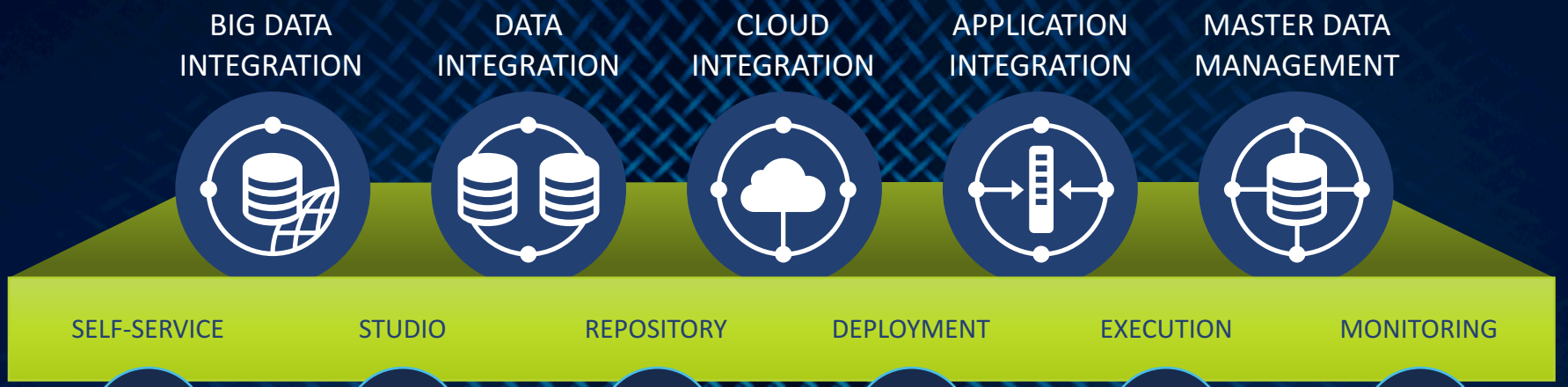
Talend Overview

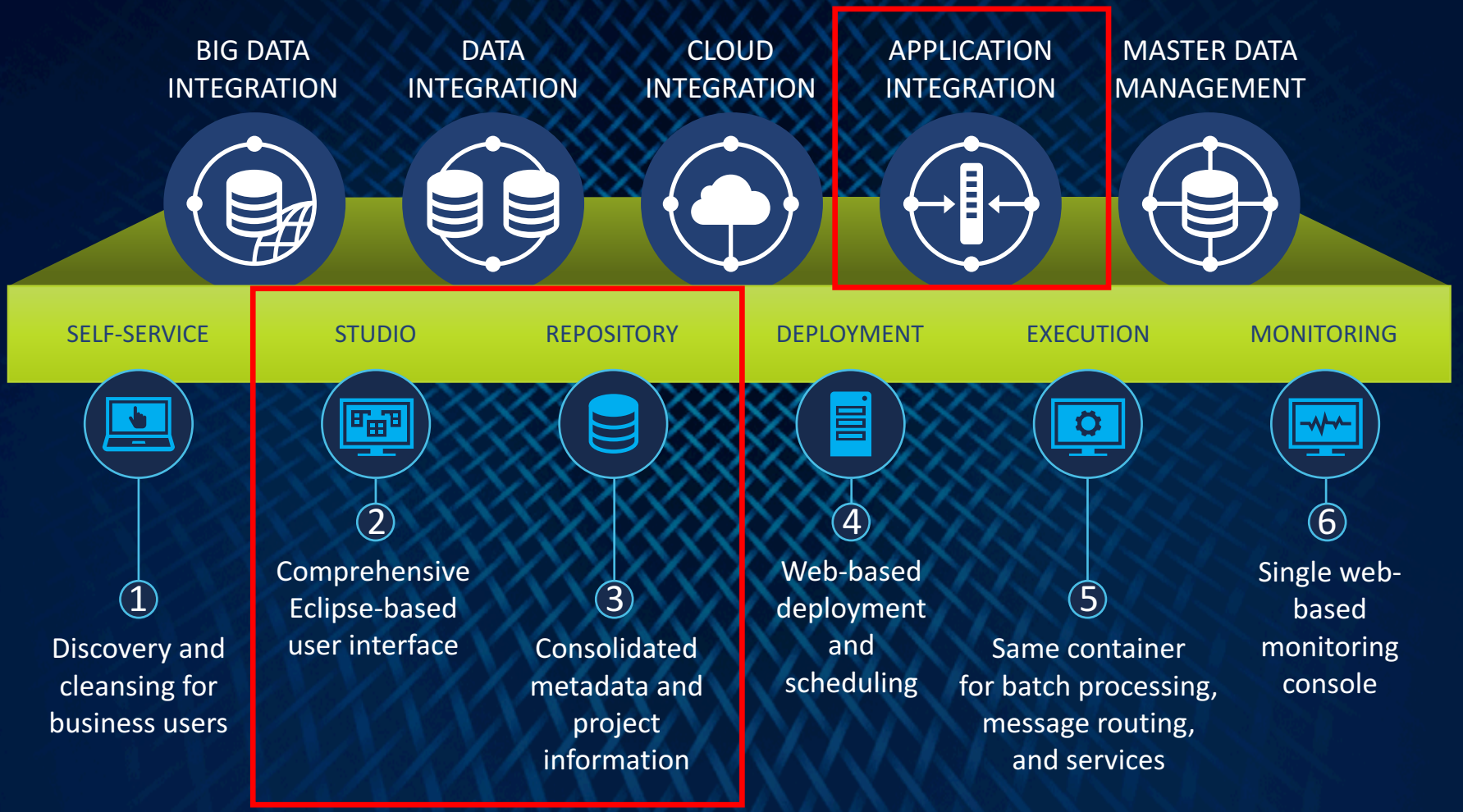
Products and Courses

talend | Data Fabric



1. Discovery and cleansing for business users
2. Comprehensive Eclipse-based user interface
3. Consolidated metadata and project information
4. Web-based deployment and scheduling
5. Same container for batch processing, message routing, and services
6. Single web-based monitoring console

talend | In this course, we'll focus on:



Objectives

At the end of this lesson, you will be able to:

- Discover the concepts and architecture behind ESB
- Describe the key ESB features
- Create mediation routes
- Exchange messages by using MOM
- Create SOAP and REST web services
- Access SOAP and REST services

Talend ESB Basics

Introduction to Talend ESB

Objectives

At the end of this lesson, you will be able to:

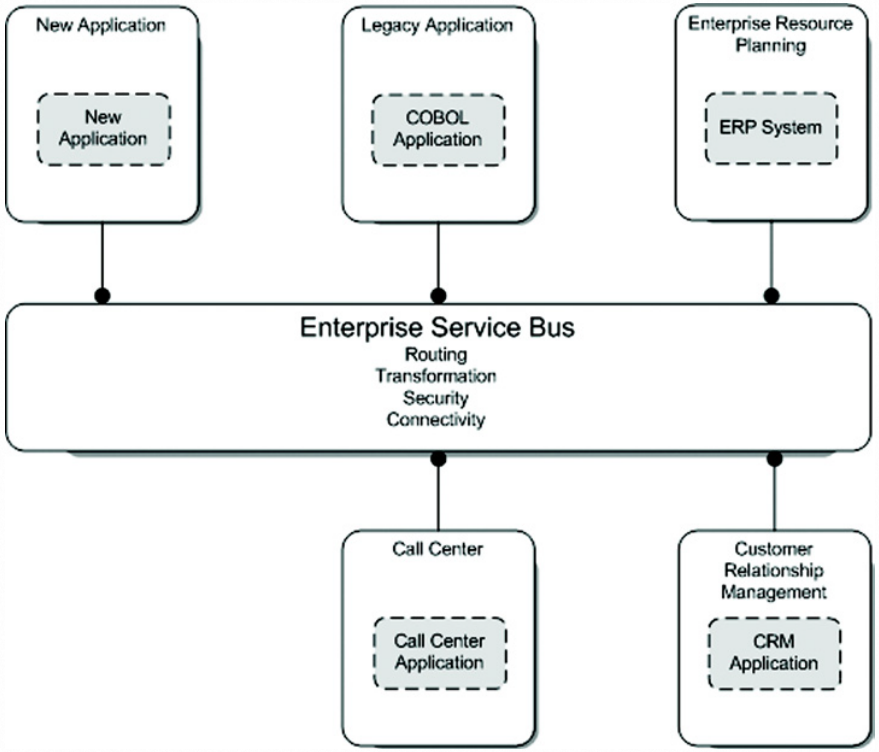
- Define ESB and application integration
- Explore ESB capabilities vs. Data Integration capabilities
- Identify the main ESB features
- Describe a simple ESB reference infrastructure

ESB concepts

- ESB stands for **enterprise service bus**
- Refers to the concepts and tools that allow **different heterogeneous applications** in the information system to **exchange messages** via standard formats in **real time**
- ESB is also referred to as **application integration**, as it facilitates communication between applications that were not designed to talk to each other

ESB concepts, *continued*

An ESB is a standards-based integration backbone that connects to applications and provides several services such as messaging, web services, message transformation, and intelligent routing



ESB concepts, *continued*

- Technically, the ESB is an evolution of the service-oriented architecture (SOA). The ESB supports all of its standards and allows developers to create web services
- As a layer on SOA, the ESB adds routing abilites to:
  - Process application messages as soon as they are issued
  - Define a pattern to process and redirect messages to other applications
- This makes the ESB an **event-driven architecture**

ESB vs. DI

How to choose between ESB and DI tools

Use DI Jobs when:

- Processing big batches of data
- The main purpose is to transform data
- The process can be scheduled and does not need real-time action

ESB vs. DI, *continued*

How do you choose between ESB and DI tools?

Use ESB routes or data services when:

- Processing single messages or single requests
- The main purpose is to react to a defined event
- Incoming messages or requests should trigger an action in real time

ESB modules and features

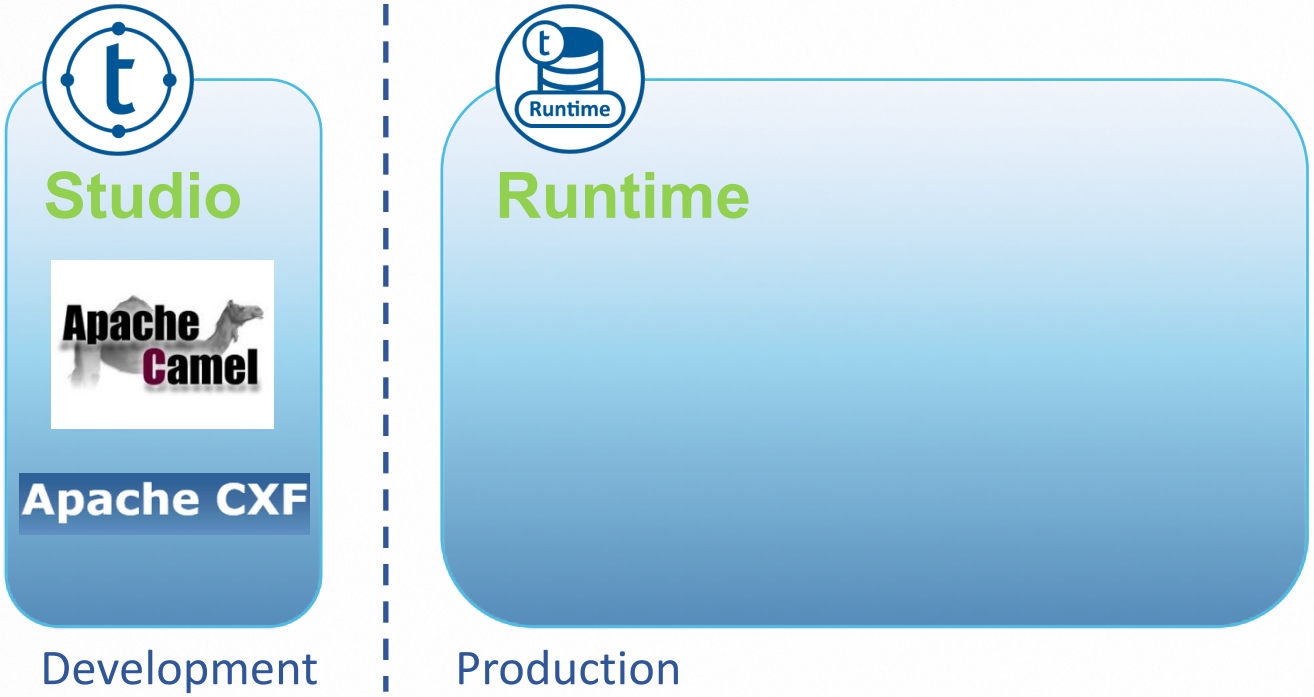
Studio

Talend Studio allows you to develop application integration programs called **routes** and **web services**

ESB modules and features

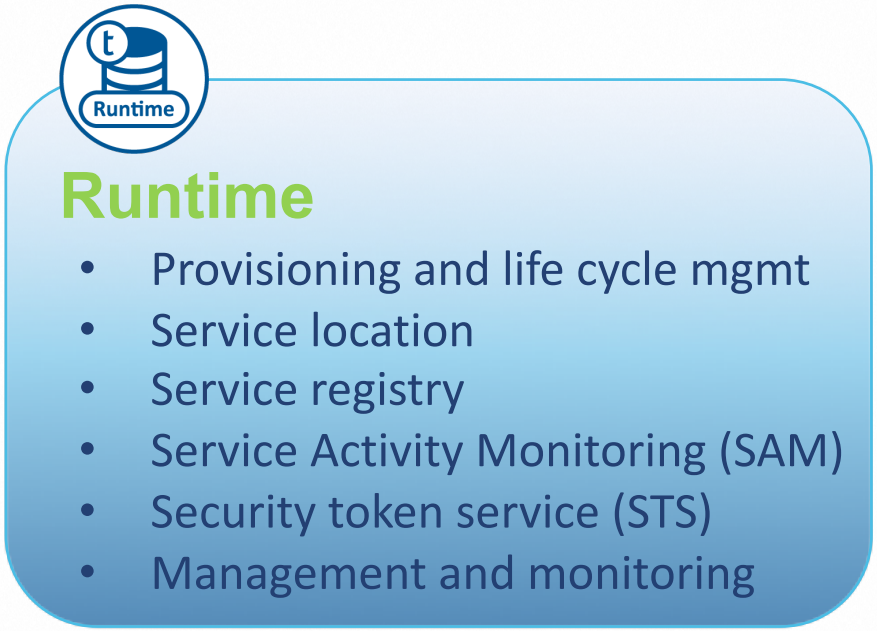
Talend Studio generates routes code using the **Apache Camel** framework and web services code using the **Apache CXF** framework

ESB modules and features



When going live, routes and web services are deployed on **Talend Runtime**, which is the deployment and **execution environment** for Talend ESB features. Talend Runtime can deploy, run, and expose natively Apache Camel- and Apache CXF-based bundles.

