way to avoid this is to use connection factories that use connection pooling.

For convenience to Camel users, ActiveMQ ships with the ActiveMQ component, which automatically configures connection pooling for improved performance. The ActiveMQ component is used as follows:

```
<bean id="activemq"
class="org.apache.activemq.camel.component.ActiveMQComponent">
<property name="brokerURL" value="tcp://localhost:61616"/>
</bean>
```

Same can be found with database connections.

Request-reply messaging

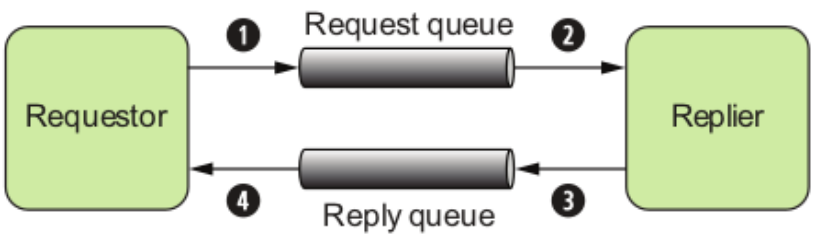


Figure 6.5 In request-reply messaging, a requestor sends a message to a request queue (1) and then waits for a reply in the reply queue (4). The replier waits for a new message in the request queue (2), inspects the `JMSReplyTo` address, and then sends a reply back to that destination (3).

Camel takes care of this style of messaging so you don't have to create special reply queues, correlate reply messages, and the like. By changing the message exchange pat- tern (MEP) to InOut , Camel will enable request-reply mode for JMS.

Message mappings: Sending to JMS

Table 6.6 When sending messages to a JMS destination, Camel body types are mapped to specific JMS message types

Camel body type	JMS message type
`String`, `org.w3c.dom.Node`	`TextMessage`
`byte[]`, `java.io.File`, `java.io.Reader`, `java.io.InputStream`, `java.nio.ByteBuffer`	`BytesMessage`
`java.util.Map`	`MapMessage`
`java.io.Serializable`	`ObjectMessage`

Message mappings: Receive from JMS

Table 6.7 When receiving messages from a JMS destination, JMS message types are mapped to Camel body types

JMS message type	Camel body type
`TextMessage`	`String`
`BytesMessage`	`byte[]`
`MapMessage`	`java.util.Map`
`ObjectMessage`	`Object`
`StreamMessage`	No mapping occurs

Note: To disable message mapping for body types, set the mapJmsMessage URI option to `false` .

Header mapping

Headers in JMS are even more restrictive than body types. In Camel, a header can be named anything that will fit in a Java String , and its value can be any Java object. This presents a few problems when sending to and receiving from JMS destinations. These are the restrictions in JMS:

- Header names that start with JMS are reserved; you can't use these header names.
- Header names must be valid Java identifiers.
- Header values can be any primitive type and their corresponding object types. These include boolean , byte , short , int , long , float , and double . Valid object types include Boolean , Byte , Short , Integer , Long , Float , Double , and String .

Great example of the conversion problems seen in system integration

6.4 Networking: Netty4 component

Another essential mode of integration is using low-level networking protocols, such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Even if you haven't heard of these protocols before, you've definitely used them—protocols such as email, FTP, and HTTP run on top of TCP.

To communicate over these and other protocols, Camel uses Netty and Apache MINA. Both Netty and MINA are networking frameworks that provide asynchronous event-driven APIs and communicate over various protocols including TCP and UDP. In this section, we use Netty to demonstrate low-level network communication with Camel.