ESB modules and features



Development

Production

Talend Runtime also provides other key ESB features

ESB modules and features

Studio

Apache Camel

Apache CXF

development

Runtime

- Provisioning and life cycle mgmt
- Service location
- Service registry
- Service Activity Monitoring (SAM)
- Security token service (STS)
- Management and monitoring

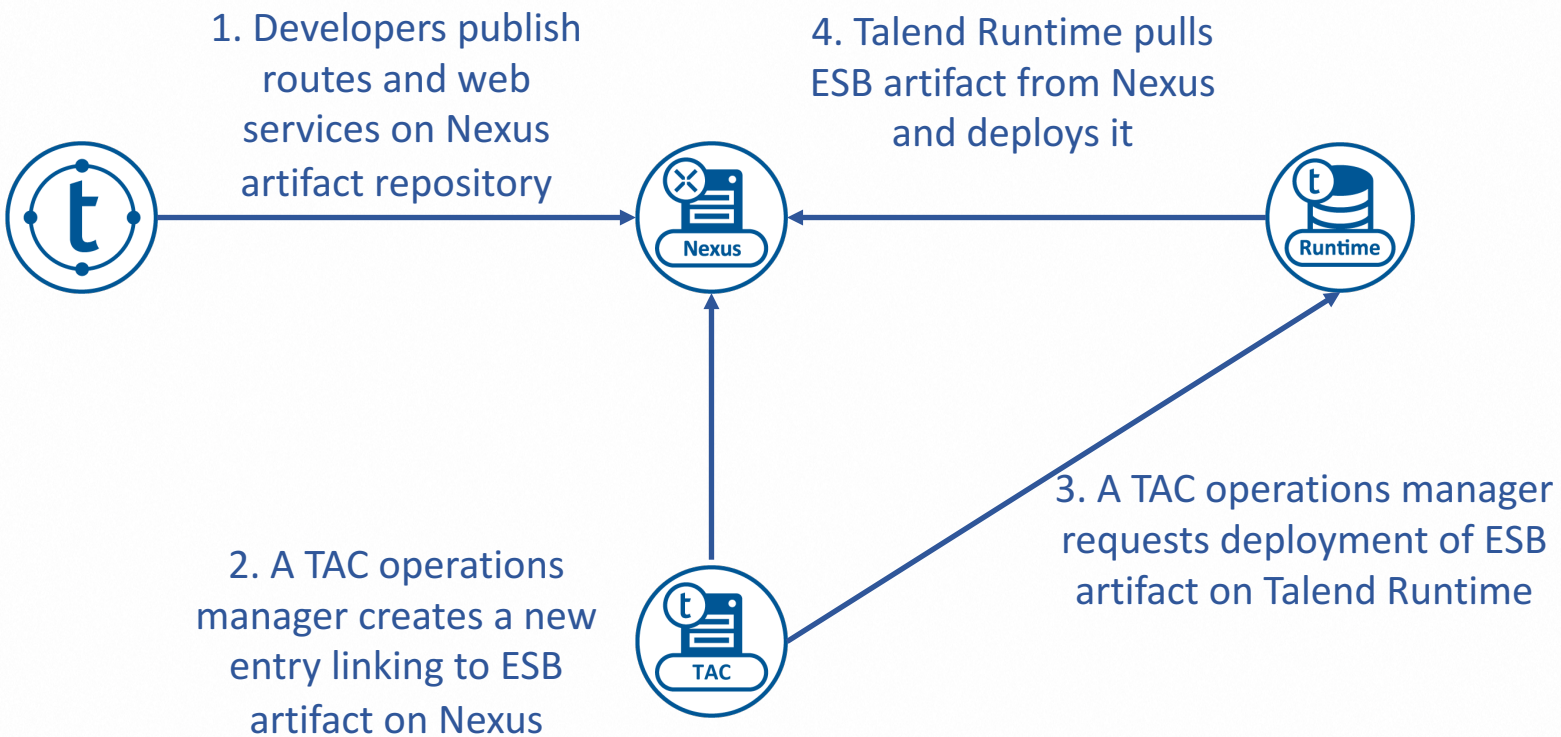
production

Message broker

ActiveMQ

A typical ESB architecture embeds a message broker to manage message distribution between applications. Talend ESB uses **Apache ActiveMQ** and is compatible with many other brokers.

ESB development life cycle



Lesson summary

- The enterprise service bus is a **real-time, event-driven** architecture
- Talend Studio for ESB allows you to develop SOAP and REST **web services,** as well as mediation **routes**
- Routes and services are deployed on a **Talend Runtime** instance
- A **message broker** such as **ActiveMQ** can be added to the ESB architecture to secure message distribution

Talend ESB Basics

Building Simple Mediation Routes with Apache Camel

Objectives

At the end of this lesson, you will be able to:

- Describe a mediation route use case
- Identify the main features of Apache Camel
- Describe a simple ESB reference infrastructure

ESB routes and data services

Talend Studio allows developers to develop application integration programs called **routes** and **data services**

ESB routes

- Talend routes are programs designed to catch messages on the information system and route (redirect) them to a destination defined by a set of rules and transformations
- Unlike Talend Jobs, routes are not supposed to end once deployed. Routes are **event-driven programs**, which means they keep scanning the information system for events and react in real time when they detect one

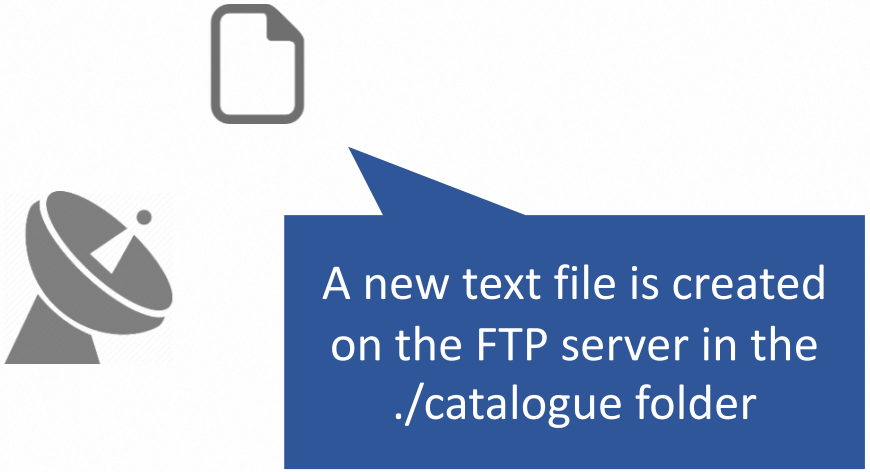
ESB routes: example

This is a consumer endpoint. Consider it an antenna or sensor: it listens in at a particular place in your ESB system. It is configured to catch a specific type of event.

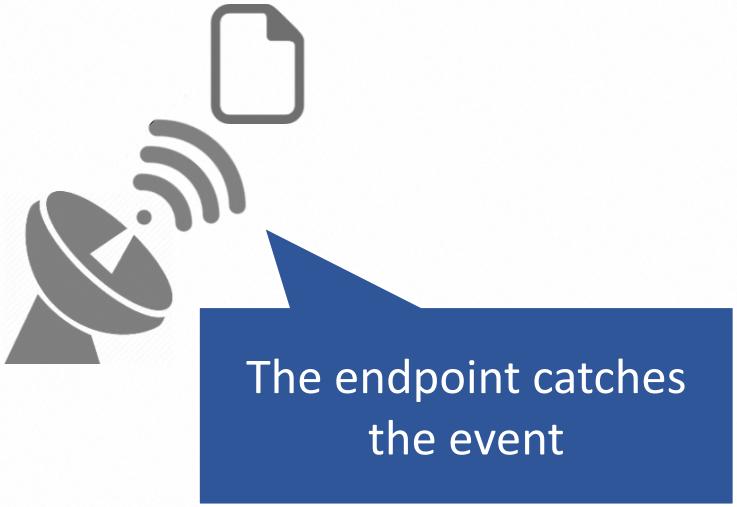
ESB routes: example, *continued*

This consumer endpoint is configured to listen to an FTP server and catch any new text file from the ./catalogue folder

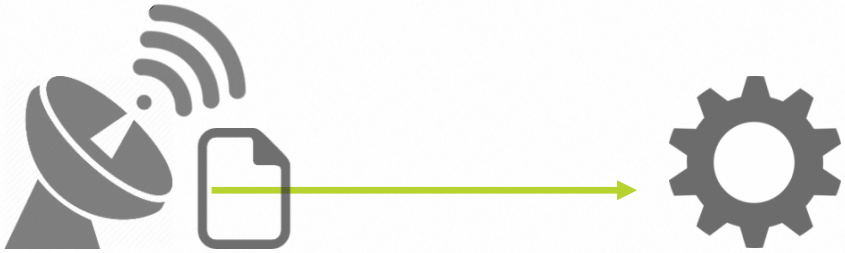
ESB routes: example, *continued*



ESB routes: example, *continued*

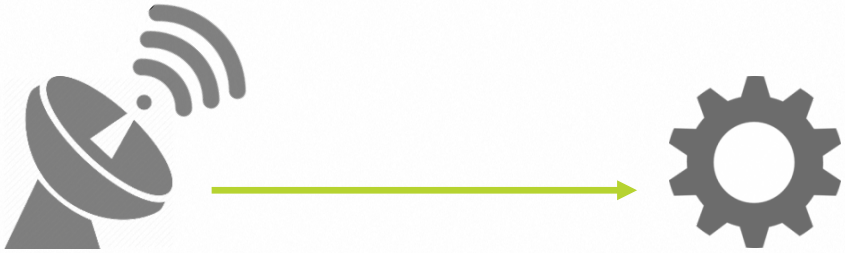


ESB routes: example, *continued*



The endpoint sends a message--the content of the file--to a route: a predetermined set of rules and transformations, and, finally, a destination

ESB routes: example, *continued*



The message is processed so that the text data is formatted in XML

ESB routes: example, *continued*

The message is then sent to its final destination (in this example, ActiveMQ)

Building mediation routes

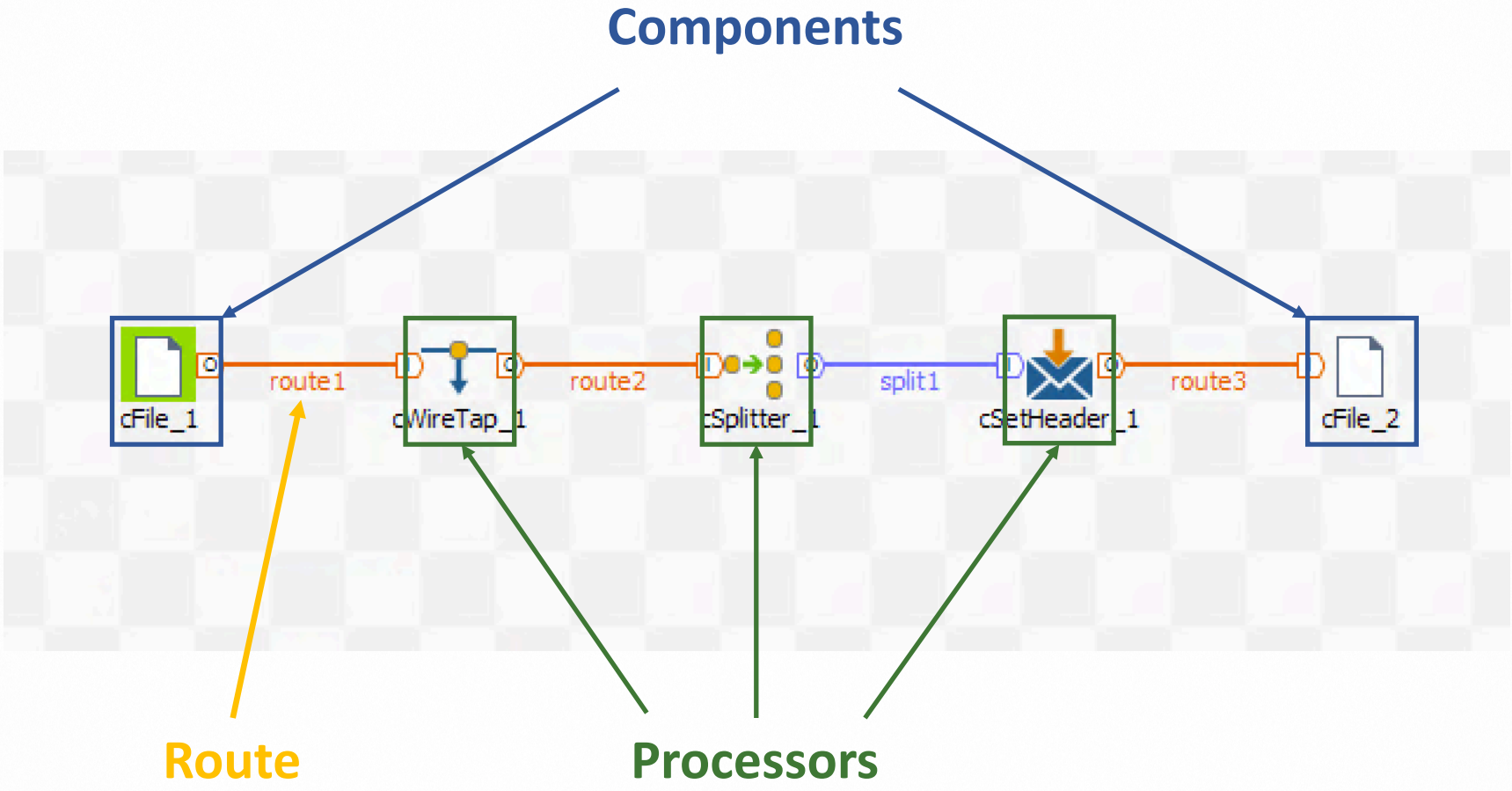
- Talend Studio helps developers build routes with graphical components. It then generates the corresponding code to build the routes
- Mediation routes are generated by using the **Apache Camel** routing engine framework