- Using Netty for network programming
- Using custom codecs

TCP server with Netty

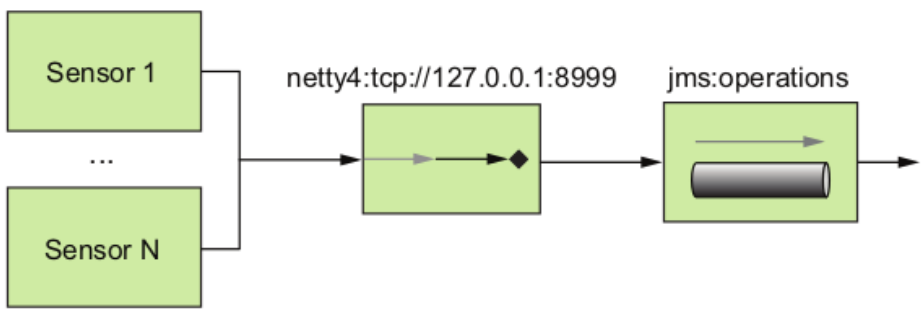


Figure 6.7 Sensors feed status messages over TCP to a server, which then forwards them to a JMS operations queue.

In Camel, a possible solution is accomplished with a single line:

```
from("netty4:tcp://localhost:8999?textline=true&sync=false")
.to("jms:operations");
```

Here you set up a TCP server on port 8999 by using Netty, and it parses messages by using the textline codec. The sync property is set to false to make this route InOnly —any clients sending a message won't get a reply back.

Textline codec

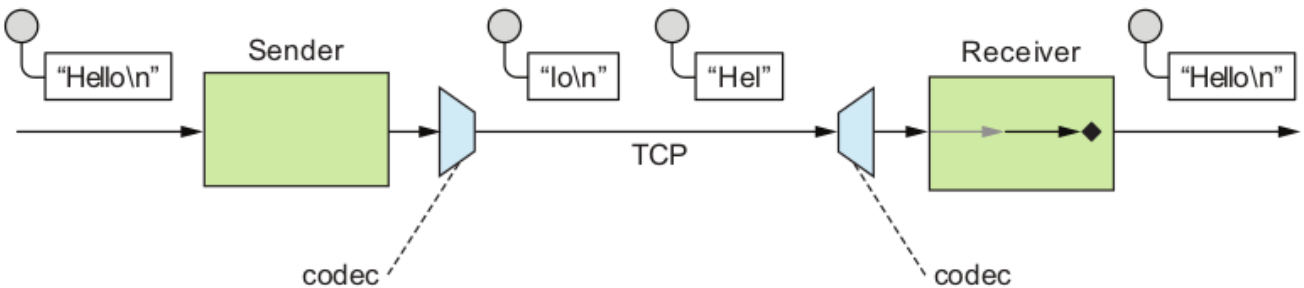


Figure 6.8 During TCP communications, a payload may be broken into multiple packets. A `textline` codec can assemble the TCP packets into a full payload by appending text until it encounters a delimiter character.

A codec decodes or encodes the message data into something that the applications on either end of the communications link can understand. As figure 6.8 illustrates, the textline codec is responsible for grabbing packets as they come in and trying to piece together a message that's terminated by a specified character.

This example is provided in the book's source in the chapter6/netty directory. Try it by using the following command: `mvn test -Dtest=NettyTcpTest`

6.5 Working with databases: JDBC and JPA components

- Accessing data with the JDBC component
- Persisting objects with the JPA component

Using Persistence

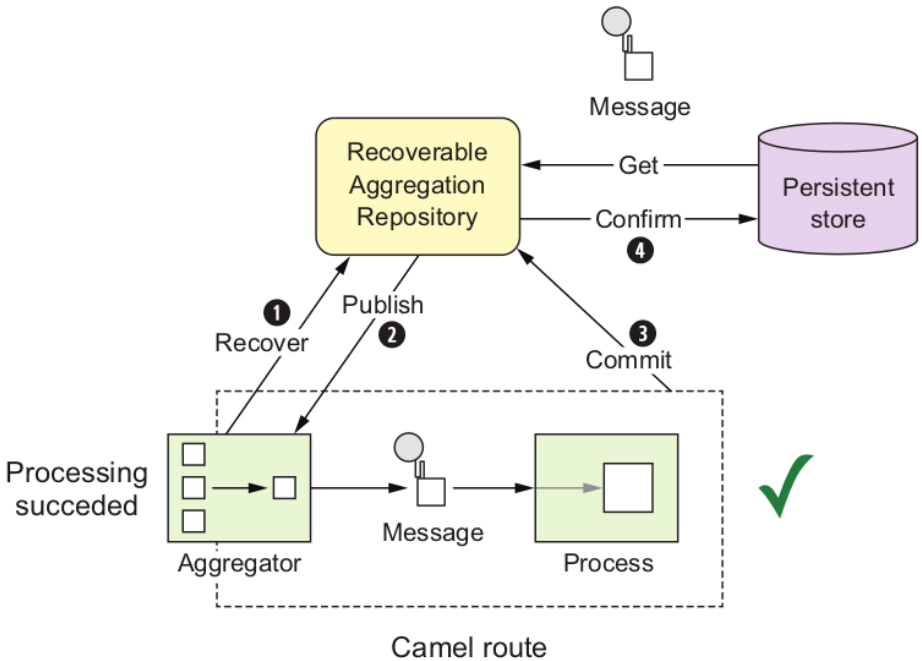


Figure 5.5 The Aggregator recovers failed messages ❶, which are published again ❷, and this time the messages completed ❸ successfully ❹.

Camel database support

In pretty much every enterprise-level application, you need to integrate with a database at some point, so it makes sense that Camel has first-class support for accessing databases. Camel has five components that let you access databases in various ways:

- *JDBC component* – Allows you to access JDBC APIs from a Camel route. item SQL component—Allows you to write SQL statements directly into the URI of the component for using simple queries. This component can also be used for call- ing stored procedures.
- *JPA component* – Persists Java objects to a relational database by using the Java Persistence Architecture.
- *Hibernate component* – Persists Java objects by using the Hibernate framework. This component isn't distributed with Apache Camel because of licensing incompatibilities. You can find it at the camel-extra project (https://github.com/camel-extra/camel-extra).
- *MyBatis component* – Allows you to map Java objects to relational databases.

Hibernate framework

Hibernate ORM (or simply Hibernate) is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

Hibernate is free software that is distributed under the GNU Lesser General Public License 2.1.

Hibernate's primary feature is mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

`https://en.wikipedia.org/wiki/Hibernate_(framework)`

SQLite RDMBS

SQLite is a relational database management system (RDBMS) contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite is ACID-compliant and implements most of the SQL standard, generally following PostgreSQL syntax.

...

SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others.[8] SQLite has bindings to many programming languages.

`https://en.wikipedia.org/wiki/SQLite`

Both Hibernate and SQLite are very popular

JMS to Database example

To demonstrate the SQL command-message concept, let's revisit the order router at Rider Auto Parts. In the accounting department, when an order comes in on a JMS queue, the accountant's business applications can't use this data. They can only import data from a database. That means any incoming orders need to be put into the corporate database. Using Camel, a possible solution is illustrated in figure 6.10.

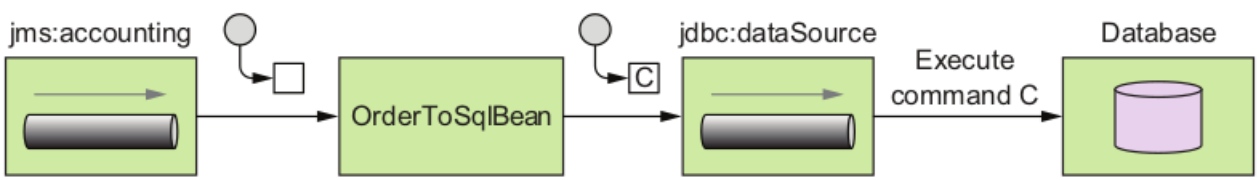


Figure 6.10 A message from the JMS accounting queue is transformed into an SQL command message by the `OrderToSqlBean` bean. The JDBC component then executes this command against its configured data source.

```
from("jms:accounting")
.to("bean:orderToSql")
.to("jdbc:dataSource?useHeadersAsParameters=true");
```