Building mediation routes

An Apache Camel route contains the following elements:

- **Components** provide the **interface** to connect to other sources or systems (file, HTTP, JMS). They handle connectivity, drivers, and protocols
- **Processors** handle **actions** between components. They can transform, enrich, validate, intercept, and even reroute messages
- **Routes** are sequences that **wire** components and processors together

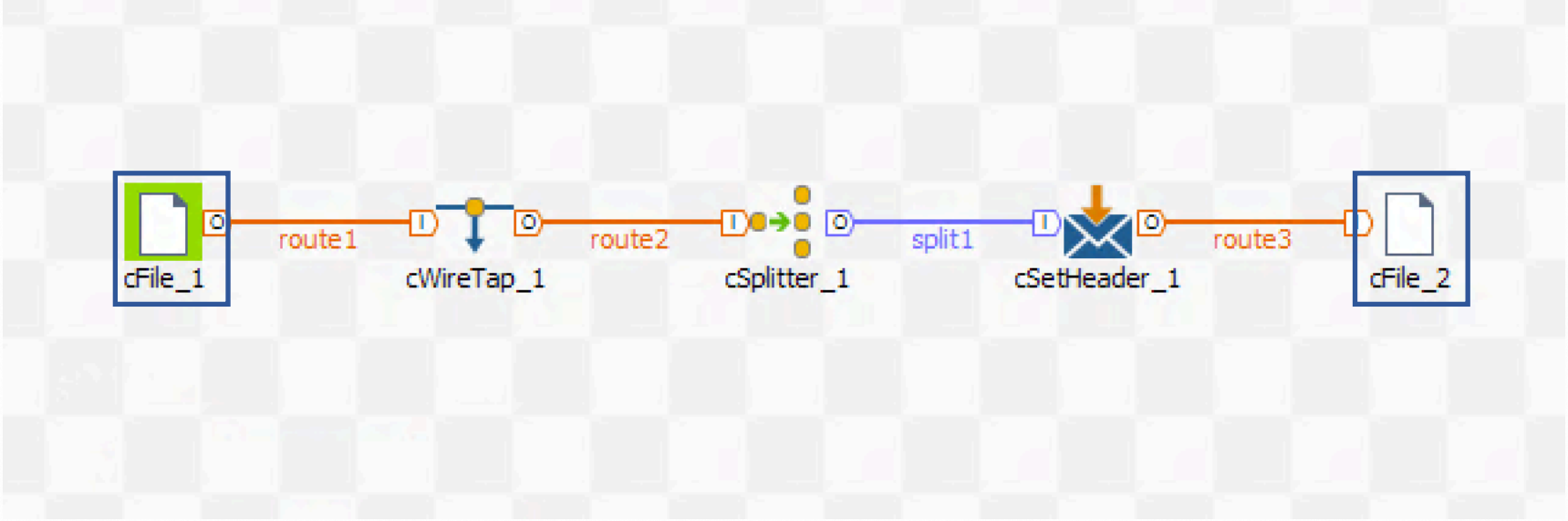
Building mediation routes: example



Endpoints

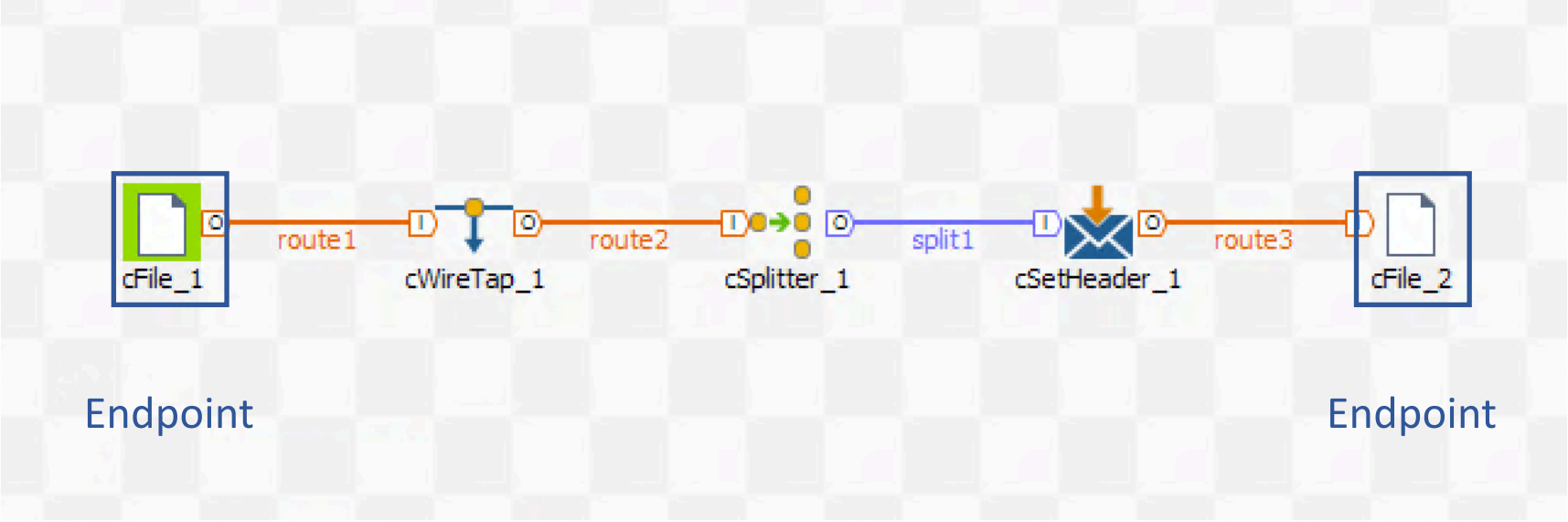
Unlike DI components, ESB (Apache Camel) components can be used as input or output for any route

- In this example, the file component reads a file when placed at the beginning of the route, and it writes a file when placed at the end of the route



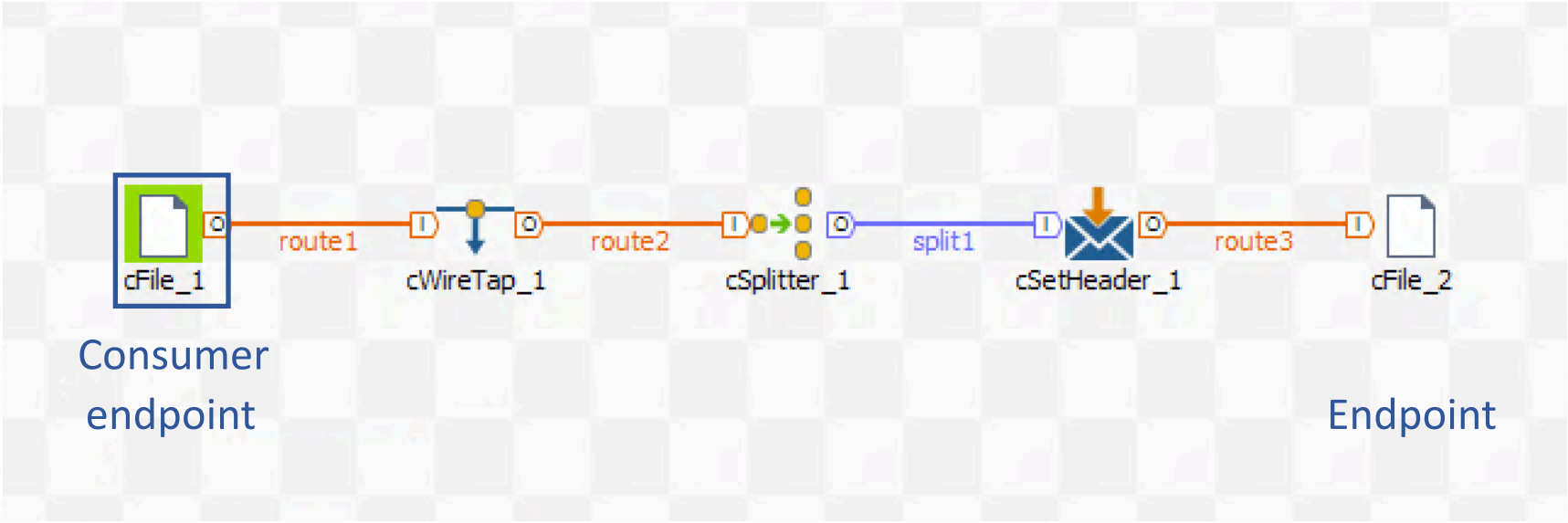
Endpoints, *continued*

Components placed at the beginning and end of a route are called **endpoints**



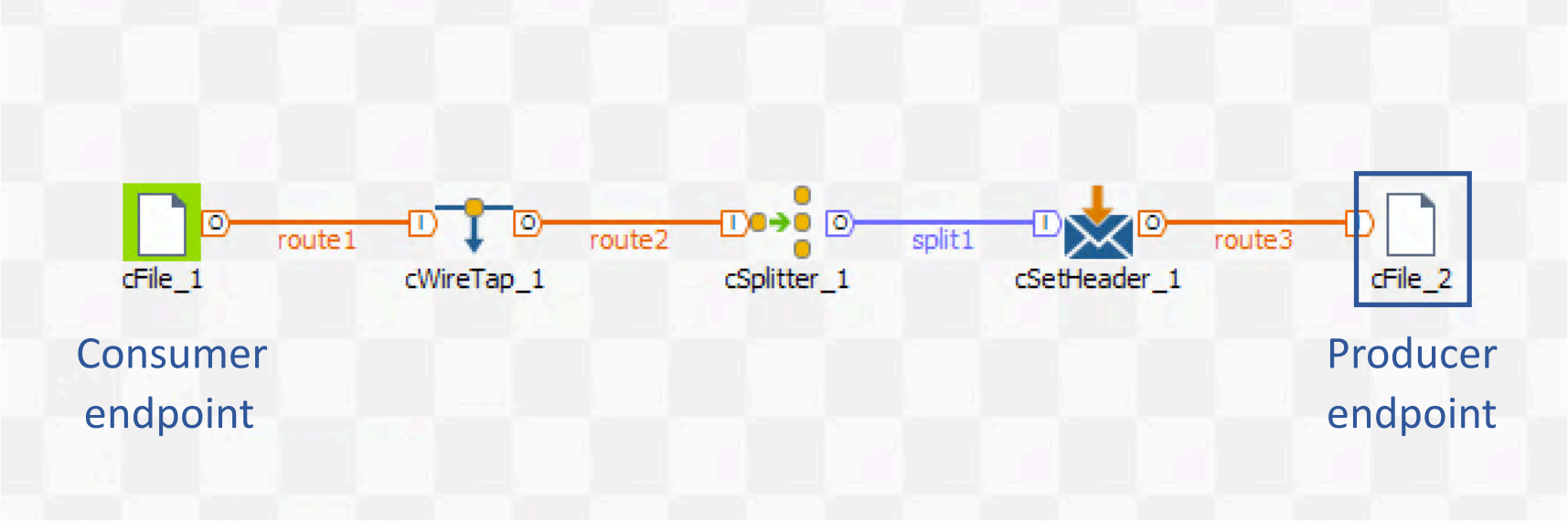
Endpoints, *continued*

- The endpoint at the beginning of a route is in charge of intercepting an event and sending it into the route for processing
- This endpoint is called a **consumer endpoint**, as it consumes any event it is programmed to catch