Uses a bean for mapping

Listing 6.3 A bean that converts an incoming order to a SQL statement:

```
public class OrderToSqlBean {
    public String toSql(@XPath("order/@name") String name,
                        @XPath("order/@amount") int amount,
                        @XPath("order/@customer") String customer,
                        @Headers Map<String, Object> outHeaders) {
        outHeaders.put("partName", name);
        outHeaders.put("quantity", amount);
        outHeaders.put("customer", customer);
        return "insert into incoming_orders"
            + "(part_name, quantity, customer) values"
            + " (:?partName, :?quantity, :?customer)";
    }
}
```

The last part is an SQL statement doing the *inserting* into the database

6.6 In-memory messaging: Direct, Direct-VM, SEDA, and VM components

Camel provides four main components in the core to handle in-memory messaging. For synchronous messaging, there are the Direct and Direct-VM components. For asyn- chronous messaging, there are the SEDA and VM components. The only difference between Direct and Direct-VM is that the Direct component can be used for communi- cation within a single CamelContext , whereas the Direct-VM component is a bit broader and can be used for communication within a JVM.

- Synchronous messaging with Direct and Direct-VM
- Asynchronous messaging with SEDA and VM

Advanced concept we won't go further into, staged event-driven architecture (SEDA) is described on Wikipedia too
`https://en.wikipedia.org/wiki/Staged_event--driven_architecture`

6.7 Automating tasks: Scheduler and Quartz2 components

Often in enterprise projects you need to schedule tasks to occur either at a specified time or at regular intervals. Camel supports this kind of service with the Timer, Sched- uler, and Quartz2 components. The Scheduler component is useful for simple recur- ring tasks, but when you need more control of when things get started, the Quartz2 component is a must. The Timer component can also be used for simple recurring tasks. The difference is the Scheduler component uses the improved Java scheduler API, as opposed to the Timer component which uses the older java.util.Timer API.

- Using the Scheduler component
- Enterprise scheduling with Quartz 231

Lets try them

You can run this simple example by changing to the chapter6/scheduler directory of the book's source and running this command: `mvn test -Dtest=SchedulerTest`

You can run this simple example by changing to the chapter6/quartz directory of the book's source and running this command: `mvn test -Dtest=QuartzTest`

To try an example using a cron trigger, browse to the chapter6/quartz directory and run the QuartzCronTest test case with this Maven command: `mvn test -Dtest=QuartzCronTest`

Make changes and rerun the examples also

6.8 Working with email

Camel provides several components to work with email:

- Mail component—This is the primary component for sending and receiving email in Camel.
- AWS-SES component—Allows you to send email by using the Amazon Simple Email Service (SES).
- GoogleMail component—Gives you access to Gmail via the Google Mail Web API.

- Sending mail with SMTP
- Receiving mail with IMAP

SMTP Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) is a communication protocol for electronic mail transmission. As an Internet standard, SMTP was first defined in 1982 by RFC 821, and updated in 2008 by RFC 5321 to Extended SMTP additions, which is the protocol variety in widespread use today. Mail servers and other message transfer agents use SMTP to send and receive mail messages.

`https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol`

Camel SMTP

Whenever you send an email, you're using the Simple Mail Transfer Protocol (SMTP) under the hood. In Camel, an SMTP URI looks like this: `[smtp|stmps]://[username@]host[:port][?options]`

Table 6.15 Common URI options used to configure the Mail component

Option	Default value	Description
`password`		The password of the user account corresponding to username in the URI.
`subject`		Sets the subject of the email being sent. You can override this value by setting a Subject message header.
`from`	`camel@localhost`	Sets what email address will be used for the From field of the email being sent.
`to`	`username@host`	The email address you're sending to. Multiple addresses must be separated by commas.
`cc`		The carbon copy (CC) email address you're sending to. Multiple addresses must be separated by commas.