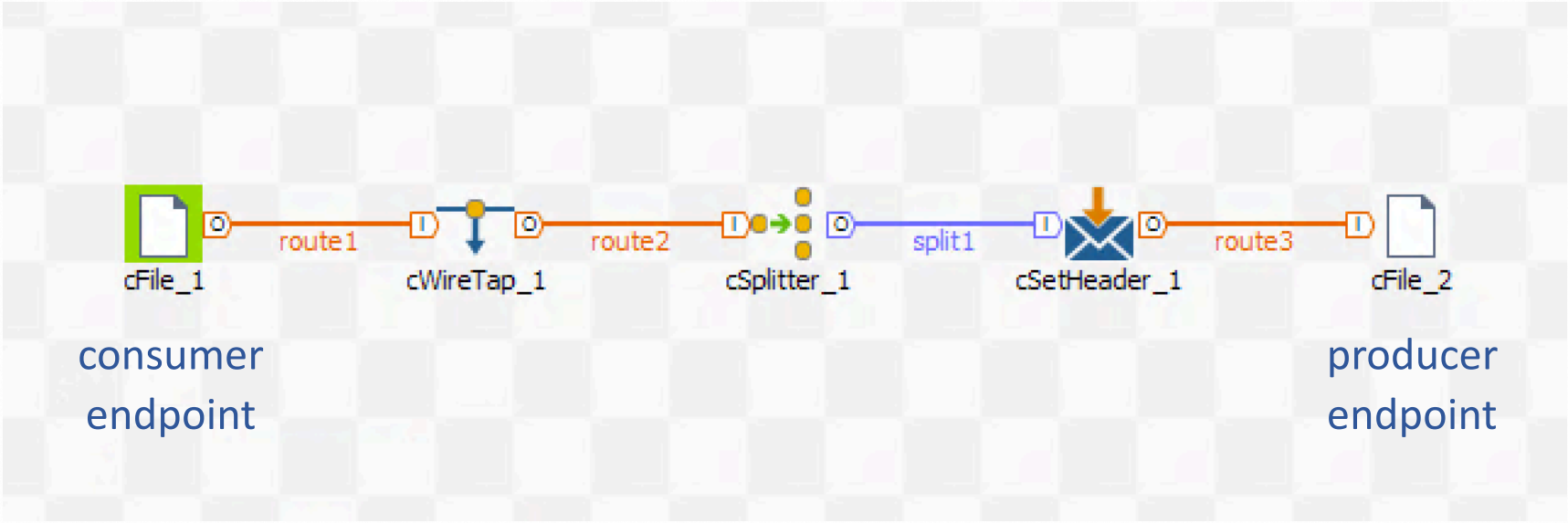
Endpoints, *continued*

- The endpoint at the end of a route is in charge of sending the processed message out of the route
- This endpoint is called a **producer endpoint,** as it produces an event on the interface that its component is configured to use



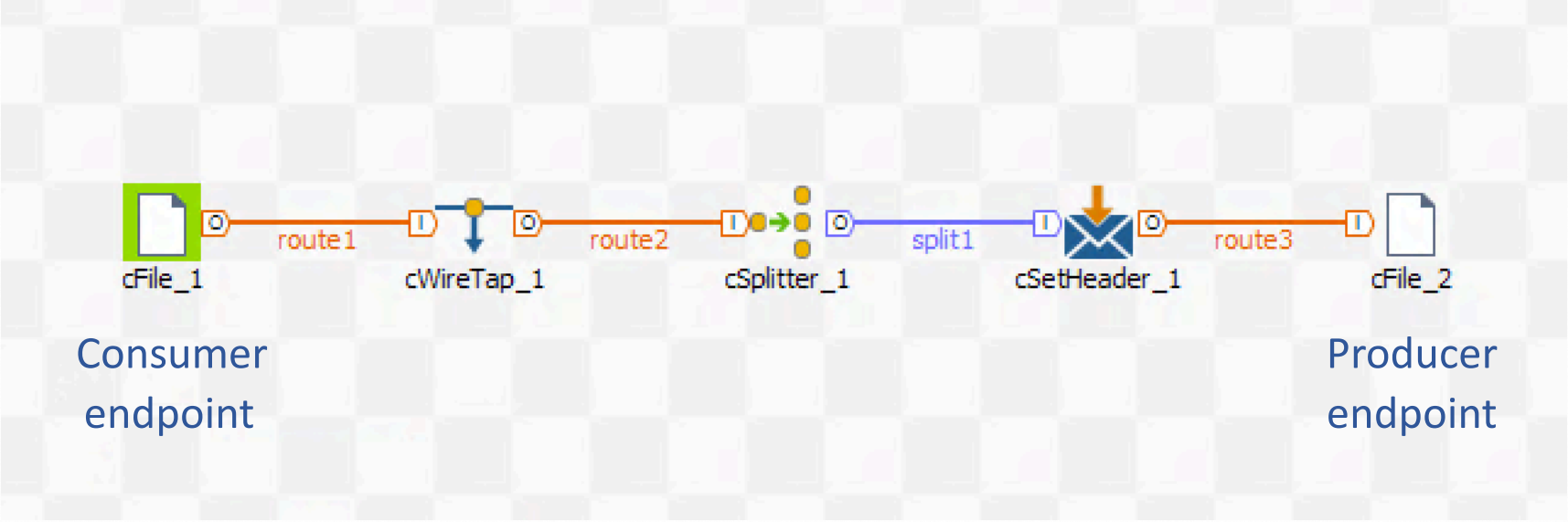
Endpoints, *continued*

- When the **consumer endpoint** catches an event, it converts it to a **message**
- The **message** follows the route--the **defined sequence of processors**--until it reaches the producer endpoint



Endpoints, *continued*

When the **message** reaches the **producer endpoint**, it **exits the route** via the interface defined in the producer endpoint component



Lab overview

Lab 1: simple mediation route

In this lab, you will:

- Create your first route
- Use the File component and configure it

Lesson summary

- ESB mediation routes are event-driven, real-time programs
- Routes catch events, convert them to messages, and route them through processors to their destinations
- A route starts with a consumer endpoint and ends with a producer endpoint

Talend ESB Basics

Messages, Exchanges, and the SIMPLE Language

Objectives

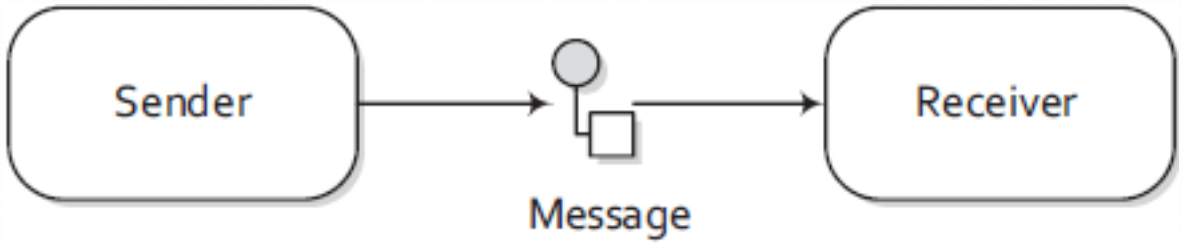
At the end of this lesson, you will be able to:

- Describe the structure of a message
- Describe the structure of an exchange
- Find header information in the Camel documentation
- Use the SIMPLE language in a mediation route

Camel message model

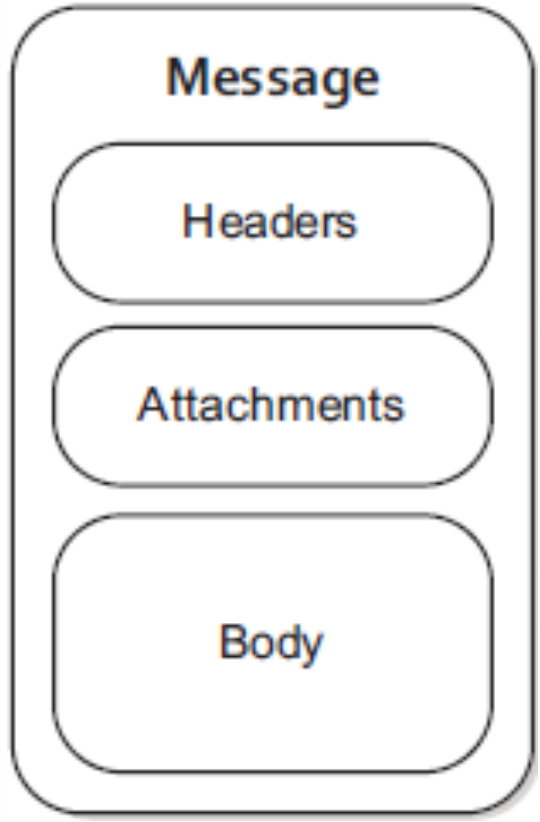
- Camel routes transport information by encapsulating it in two structures: the **message** and the **exchange**
- Each structure is a java class that can be accessed in the routes

Camel message



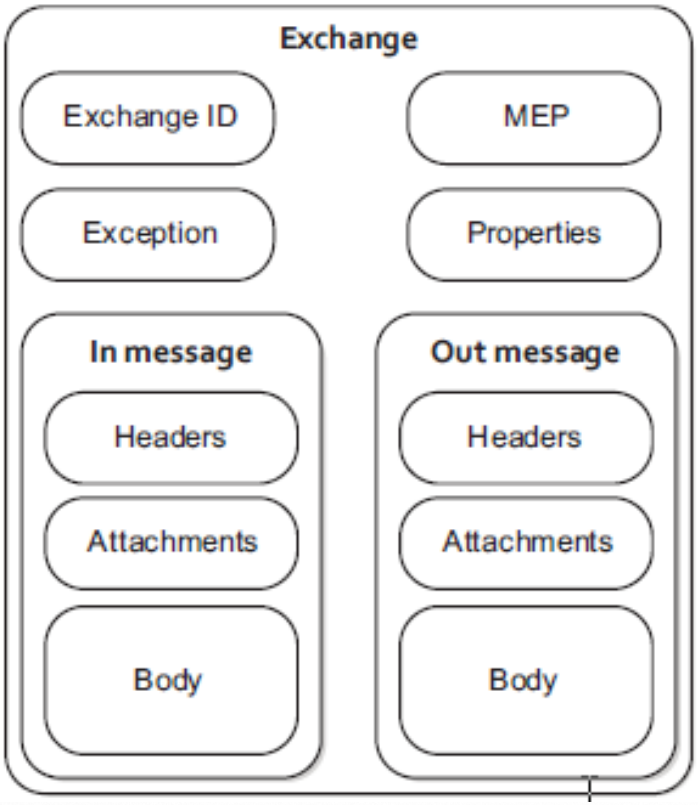
- The message is the **basic entity** that systems use to communicate
- In Camel, a message is **one-way only**: from the sender to the receiver

Camel message, *continued*



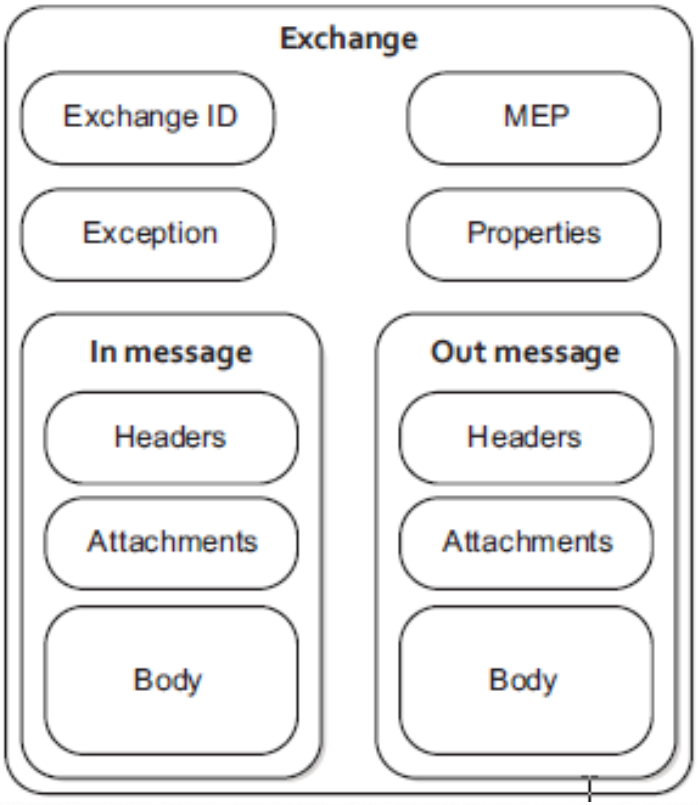
- The Camel message has a **body** (also called a **payload**) where the content of the message is stored
- The **headers** are **variables** attached to the message, stored as key/value pairs
- The message can also embed **attachments** of any type

Camel exchange



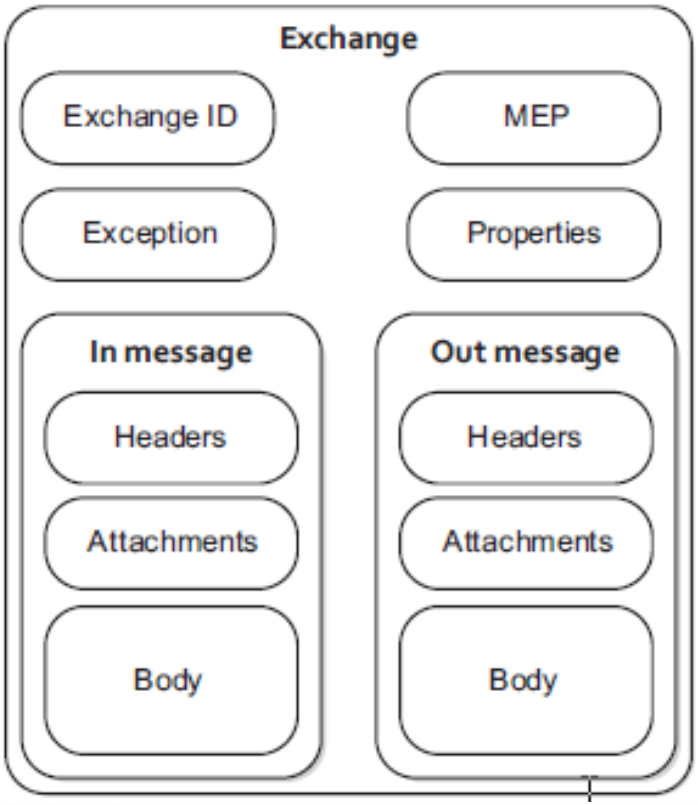
- The exchange is a superior entity, as it **contains one or two messages**
- Because the message is one-way only, the exchange can hold an In message and an Out message, handling **bidirectional communication**

Camel Exchange, *continued*



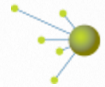
- The unique **exchange ID** is generated by the routing engine
- The **properties** are **variables** attached to the exchange, stored as key/value pairs
- The **Exception** area stores any error or exception message that the exchange might run into while being routed

Camel Exchange, *continued*



- **MEP** stands for **message exchange pattern**; this value defines the number of messages in the exchange
- **By default**, an exchange has only an **In message** and the MEP is set to "**In Only**"
- If bidirectional communication is needed, the MEP can be changed to **"In Out,"** which automatically creates an Out message

Message headers and exchange properties



- Message **headers** and exchange **properties** are variables stored as **key/value pairs**
- Depending on the components you use, Camel automatically creates header or property variables to describe your message
  - For instance, when using a File component, Camel writes header variables such as CamelFileName, CamelFilePath, and CamelFileLenght
- Message headers and exchange properties added by Camel are described in the documentation for each component

Message headers and exchange properties

You can also add or modify your own message headers and exchange properties

- By using the dedicated **Talend components** cSetHeader or cSetProperty
- By accessing the message or exchange **via Java code**

SIMPLE language

- The SIMPLE language is used to easily access exchange and message elements
- The SIMPLE language uses **${*expression*}** placeholders for complex expressions that contain constants and literals
- When run into, a SIMPLE language expression is evaluated and executed on the current message/ exchange in the route

SIMPLE language, *continued*

Using the SIMPLE language, you can access the following variables

Variable	Type	Description

`header.XXX`
`in.header.XXX`
`in.headers.XXX`
	`Object`	Contains the input message header `XXX`

`out.header.XXX` `out.headers.XXX`	`Object`	Contains the output message header `XXX`
`property.XXX`	`Object`	Contains the `Exchange` property `XXX`
`exchangeId`	`String`	Contains the unique ID of the `Exchange`
`sys.XXX` `sysenv.XXX`	`String`	Contains the system environment `XXX`
`exception`	`Object`	Contains the exception on the `Exchange`, if any exists

SIMPLE language examples

- To access the body of your In message:

  ${in.body}

- To access a header variable called fileName in the In message:

  ${in.header.fileName}

- To access a property variable called correlationID:

  ${property.correlationID}

SIMPLE language, *continued*

- The SIMPLE language also offers basic operators and functions
- The complete list is available on the Apache Camel website, along with documentation and examples

Lab overview

Lab 2–Logging / Lab 3–Setting body and headers

In these labs, you will:

- Use the cLog component
- Write a custom log message by using SIMPLE language expressions
- Set a custom body and custom headers

Lesson summary

- ESB routes transport messages stored in exchanges
- The core of the message is stored in the message body
- Headers and properties store additional valuable information
- Message and exchange information can be accessed via SIMPLE language instructions or Java code

Talend ESB Basics

Message Processing with Java Beans

Objectives

At the end of this lesson, you will be able to:

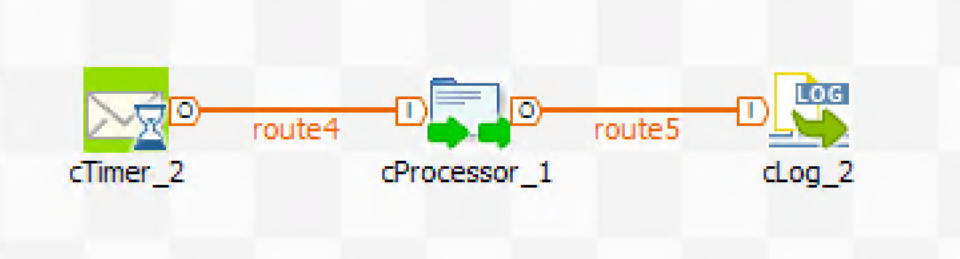
- Access message elements with Java code
- Include Java beans in mediation routes
- Pass a message and its headers as parameters to a bean

Accessing a message with Java

- The current Java instance of an exchange in a route is called an **exchange**
- By invoking the exchange instance, you can access its properties, exception messages, and in/out messages
  - To access the in message and read the body/write in the body
    - exchange.getIn().getBody();
    - exchange.getIn().setBody("Hello World!", String.class);
  - Access a header
    - exchange.getIn().getHeader("myHeader");

Using Java to process a message, *continued*

- If you need to run a few lines of Java to process a message, you can use the cProcessor component
- Java instructions are run each time a message passes through the component



Using Java beans

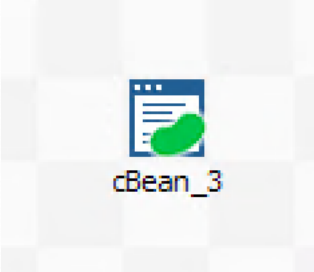
- Java beans are simple Java classes you can call inside a route to process your exchange or message
- Create a Java bean:
  - When you need to run a more complex block of code
  - When you need to create a reusable method

Using Java beans, *continued*

- In the code of a Java bean, it is very easy to pass the exchange, or even some message elements, as parameters to a method

```
public static void updateMessageBody(Exchange exch, @Header("Company") String company) {
    Random randomGenerator = new Random();
    int randomInt = randomGenerator.nextInt(3);
    if (randomInt == 1) {
```

- Java beans can be called by using the Talend component cBean



Lab overview

Lab 4: Using cBeans

In this lab, you will:

- Update Lab 3 to add a new route using a cProcessor
- Write a Java bean to process messages
- Call Java beans in two ways
  - Call a reference
  - Call a new instance

Lesson summary

- Talend ESB allows you to create custom Java code and include it in your mediation routes
- The cProcessor component allows you to easily embed a few lines of code
- You can create reusable code and methods in simple Java classes called beans
- You can easily invoke a bean by using the cBean component

Talend ESB Basics

Rerouting a Message

Objectives

At the end of this lesson, you will be able to:

- Send a message from one route to another
- Use the cDirect component as a producer and consumer

Rerouting messages

- In the Talend Studio, the mediation units you can create are called **Camel contexts**. A Camel context is a container that holds:
  - **Components**, which are used in the container
  - **Processors** and their configurations
  - **Routes**, the sequences of components and processors
  - **Beans**, the classes used in the container
- When the operations team deploys an ESB build, the package it sends to the runtime is the Camel Context

Rerouting within the same context

- A Camel context can have several routes
- We recommended keeping routes short enough to be easily isolated and tested
- Some routes can be accessed by multiple other routes so that their processes can be shared
- In the same Camel context, routes must be able to send their messages to other routes to continue processing

Rerouting within the same context, *continued*

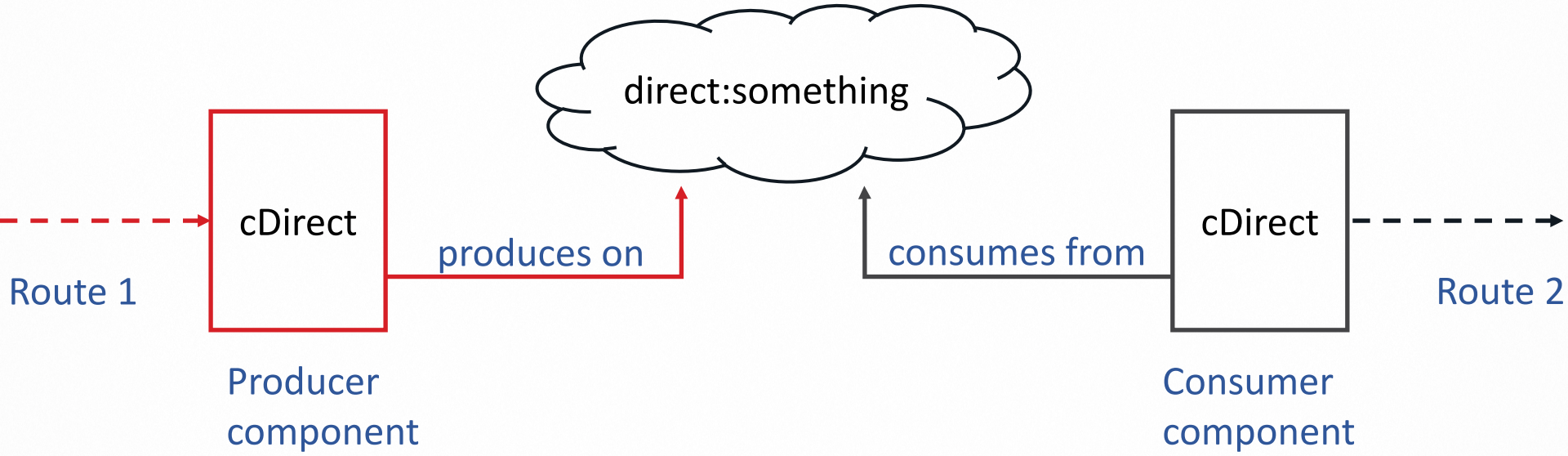
- The Camel direct component is used to reroute a message
- When used as a consumer, the direct component is given a unique URI for its context. The URI is built as follows:
  - **direct:something**
    - **direct** is the mandatory scheme of the URI
    - **something** is the unique name of the direct component
- Once the consumer endpoint is configured, it consumes any message sent to this URI

Rerouting within the same context, *continued*

- When used as a producer, the direct component is configured with the URI to which it must send the message
- Any message reaching the producer direct component is produced (sent) to the URI

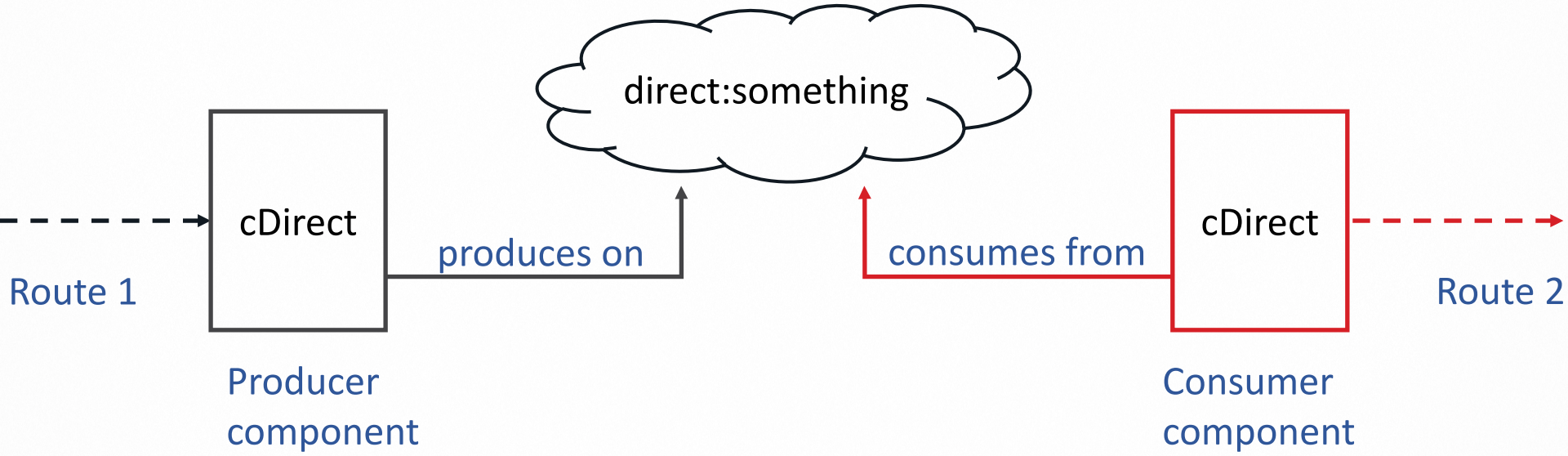
Rerouting within the same context, *continued*

When a message arrives at the end of Route 1, the cDirect component produces (sends) the message to the direct:something URI



Rerouting within the same context, *continued*

As soon as the message is produced on the URI, the cDirect component from Route 2 sees this as an event and consumes it



Rerouting to different Camel contexts

- The direct component (cDirect in the Studio) allows rerouting only within the same context
- If you need to reroute a message to a different Camel context, you can use:
  - The cDirectVM component—same as the cDirect component but can reach outside its own context
  - The SEDA component—sends to a destination outside its context, asynchronous behavior and queuing ability
  - A message broker—more reliable, makes sure the message is distributed and allows both point-to-point messaging and broadcast

Lab overview

Lab 5: Using cDirect

In this lab, you will reroute a message using the cDirect component

Lesson summary

- You can reroute messages within a Camel context by using the cDirect component
- The cDirect component uses a URI to produce and consume messages on it
- The cDirectVM and SEDA components use the same logic but allow you to expose a URI out of the Camel context, thereby allowing rerouting from one context to another

Talend ESB Basics

Enterprise Integration Patterns

Objectives

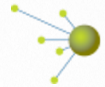
At the end of this lesson, you will be able to:

- Define enterprise integration patterns (EIP)
- Name four common EIPs
- Use EIPs in a mediation route

Enterprise integration patterns

- **Enterprise Integration Patterns** is a book by Gregor Hohpe and Bobby Woolf. It describes 65 design patterns that can be used by enterprise application integration and message-oriented middleware in the form of a pattern language.
- This book is considered the most comprehensive work on application integration.

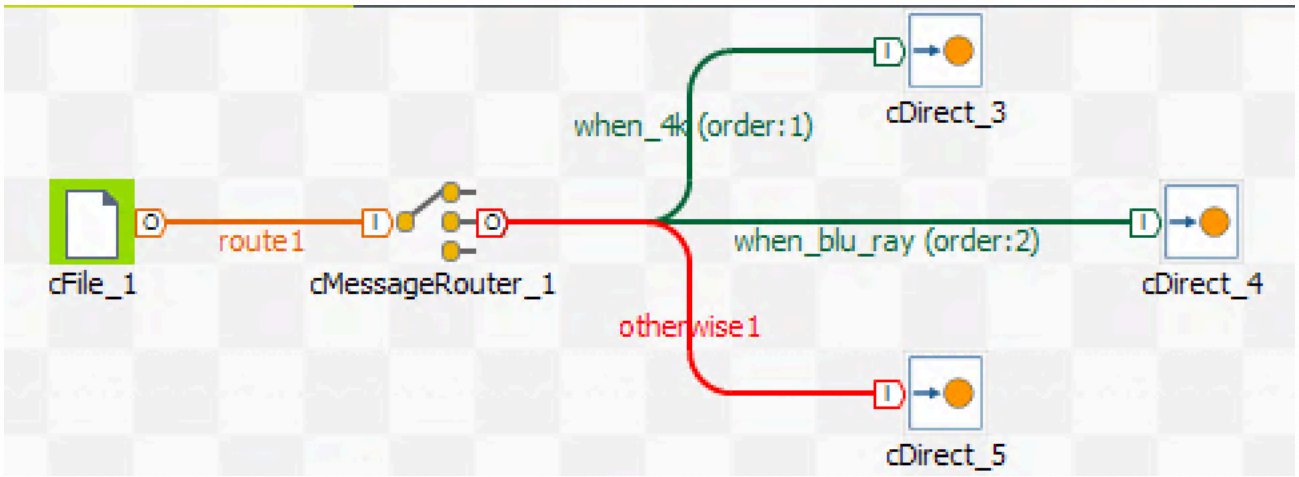
Enterprise integration patterns, *continued*



- Each **pattern** represents an application integration **unit action**
- **EIPs** are implemented as Camel processors.
- They can be found in the palette of the Studio
- Reading the Enterprise Integration Patterns book is strongly recommended as a routing design prerequisite

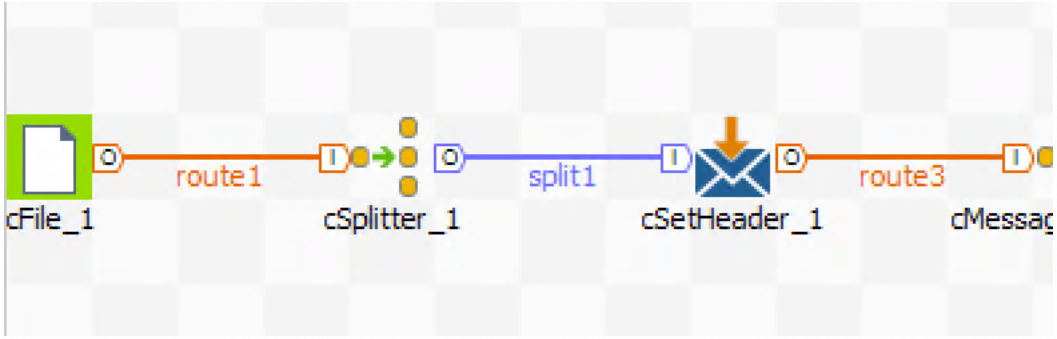
cMessageRouter

This EIP helps you **route** a message to one of several destinations **based on a condition**



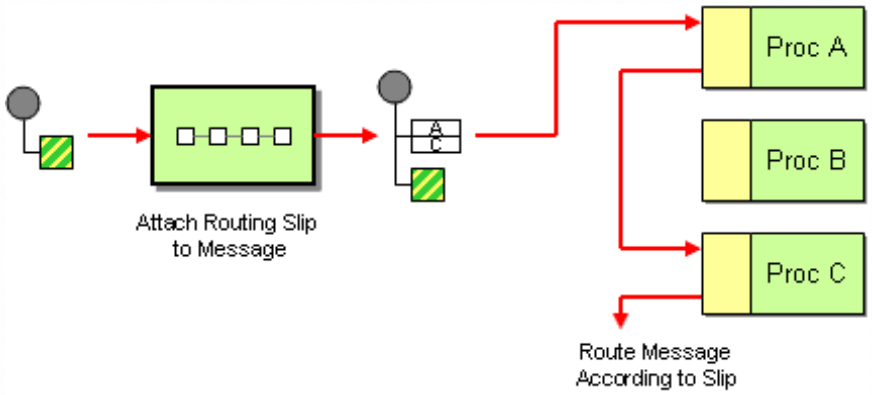
cSplitter

This EIP **splits** an input message into **several messages** based on a defined rule. In this training, you will split an XML message into smaller chunks **according to an Xpath query**.



cRoutingSlip

This EIP reroutes a message through a defined sequence of routes. The sequence can be variable for each message. In the corresponding lab for this EIP, the sequence is defined after an analysis of the message content.



cMulticast

This EIP sends one message in parallel to many destinations

Lab overview

Lab 6: EIP cMessageRouter
Lab 7: EIP cSplitter
Lab 8: EIP cRoutingSplit
Lab 9: EIP cMulticast

In these labs, you will use EIPs to:

- Reroute messages based on conditions, thanks to cMessageRouter
- Split a message into several messages based on an Xpath query with cSplitter
- Route a message to a sequence of processing routes with cRoutingSlip
- Route a message to multiple destinations with cMulticast

Lesson summary

- Enterprise integration patterns bring more design capabilities to mediation routes
- EIPs are available in the palette of Talend Studio
- EIPs can be combined, or combined with other processors, to create richer mediation routes

Talend ESB Basics

Calling Data Integration Jobs in Mediation Routes

Objectives

At the end of this lesson, you will be able to:

- Call DI Jobs in mediation routes
- Configure a DI Job to receive a Camel exchange

Calling data integration Jobs

- Mediation routes are very good at integrating applications, routing, and delivering messages according to your provided set of rules
- However, mediation routes are not made for data transformation or data processing
- If you need to transform data, Talend offers the ability to call a DI Job from a route

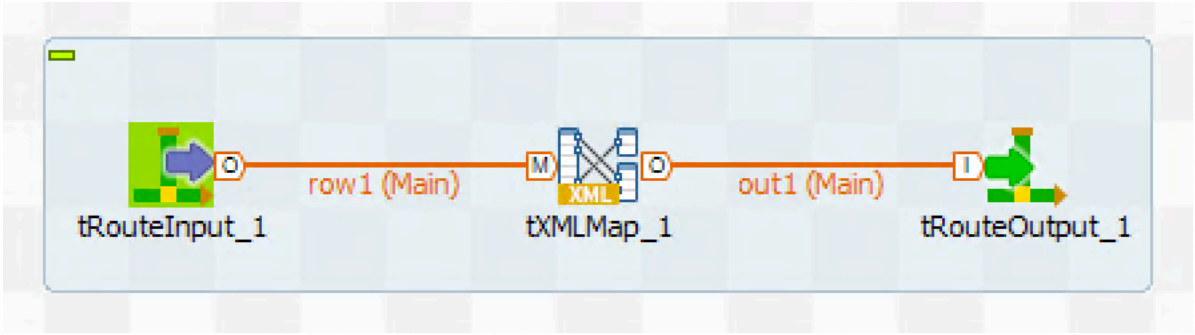
Call a DI Job from a route

- To call a DI Job, use the cTalendJob component
- When the message reaches the cTalendJob component, data is transferred to the job and processed. When the job is finished processing data, it returns the data in a message that goes along the route.

Receive a route message in a DI Job

To enable a DI Job to be used in a route, you need to use **specific components**

- **tRouteInput**—this component gets the exchange from the route and can be configured to get the body, headers, and properties in a DI schema
- **tRouteOuput**—this component builds the output message from the DI Job and can send data in the body, properties, or headers



Lab overview

Lab 10: Calling a DI Job from a route

In this lab, you will:

- Create a route that calls DI Jobs with the cTalendJob component
- Create two DI Jobs that can be called by a route
- Distribute messages to be processed by one of the DI Jobs

Lesson summary

- As part of the Talend Unified Platform, DI Jobs can be used and called from mediation routes with a dedicated component
- To be able to respond to routes, DI Jobs need specific input and output components

Talend ESB Basics

Enterprise Integration Patterns (Advanced)

Objectives

At the end of this lesson, you will be able to:

- Use more-advanced EIPs

cWireTap

This EIP intercepts messages on a route without disrupting the global process. It is used mostly to capture and log messages.

cLoadBalancer

This EIP can implement a distribution strategy to deliver message to several routes. It is used to balance messages between different routes or platforms.

cThrottler

- This EIP makes sure a specific endpoint does not get overloaded, or that you do not exceed an agreed SLA with an external service by limiting the number of messages to be processed in a route in a particular time period.
- All exceeding messages are delayed and enqueued for future processing.

Lab overview

Lab 11: EIP cWireTap
Lab 12: EIP cLoadBalancer
Lab 13: EIP cThrottler

In these labs, you will use EIPs to:

- Intercept a message and reroute it out of the main route without disrupting it
- Load balance the distribution of messages between two routes
- Regulate the number of files processed by a route

Talend ESB Administration

Using a Message Broker

Objectives

At the end of this lesson, you will be able to:

- Define enterprise messaging
- Choose between two messaging strategies
- Send messages from a route to a message broker
- Receive messages from a message broker

ESB messaging

- Talend ESB server embeds a message broker called ActiveMQ. This module is **message-oriented middleware (MOM)**.
- Its only purpose is to handle messages and make sure they are delivered or broadcasted to their recipients.
- ActiveMQ uses standard formats (natively, JMS by default) to make sure messages can be delivered to and understood by any application.

ESB messaging, *continued*

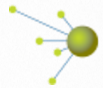
There are two strategies for delivering a message using MOM:

- Point-to-point messaging, using **queues**
- Broadcasting, using **topics**

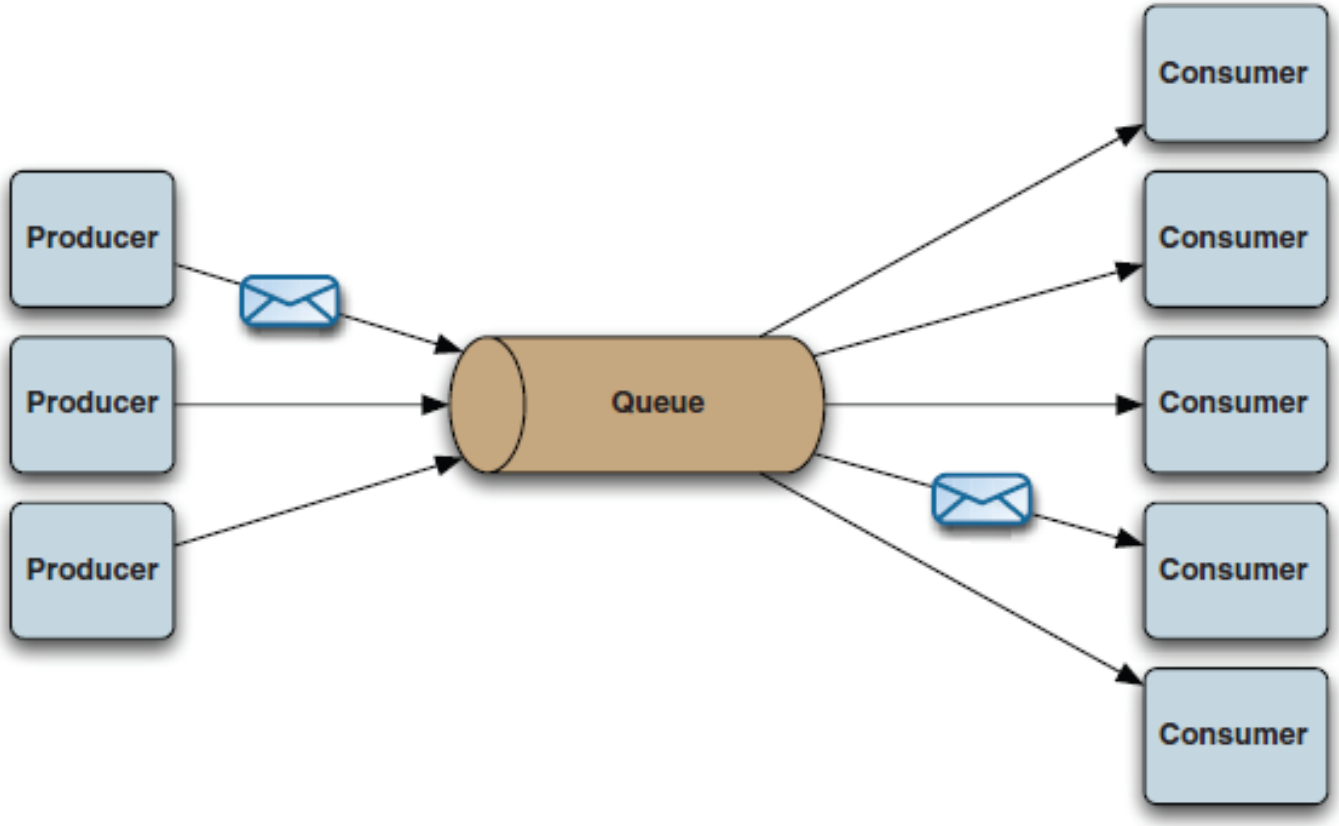
ESB messaging: point-to-point queue

- The sender of the message is called a **producer**.
- The producer sends a message to a queue on the MOM. A queue is a destination with a unique name.
- The recipient of the message is called a **consumer**.
- The consumer is connected to the MOM and listening to one queue.
- When a message is delivered to the queue, the consumer reads it.

ESB messaging: point-to-point queue, *continued*



- When a consumer has read the message, **it is considered consumed and removed from the queue**
- A message sent on a queue **can have only one recipient**

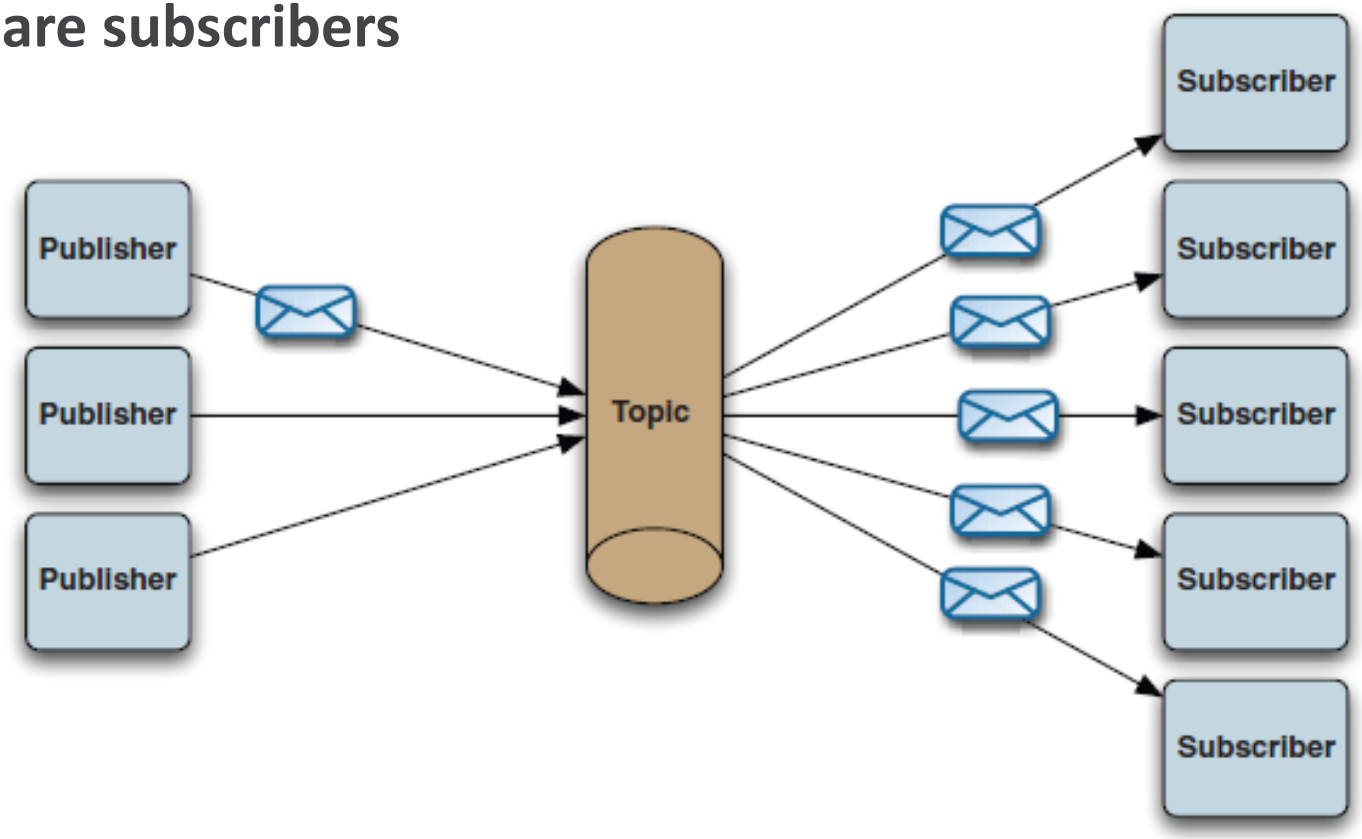


ESB messaging: broadcast topic

- The sender of the message is called a **publisher**.
- The publisher sends a message to a topic on the MOM. A topic is a destination with a unique name.
- The recipient of the message is called a **subscriber**.
- Subscribers are connected to the MOM and listening to the topic.
- When a message is delivered to the topic, **all connected subscribers receive it.**

ESB messaging: broadcast topic, *continued*

- When the subscribers receive the message, **it is removed from the topic**
- A message sent on a queue **can have as many recipients as there are subscribers**



JMS components

- Talend ESB offers a cJMS component to send messages to or receive messages from a message broker
- Like the database connection components in DI Jobs, the ESB palette has a cMOMConnection component that creates a persistent connection to the message broker

Lab overview

Lab 14: Using a message broker

In this lab, you will:

- Connect to the ActiveMQ message broker
- Send messages to queues on the message broker using the cJMS component

Lesson summary

- ActiveMQ can deliver standardized messages to applications by using two strategies: queues and topics
- Talend ESB has components to connect to most message brokers and send/receive messages using the JMS standard format

Talend ESB Basics

SOAP Web Services

Objectives

At the end of this lesson, you will be able to:

- Define SOAP web services
- Explain a standard web service use case
- Create a SOAP web service in Talend Studio
- Test a SOAP web service using a dedicated client
- Create a job to consume a web service

SOAP web services

- SOAP web services expose operations on a server. These operations can be called remotely by sending an XML-formatted query.
- The SOAP protocol usually uses HTTP as the transport layer.
- All SOAP services declare a WSDL (Web Services Definition Language) file. This file defines all available operations, along with their request and response message formats.

SOAP web services: example

A Talend runtime server exposes a SOAP web service on the Internet. This service can place an order in an e-commerce system.

SOAP web services: example, *continued*

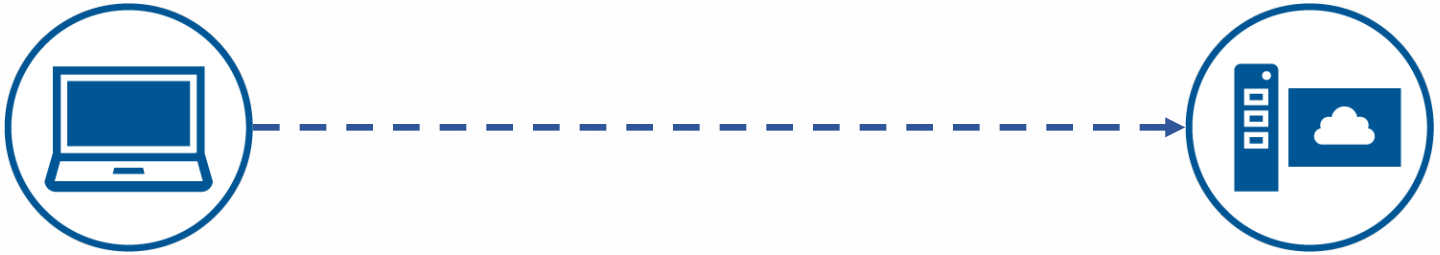
This web service-client machine needs to place an order by sending a request to the SOAP service

SOAP web services: example, *continued*

```
<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:ser="http://www.talend.org/service/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:processOrder>
         <date>2016-10-09</date>
         <time>10:23:58</time>
         <productID>TAL-DE-VR002</productID>
         <quantity>1</quantity>
         <customerID>CU0805</customerID>
      </ser:processOrder>
   </soapenv:Body>
</soapenv:Envelope>
```

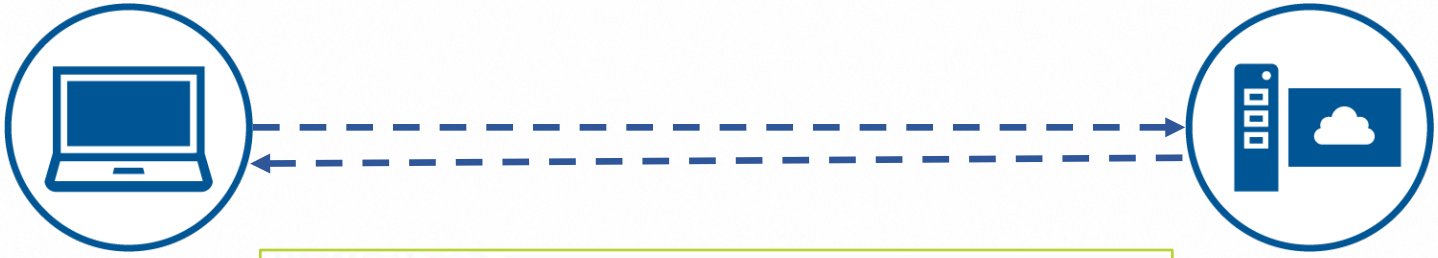
The client sends a web service request message over HTTP to the web service

SOAP web services: example, *continued*



The server processes the request and places the order according to the information contained in the request message

SOAP web services: example, *continued*



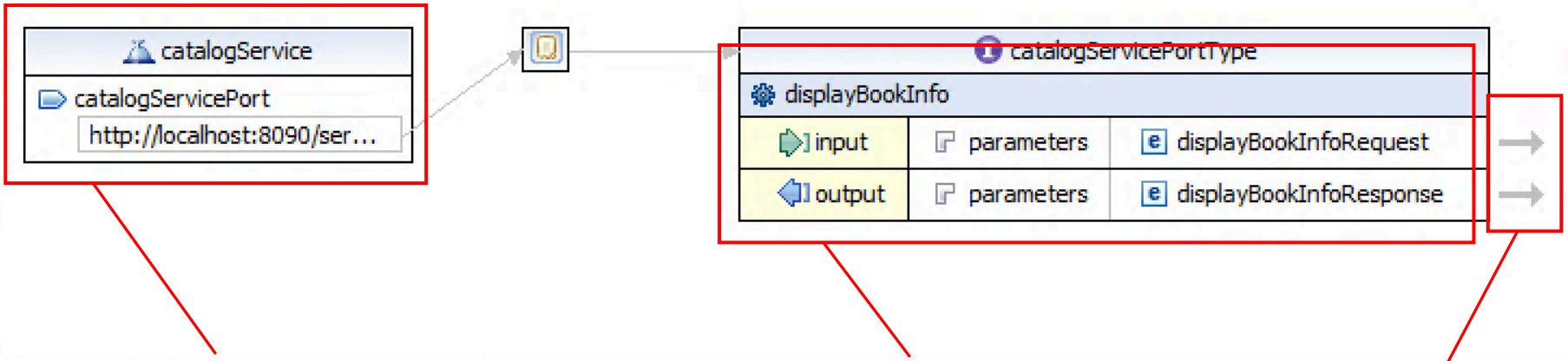
```
<soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <tns:processOrder
          xmlns:tns="http://www.talend.org/service/">
         <response>Created order #1009</response>
         <returnCode>1</returnCode>
      </tns:processOrder>
   </soap:Body>
</soap:Envelope>
```

When the operation is complete, the server sends a response message to the web service client

Creating a SOAP web service

- To create a SOAP web service with Talend Studio, you first need to create a WSDL file.
- The WSDL file is the contract that provides a comprehensive description of your service and its properties:
  - How to access it
  - Where to access it
  - Which operations are available
  - Which formats are expected for the input (request) and the output (response)

Creating a SOAP web service, *continued*



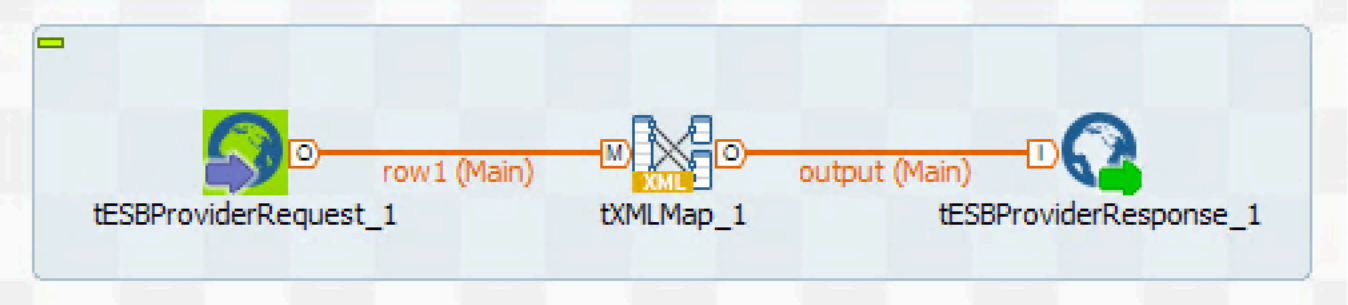
Port information
(service access URI)

Operation definition

Input/output
format definition

Creating a SOAP web service

- Once the service's operations are defined in the WSDL, each operation can be assigned a DI Job
- To be assigned to a service operation, a DI Job needs to start with SOAP web service component tESBProviderRequest and end with tESBProviderResponse



Lab overview

Lab 16: Create a simple SOAP service
Lab 17: Create a SOAP consumer Job
Lab 18: Create a SOAP service to access a database

In this lab, you will:

- Learn how to create a SOAP web service from scratch
- Design a WSDL with the help of the Talend Studio GUI
- Assign a Talend Job to your web service operation
- Create a Job to consume your web service
- Create a second SOAP web service to access a database and display information

Talend ESB Basics

REST Web Services

Objectives

At the end of this lesson, you will be able to:

- Define REST web services
- Explain a standard REST web service use case
- Create a REST web service in Talend Studio
- Create a job to consume a REST web service

REST web services

- REST web services expose a resource. They make the resource available with four basic operations: GET, POST, PUT, and DELETE.
  - GET allows you to SELECT data from the resource
  - POST allows you to INSERT data in the resource
  - PUT allows you to UPDATE data in the resource
  - DELETE allows you to DELETE data in the resource
- REST services are queried simply by calling a URL

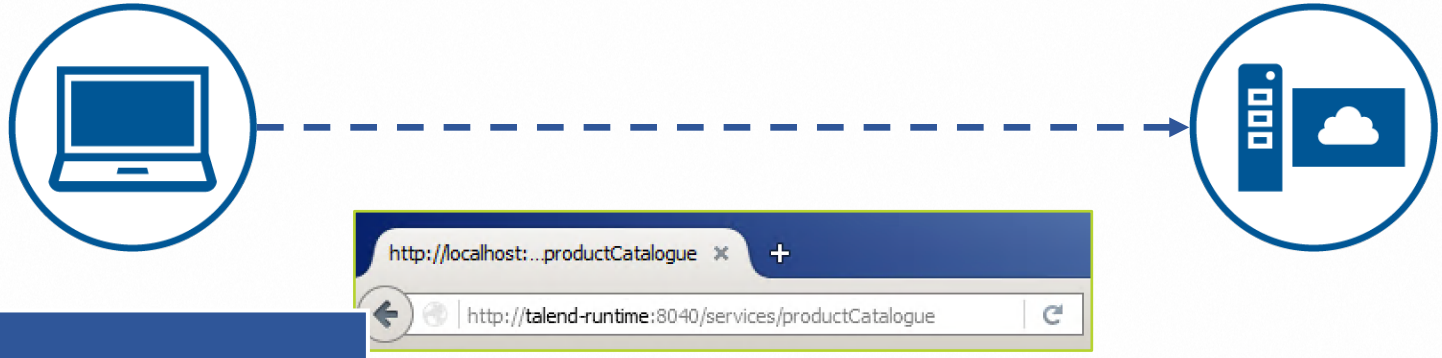
REST web services: example

A Talend runtime server exposes a REST web service on the intranet. This service provides access to a company resource: the product catalog.

REST web services: example, *continued*

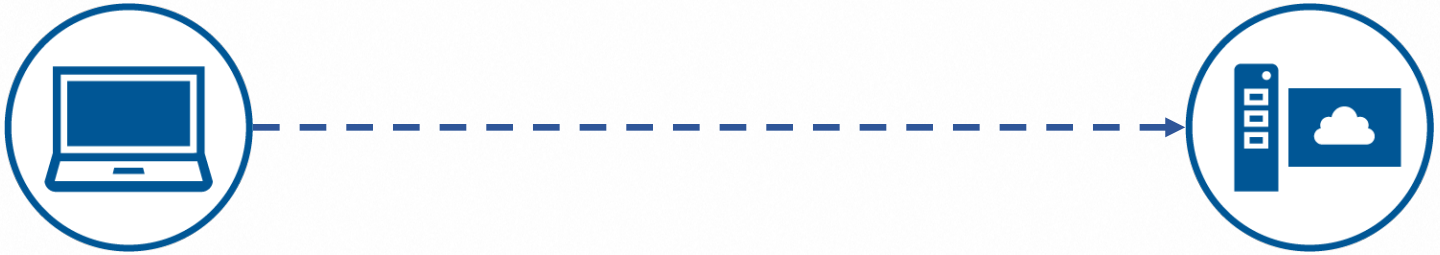
This web service-client machine needs to get product information by querying the REST service.

REST web services: example, *continued*



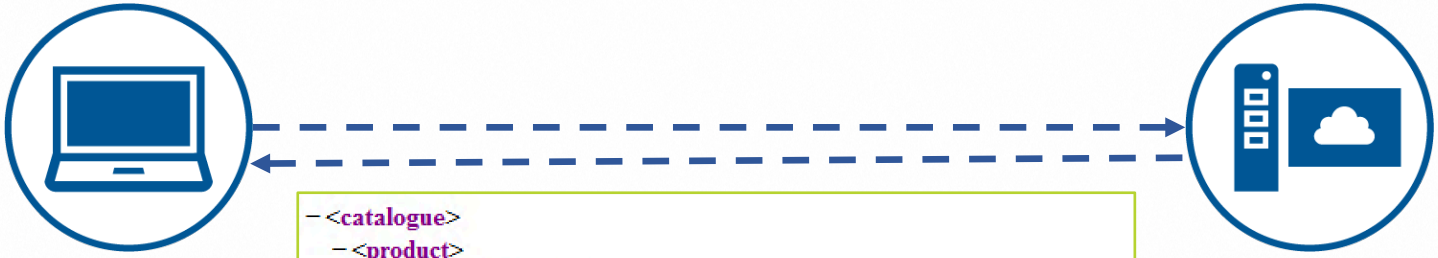
The client sends a web service request via HTTP to the REST web service

REST web services: example, *continued*



The server receives the request and analyzes the requested action. Then it accesses the resource to complete the request.

REST web services: example, *continued*



```
− <catalogue>
  − <product>
      <id>231035933</id>
      <name>Talend Dog T-Shirt</name>
      <publisher>Talend Inc.</publisher>
      <numRating>123</numRating>
      <rating>2.0</rating>
      <icon>dog.png</icon>
      <description>Doggie t-shirt from American Apparel !!</descriptio
      <price>12.0</price>
    </product>
  − <product>
      <id>231035934</id>
      <name>Talend Jr. Spaghetti Tank</name>
      <publisher>Talend Inc.</publisher>
      <numRating>34</numRating>
      <rating>4.0</rating>
      <icon>spaghetti.png</icon>
      <description>Spaghetti tank from American Apparel !</description>
      <price>16.99</price>
    </product>
```

The server sends a response message to the web service client.

Creating a REST web service

- REST web services do not need a contract (as opposed to the WSDL file in the SOAP standard)
- A REST service is accessible via a simple URI and an HTTP verb: GET, POST, PUT, or DELETE

Creating a REST web service, *continued*

A complete REST URI consists of:

- The REST endpoint--the root address on which the service is exposed
- the URI pattern--it completes the REST endpoint to specify which resource to access

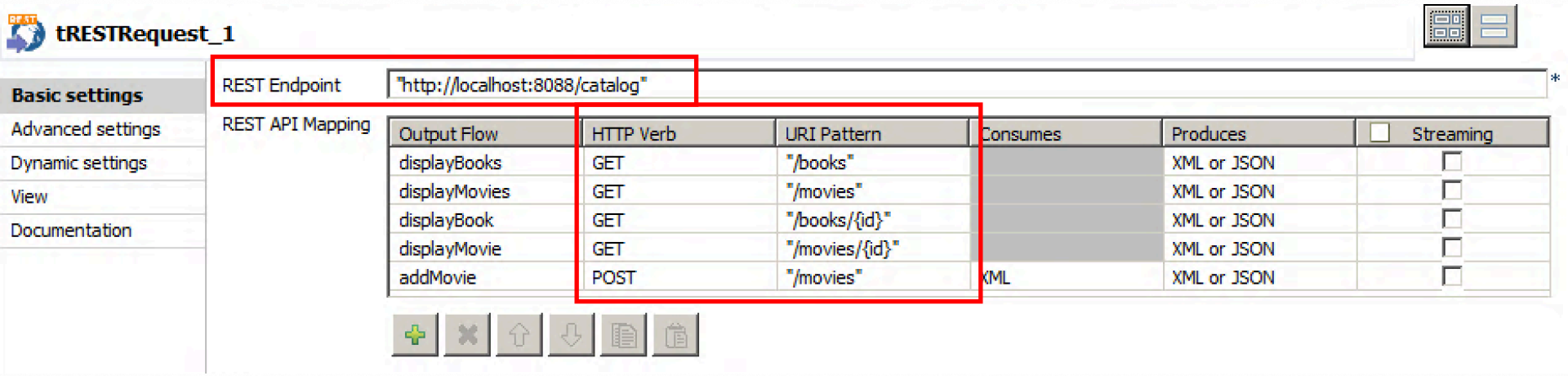
http://localhost:8040/myService/productCatalog

REST endpoint

URI pattern

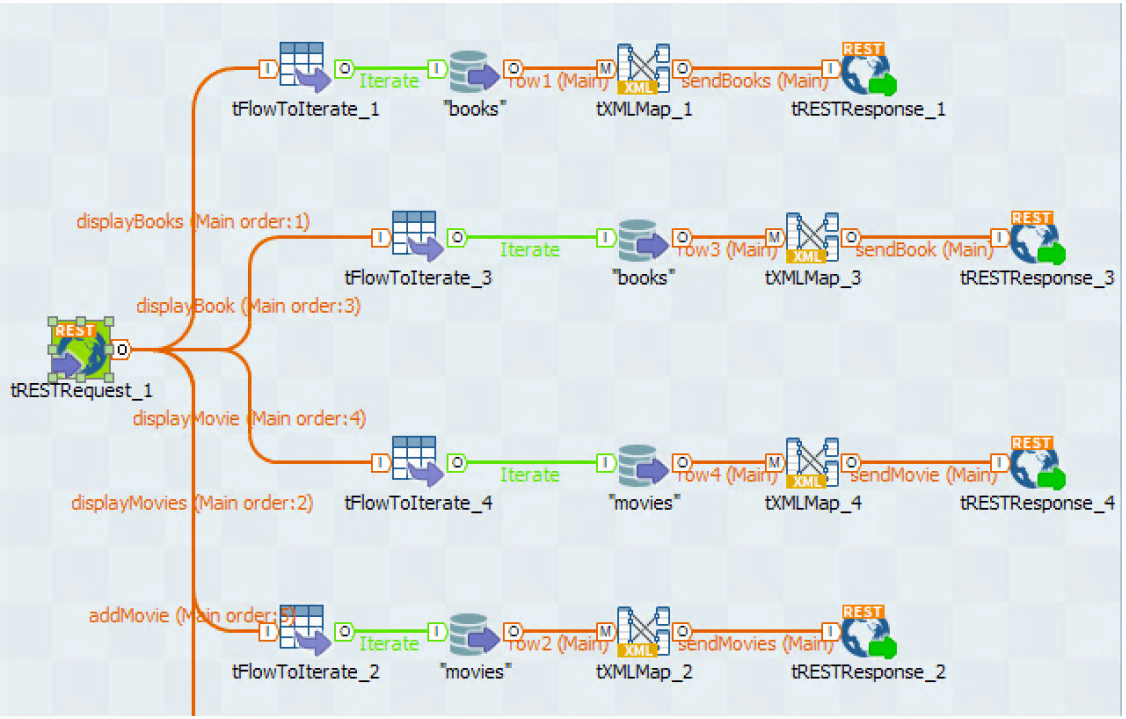
Creating a REST web service, *continued*

- To create a REST web service in Talend Studio, create a Job and designate a tRESTRequest component as the main input component
- This component defines the REST endpoint as well as the available URI patterns and their HTTP verbs



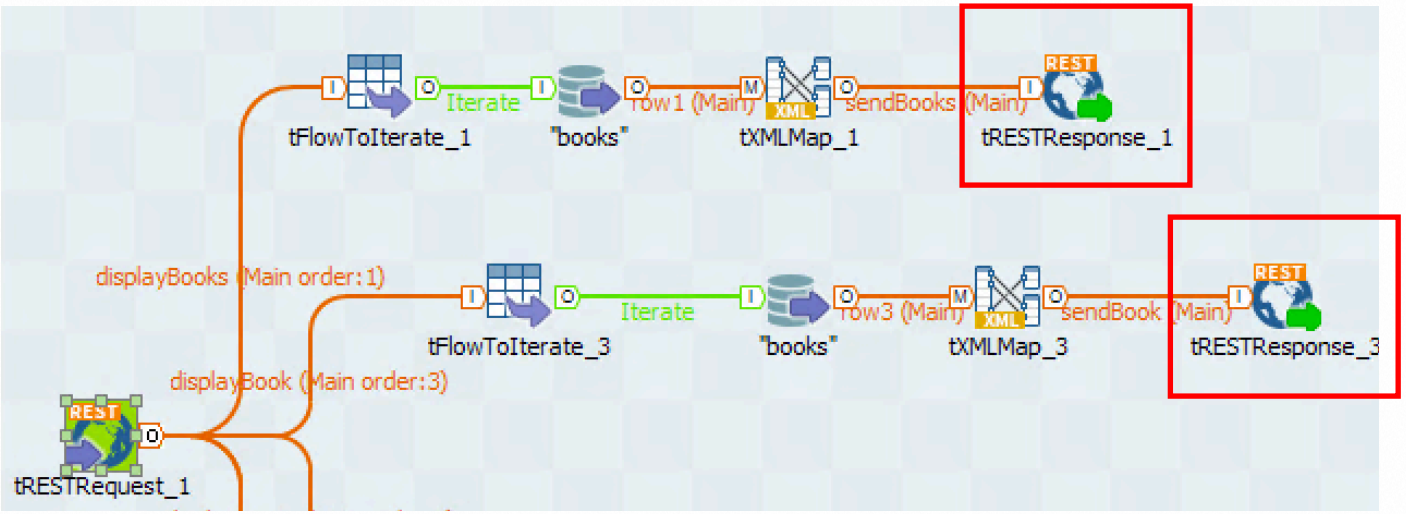
Creating a REST web service, *continued*

Each unique combination of REST endpoint, URI pattern, and HTTP verb results in a new flow leading to its own process



Creating a REST web service, *continued*

The output component for a REST service is the tRESTResponse component, which is in charge of sending the response back to the service consumer



Lab overview

Lab 19 : Create a REST service to read data in a database
Lab 20: Consume a REST service in a DI Job
Lab 21: Create a REST service to write data in a database

In this lab, you will:

- Create REST services to access a database using the HTTP operations GET and POST
- Create a DI Job to consume your REST web services