SMTP should always be protected with password!

Postfix

Postfix is a free and open-source mail transfer agent (MTA) that routes and delivers electronic mail.

It is released under the IBM Public License 1.0 which is a free software license. Alternatively, starting with version 3.2.5, it is available under the Eclipse Public License 2.0 at the user's option.[2]

Originally written in 1997 by Wietse Venema at the IBM Thomas J. Watson Research Center in New York, and first released in December 1998[3], Postfix continues as of 2020 to be actively developed by its creator and other contributors.

Source: `https://en.wikipedia.org/wiki/Postfix_(software)`

Home page: `http://www.postfix.org/`

Dovecot

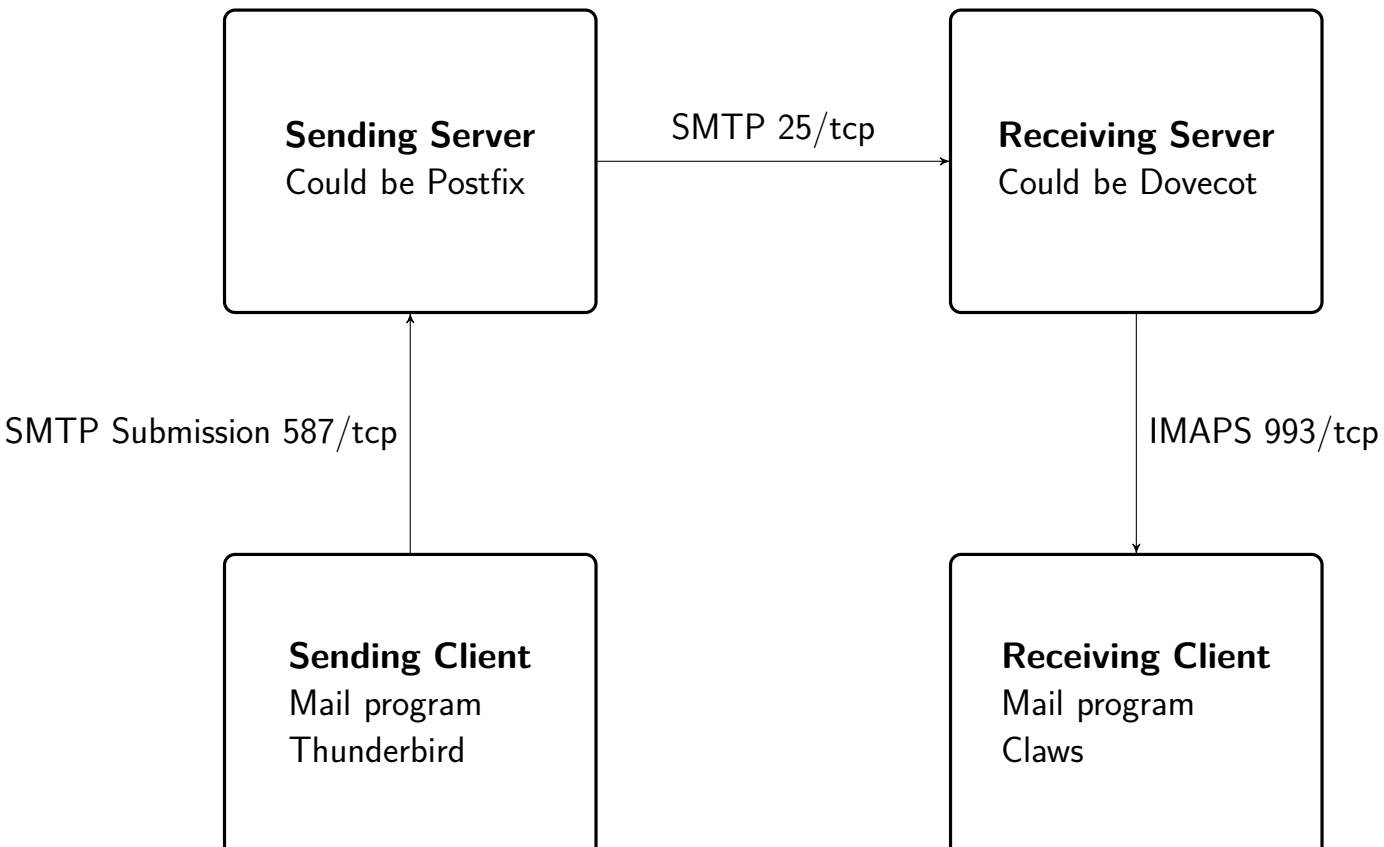
> Dovecot is an open-source IMAP and POP3 server for Unix-like operating systems, written primarily with security in mind.[3] Timo Sirainen originated Dovecot and first released it in July 2002. Dovecot developers primarily aim to produce a lightweight, fast and easy-to-set-up open-source email server.
>
> The primary purpose of Dovecot is to act as mail storage server. Mail is delivered to the server using some mail delivery agent (MDA) and stored for later access with an email client (mail user agent, or MUA).

Source: `https://en.wikipedia.org/wiki/Dovecot_(software)`

> Dovecot is an open source IMAP and POP3 email server for Linux/UNIX-like systems, written with security primarily in mind. Dovecot is an excellent choice for both small and large installations. It's fast, simple to set up, requires no special administration and it uses very little memory.

Home page: `https://www.dovecot.org/`

Example of SMTP and IMAP



6.9 Summary and best practices

The Camel website has documentation on all components available – We could cover only the most widely used and important components in this book, so if you need to use one of the many other components, documentation is available at `http://camel.apache.org/components.html`.

Part 6: Run application which incorporates system integration

A lot of applications are running ActiveMQ inside. A list is available on the web page:
`http://activemq.apache.org/projects-using-activemq`

There are also alternatives to ActiveMQ, one such is RabbitMQ.

RabbitMQ

RabbitMQ is an open-source message-broker software (sometimes called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols.[1]

The RabbitMQ server program is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages.

Source: `https://en.wikipedia.org/wiki/RabbitMQ`

Home page: `https://www.rabbitmq.com/`

Example from RabbitMQ tutorial: Sending

Sending

The following code fragment establishes a connection, makes sure the recipient queue exists, then sends a message and finally closes the connection.

```
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')

print(" [x] Sent 'Hello World!'")

connection.close()
```

Example from: `https://www.rabbitmq.com/tutorials/tutorial-one-python.html`

Example from RabbitMQ tutorial: Receiving

Receiving

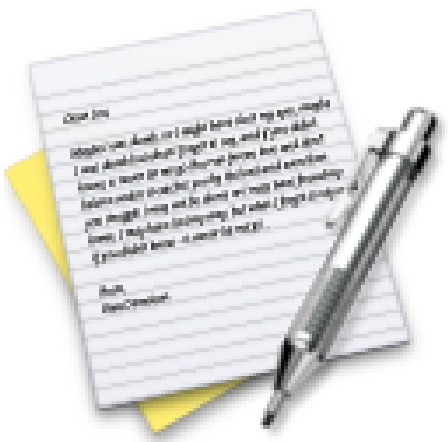
Similarly, the following program receives messages from the queue and prints them on the screen:

```
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

#channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)
# With apt package python-pika 0.11.0-4, this works:
channel.basic_consume(callback, 'hello', no_ack=True)

print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```

Exercise



Now lets do the exercise

Runing RabbitMQ with Python

which is number **8** in the exercise PDF.

Part 7: Do some research

Investigate real-life examples

Discussions about application that use these components

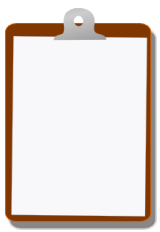
Find applications using Messages

1) Popular app, does Twitter use messaging?

2) One commercial application using or providing messaging

3) One Open Source application using or providing messaging

For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools