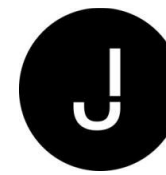


C1

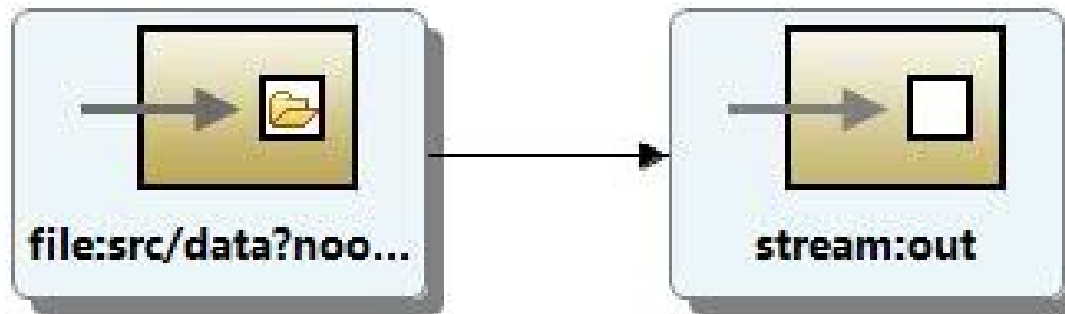
# **APACHE CAMEL**



C1.1

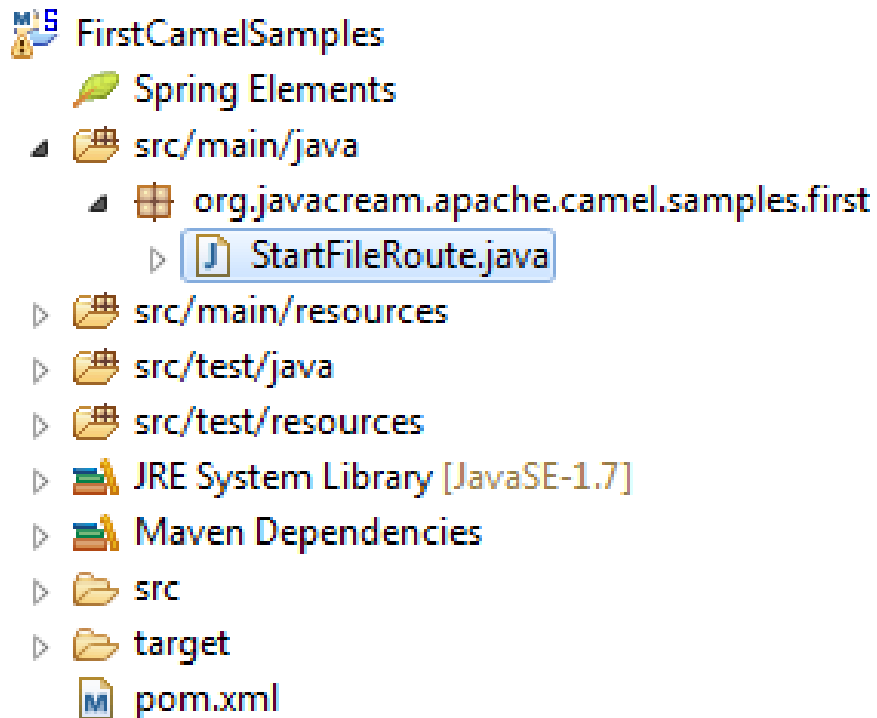
# **EIN ERSTES BEISPIEL**













- Ausgabe eines Dateiinhalts auf die Konsole



```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <route>  
    <from uri="file:src/data?noop=true" />  
    <to uri="stream:out" />  
  </route>  
</camelContext>
```

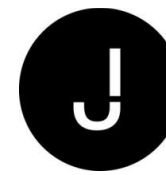
## ■ StartFileRoute - Run As - Java Application



- ▲  org.javacream.camel.first.SpringRouteStarter [4016][Connected]
- ▶  MBeans
- ▲  Camel
  - ▲  camel-1
    - ▲  Endpoints
      - ▲  file
        -  src/data?noop=true
      -  stream
      -  out
    - ▲  Routes
      - ▲  route1
        - ▶  file:src/data?noop=true

```
[main] SpringCamelContext INFO Apache Camel 2.
[main] ManagedManagementStrategy INFO JMX is enabled
[main] DefaultTypeConverter INFO Loaded 183 type
[main] SpringCamelContext INFO AllowUseOrigina
[main] SpringCamelContext INFO StreamCaching :
[main] FileEndpoint INFO Endpoint is cor
[main] FileEndpoint INFO Using default r
[main] SpringCamelContext INFO Route: route1 :
[main] SpringCamelContext INFO Total 1 routes,
[main] SpringCamelContext INFO Apache Camel 2.

<?xml version="1.0" encoding="UTF-8"?>
<person user="james">
  <firstName>James</firstName>
  <lastName>Strachan</lastName>
  <city>London</city>
</person>
<?xml version="1.0" encoding="UTF-8"?>
<person user="hiram">
  <firstName>Hiram</firstName>
  <lastName>Chirino</lastName>
  <city>Tampa</city>
</person>
```



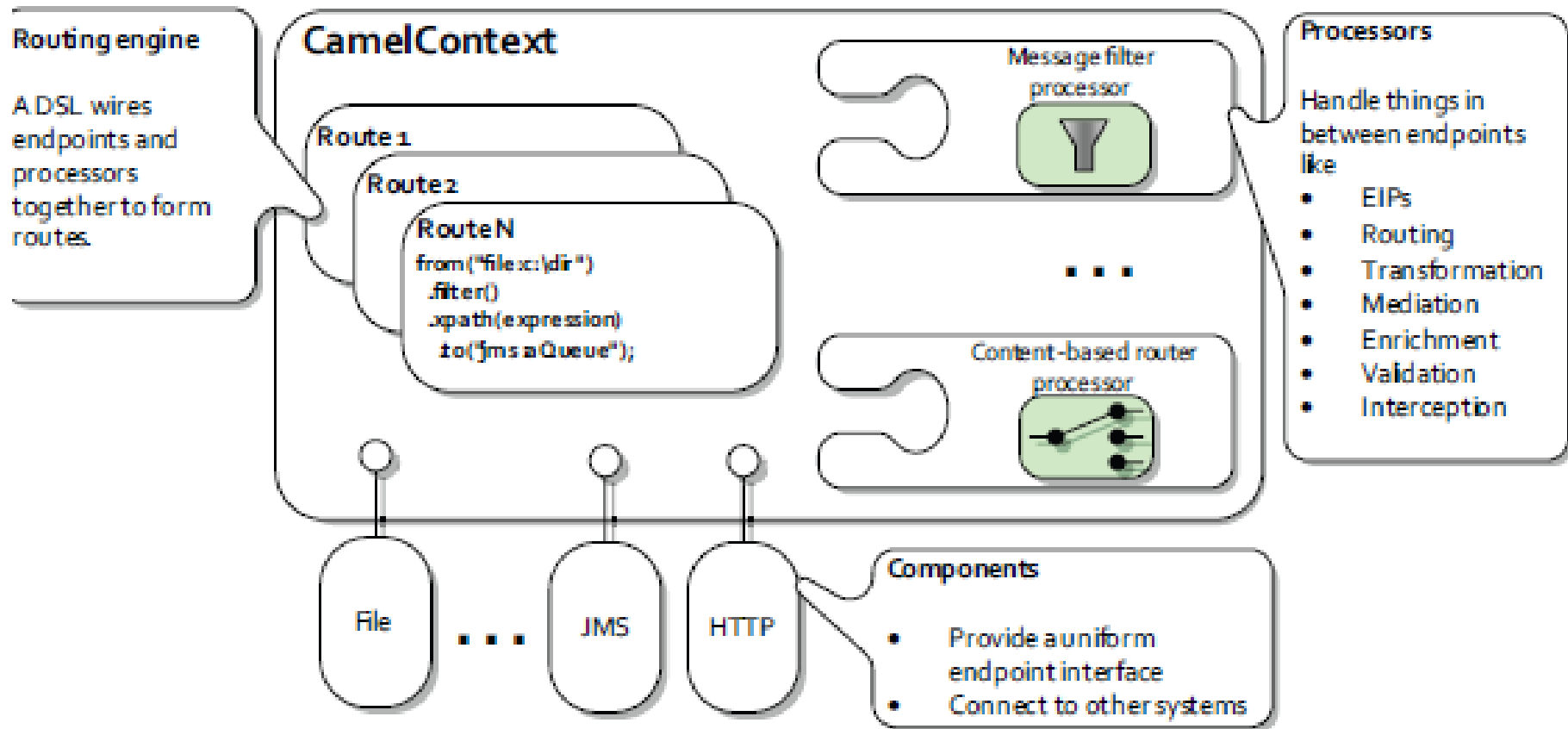
C1.2

# **PRODUKTÜBERSICHT UND ARCHITEKTUR**

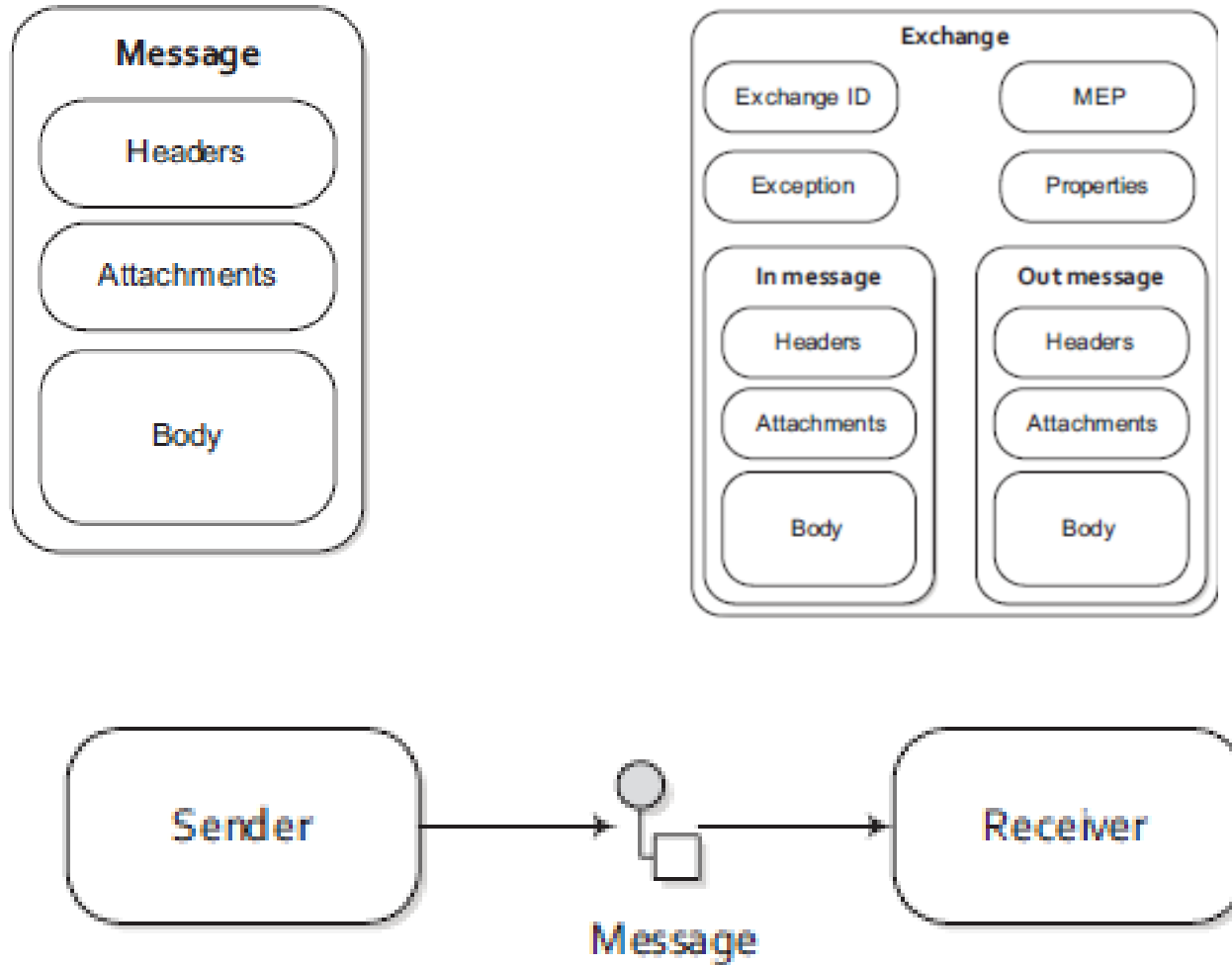


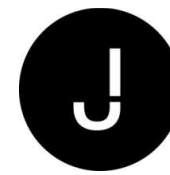
- „ein Open Source Integrations-Framework“
  - Offizielle Beschreibung
- kein vollständiges ESB-Produkt
  - enthält aber die notwendigen Elemente, um einen ESB zu realisieren
- eine Implementierung der Patterns der Enterprise Application Integration
- keine Business Process Engine
- ein Produkt, das von anderen Produkten benutzt wird
  - Apache ServiceMix
  - Apache MQ

- Routing und Mediation
- Domain-Specific Languages (DSLs)
  - Java ist zur Definition von Routen nicht gesetzt
- Implementierung der Enterprise Integration Patterns (EIPs)
  - z. B. Payload-abhängiges Routing
- Modulare Architektur mit leichtgewichtigem Kern
  - Umfangreiche Komponentenbibliothek ist in der Standard-Distribution enthalten
- Konsistentes Programmiermodell
  - POJO-Unterstützung (Plain Old Java Objects)
  - Einfache Konfiguration
- Test Werkzeuge
- Kompetente, aktive Community
  - Kommerzieller Support über Fuse/RedHat



# Message und Exchange





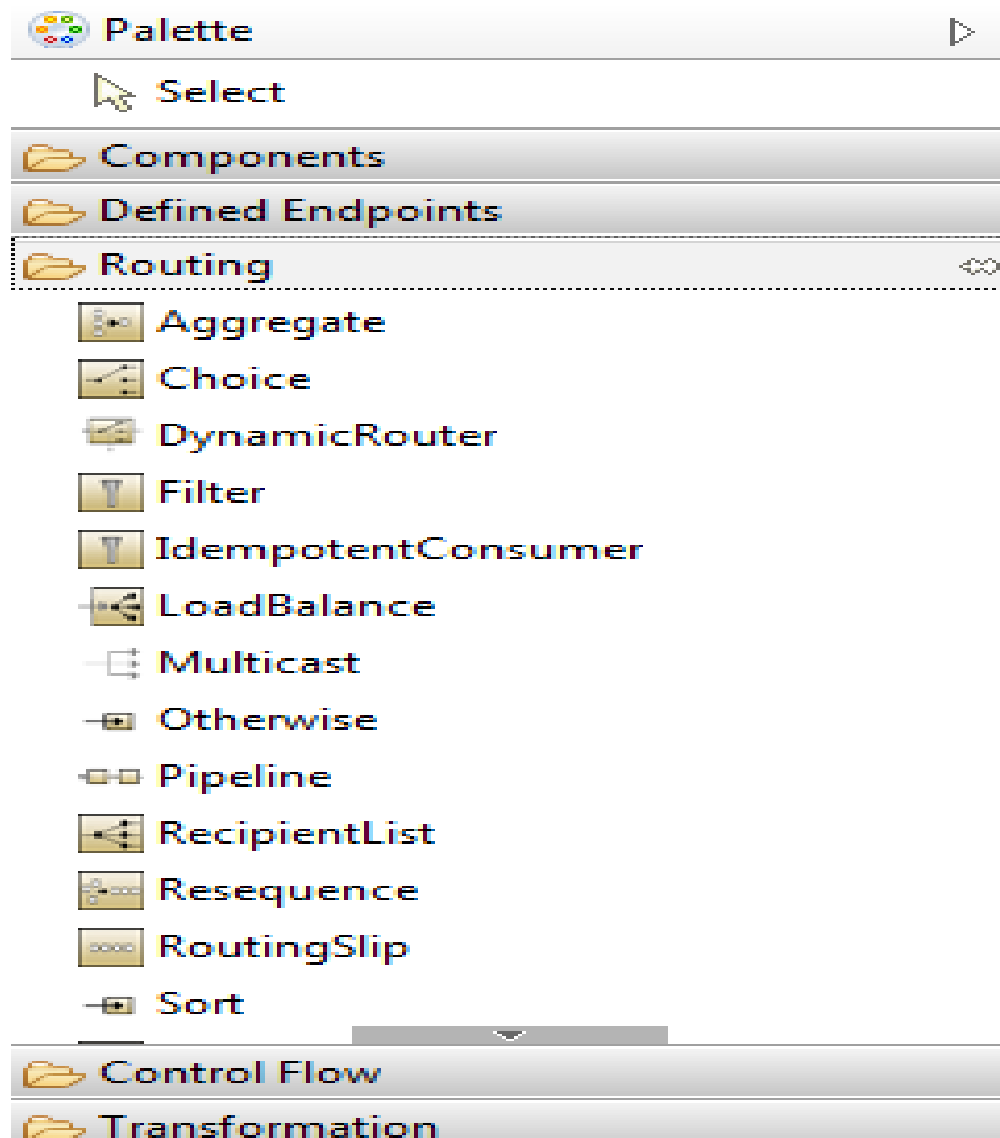
- Eine Sammlung gängiger „Patterns“
  - Eigentlich eine Sammlung von Symbolen mit semantischer Bedeutung
- <http://www.enterpriseintegrationpatterns.com>
- IDE-Unterstützung
  - Palette mit Symbolen für Drag&Drop

# Enterprise Integration Patterns: Eclipse

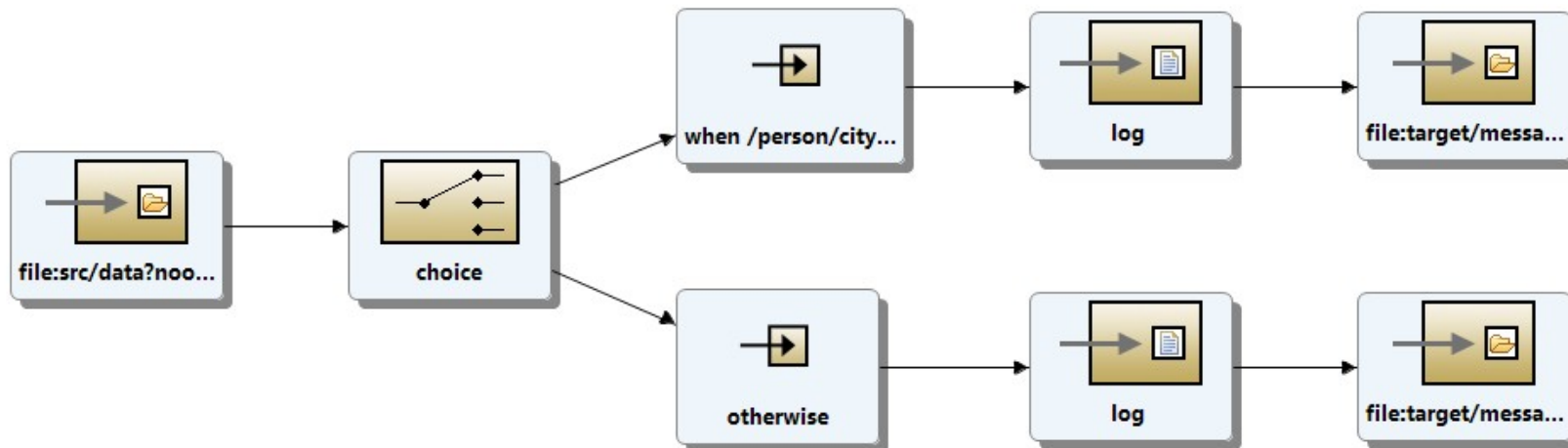


**JAVACREAM**

Training  
Consulting  
Projectmanagement



- Die Engine arbeitet einen gerichteten Grafen ab
- Dieser definiert den Ablauf der Nachrichtenverarbeitung



- Camel unterstützt gängige Programmiersprachen zur Routen-Definition
  - Java
  - Groovy
  - Spring XML
  - Scala
- Die Prinzipielle Arbeitsweise ist für alle Sprachen identisch
  - Definition eines Inbound-Endpoints zum Einstieg in die Route
    - Angabe des zu verwendenden Protokolls
    - Weitere Konfigurationseinstellungen
  - Definition der Routen-Logik
    - Kann für erste Beispiele entfallen
  - Definition eines Outbound-Endpoints zum Ausstieg aus der Route
    - Angabe des zu verwendenden Protokolls
    - Weitere Konfigurationseinstellungen



- Java DSL

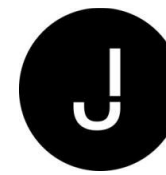
```
from("file:data/inbox").to("jms:queue:order");
```

- Spring DSL

```
<route>  
  <from uri="file:data/inbox"/>  
  <to uri="jms:queue:order"/>  
</route>
```

- Scala DSL

```
from "file:data/inbox" -> "jms:queue:order"
```



C1.3

## **CAMEL RUNTIME**

- Standalone
  - Der Camel-Context wird als Java-Prozess innerhalb einer eigenen Main-Anwendung gestartet
- Embedded
  - Camel wird in einer beliebigen Java-Anwendung benutzt
- Als Anwendung innerhalb eines Applikationsservers
  - Typischerweise als WAR-Datei
  - Vorsicht: Camel öffnet je nach Endpoints eigene Socketverbindungen

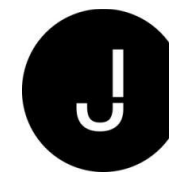
- Der CamelContext wird in einer main-Methode erzeugt

```
import org.apache.camel.main.Main;
public class JavaRouteStarter {
    public static void main(String[] args) throws Exception {
        Main main = new Main();
        main.enableHangupSupport();
        main.addRouteBuilder(new RouteBuilder() {
            @Override
            public void configure() throws Exception {
                //Routen-Definition
            }
        });
        main.run(args);
    }
}
```

- Der CamelContext wird in einer Spring-Konfiguration definiert und durch Erzeugen des Spring-Kontextes erzeugt

```
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringRouteStarter {
    public static void main(String[] args) throws Exception {
        new ClassPathXmlApplicationContext(
            "/META-INF/spring/camel-context.xml");
    }
}
```

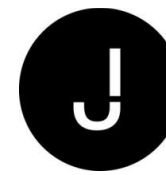


- Hier wird der Camel-Kontext von einem Servlet-Container hochgefahren
- Meistens wird hier ein Servlet-Listener von Spring benutzt, der eine Spring-Konfiguration einlädt
  - Konfiguration in der web.xml
- Zusätzliche Servlets/Listener können beliebig ergänzt werden
  - Beispielsweise das Apache CXF-Servlet für Web Services

```
<web-app ...>
<!-- location of spring xml files -->
<context-param>
<param-name>contextConfigLocation</param-name>
    <param-value>classpath:camel-config.xml</param-
value>
</context-param>

<!-- the listener that kick-starts Spring -->
<listener>
<listener-
class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>

...
```



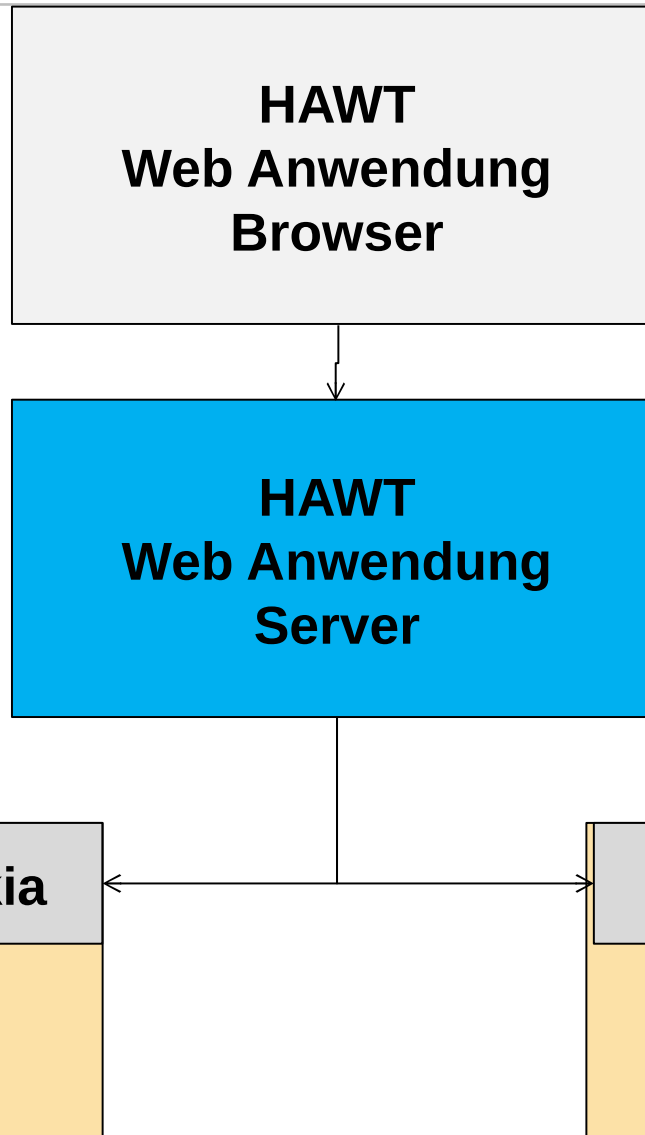
C1.4

# **ÜBERWACHUNG UND ADMINISTRATION**



- Camel ist auf JMX ausgerichtet
  - Administration
    - Zugriff auf Konfigurationseinstellungen
    - Aufruf steuernder Operationen
  - Überwachung
    - Interne Erfassung von Routen-spezifischen Metriken
- Camel ist keine „Integration Suite“
  - Komfortable Web-Konsolen etc. sind nicht (mehr) Bestandteil des Projekts
  - Diese werden jedoch von anderen Herstellern angeboten
    - kommerziell
    - aber auch Open Source

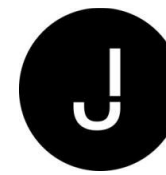
- Projekt der JBoss –Community
  - Java-basiert
- Allgemeines Überwachungswerkzeug mit PlugIns für verschiedene Produkte
  - unter anderem natürlich Camel
- Installation als Web Archiv in einem Web Server
- Der Zugriff auf Metrik-Informationen erfolgt über Jolokia
  - Jolokia selber ist eine von Camel völlig unabhängige Anwendung, die JMX-Informationen über eine REST-Schnittstelle zur Verfügung stellt





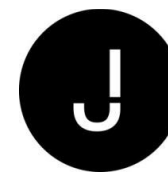
The screenshot shows the Hawtio web console interface. At the top, there's a navigation bar with tabs: ActiveMQ, Camel (selected), Connect, Dashboard, Health, JMX, Logs, Threads, and Tomcat. Below the navigation bar, there's a search bar labeled "Filter...". On the left side, there's a tree view of Camel contexts. Under "Camel Contexts", there's a sub-entry "camel-1", which contains "JavacreamCamelRoutes". Under "JavacreamCamelRoutes", there's a sub-entry "Routes", which contains "file\_to\_stream". Under "file\_to\_stream", there's a sub-entry "file:c:/\_training\_apache\_carr", which contains "to1". On the right side, there's a panel for the selected context. It has tabs: Attributes (selected), Operations, and Chart. Below the tabs, there are buttons: Start, Pause, and Destroy. Below the buttons, there's a table with two columns: Property and Value. The table contains the following rows:

Property	Value
Allow use original message	true
Application context class name	WebappClassLoader context: javacream-cam
Camel	JavacreamCamelRoutes
Camel version	2.15.1.redhat-620133
Class resolver	org.apache.camel.impl.DefaultClassResolver
Delta processing time	-8



C2

# **PROGRAMMIERUNG**



C2.1

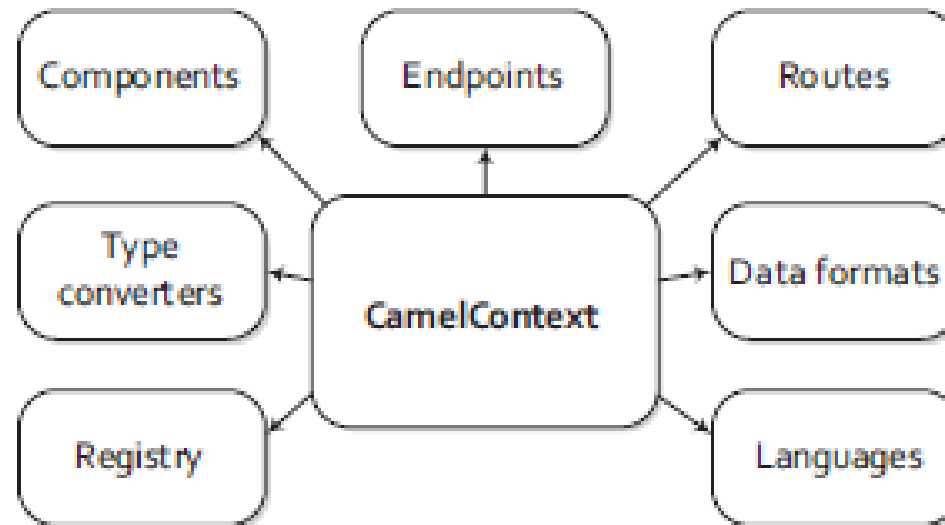
## **ELEMENTE**

- Endpoints
  - Definieren die Endpunkte einer Route, über die Nachrichten ausgetauscht werden
- Processors
  - Eine Route besteht aus einzelnen Processors
    - Die untereinander Nachrichten austauschen
  - Processor ist ein abstraktes Konzept
- Komponenten
  - Komponenten definieren die konkrete Ausprägung eines Verhaltens
    - Endpoints
    - Processor-Implementierungen
      - Nachrichtenverarbeitung
      - Ablaufsteuerung
      - Transformationen
- Routen
  - Definieren einen Ablauf einer Nachrichtenverarbeitung

- Routen
  - verbinden Endpoints
    - Ein eingehender Endpoint wird mit einem Ausgang verknüpft
  - definieren den Pfad, den ein Exchange während der Ausführung der Route durchläuft
  - sind konfigurierbar
  - definieren einen Ausführungskontext
  - werden von einem Entwickler konkret programmiert
    - Routen sind nicht nur einfache „Palette-Drag&Drop“-Diagramme!
    - Damit sind für Routen alle Richtlinien der Software-Qualität zu berücksichtigen
      - Versionierung
      - Dokumentation
      - DRY & KISS
        - „Don't Repeat Yourself“, „Keep it simple“



- Dieser hält alles zusammen



- Realisierung
  - Simple HashMap
  - Spring Context
  - OSGi Service Registry

- Elementare Typen sind
  - CamelContext und DefaultCamelContext
  - RouteBuilder
  - Processor-Interface
    - Ein Processor verarbeitet Nachrichten
  - Predicate-Interface
    - Ein Predicate prüft Bedingungen und steuert damit den Ablauf der Route
  - Component-Implementierungen
    - Konkrete Endpoints
    - Transformer
    - EIP-Implementierungen wie Splitter und Aggregator
- Der RouteBuilder stellt ein fluentes API zur Verfügung, um programmatisch den Graphen der Route zu definieren

```
CamelContext context = new DefaultCamelContext();
ConnectionFactory connectionFactory =
    new
ActiveMQConnectionFactory("vm://localhost");
context.addComponent("jms",
    JmsComponent.jmsComponentAutoAcknowledge(connectionFactor
y));
context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("ftp://orders.com/?username=u&password=pwd").
            to("jms:incomingOrders");
    }
});
context.start();
```

- Erweiterung der Spring-Konfiguration um den camel-  
Namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">
...
<camelContext
xmlns="http://camel.apache.org/schema/spring"/>
</beans>
```

Die Route wird innerhalb des CamelContexts als hierarchischer Graph definiert

```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <route>  
    <from  
      uri="ftp://orders.com/?username=u&password=pwd"/>  
    <to uri="jms:incomingOrders"/>  
  </route>  
</camelContext>
```

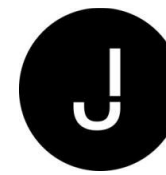
- RouteBuilder-Implementierungen werden innerhalb eines Spring-Contextes definiert

- Statisch

```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <routeBuilder ref="ftpToJmsRoute"/>  
</camelContext>
```

- Dynamisch

```
<camelContext  
  xmlns="http://camel.apache.org/schema/spring">  
  <packageScan>  
    <package>org.javacream.routes</package>  
  </packageScan>  
</camelContext>
```



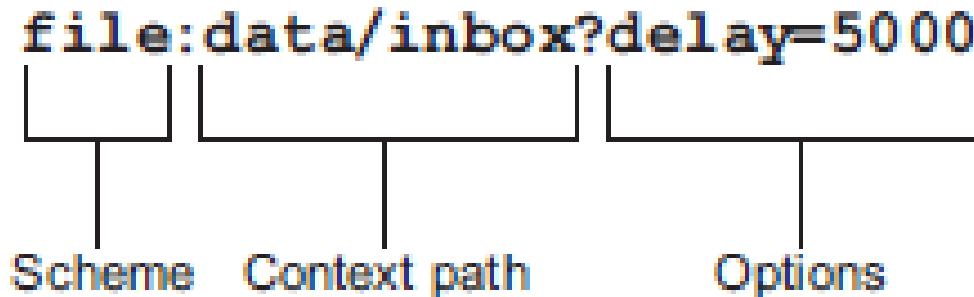
C2.2

## **ENDPOINTS**

- Apache Camel enthält einen ganzen Satz von fertigen Endpoints
  - Eigene Komponenten können bei Bedarf implementiert werden
- Endpoints definieren die Einstiegs- und Ausstiegspunkte der Routen
- Jede Komponenten-Implementierung definiert einen Satz von Eigenschaften
  - Aus der Sicht eines Programmiers definieren die Komponenten ein Programmier-API
  - Dieses wird mit den von Camel zur Verfügung gestellten Sprachen benutzt
  - Convention over Configuration
    - Nicht speziell definierte Werte werden mit einem „Reasonable Default“ vorbelegt
- Endpoints erzeugen oder konsumieren Exchanges
  - Dabei werden Endpoint-spezifische Header gesetzt
    - file-Endpoint mit Property „CamelFileName“
  - Namen der Properties sind der Endpoint-Dokumentation zu entnehmen



- Eine Übersicht gebräuchlicher Camel-Komponenten umfasst:
  - file
  - ftp/sftp
  - jms
  - jdbc
  - Web Service
    - soap
    - rest
  - bean
    - Aufruf einer Spring-Bean-Methode
  - direct
    - Aufruf einer Route innerhalb des selben Kontextes
- Aktuell Liste Bestandteil der Online-Dokumentation

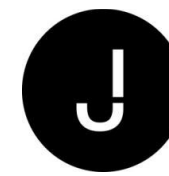


- Das Schema identifiziert die zu verwendende Komponente
  - Diese wird über konfiguriert über
    - den Context Path und
    - die Optionen

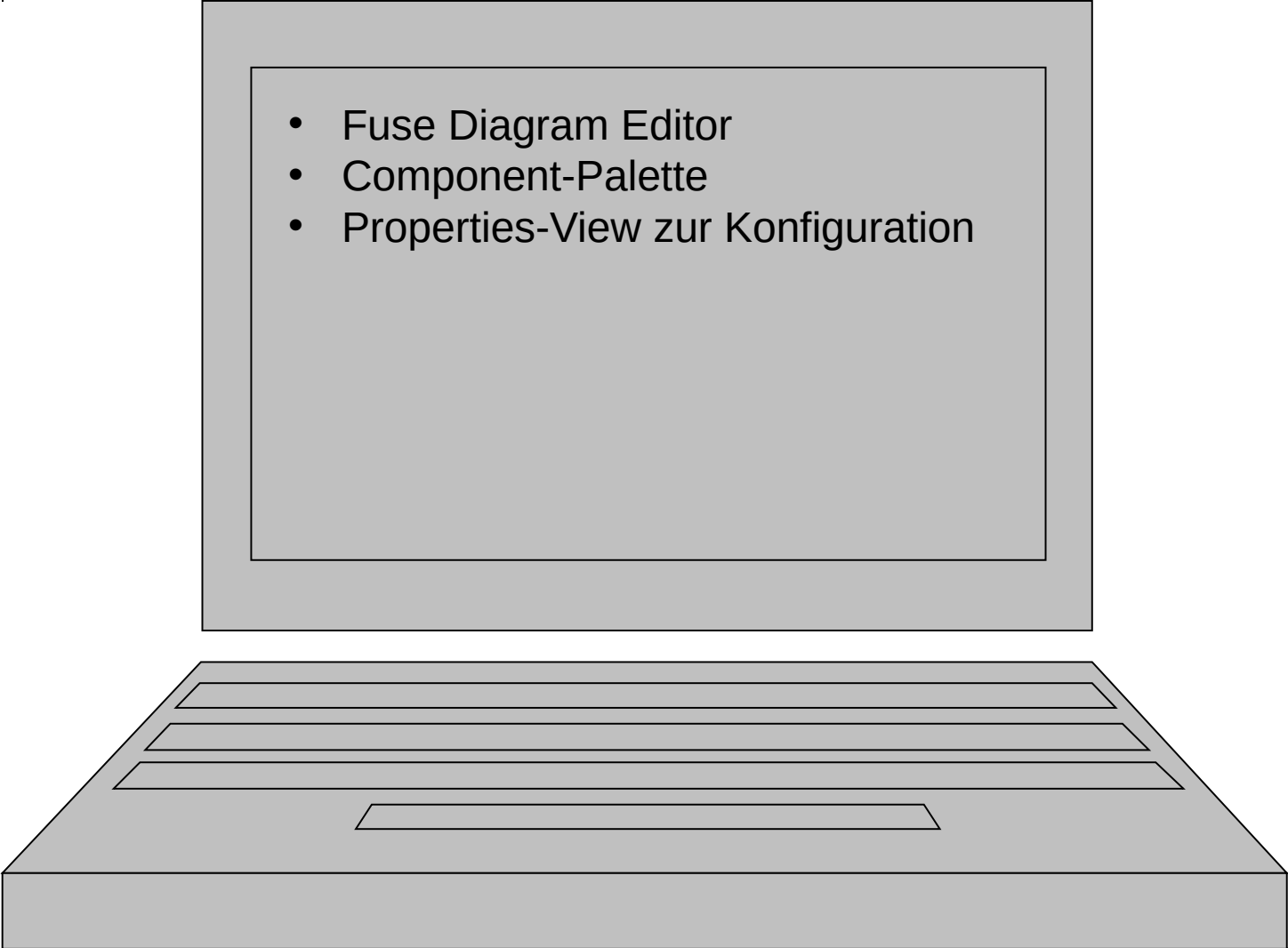
```
//Erzeugen einer ConnectionFactory
ConnectionFactory connectionFactory =
    new
    ActiveMQConnectionFactory("vm://localhost");
context.addComponent("jms",

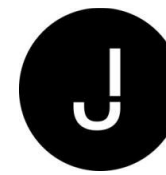
//Registrieren unter der jms-URI
JmsComponent.jmsComponentAutoAcknowledge(connectionFac
tory));
```

- Zur Bereitstellung von WebServices-Endpoints benutzt Camel die CXF-Komponente
  - Apache CXF ist ein von Camel unabhängiges Projekt, das eine Web Services Runtime implementiert
- Dabei implementiert die Route den Web Service in verschiedenen Ausprägungen
  - Der Outbound Endpoint wird als Web Service zur Verfügung gestellt
    - Camel realisiert damit Web Services Fassaden
  - Bridge zwischen Web Services
    - Proxy-Server aus Firewall-Gründen
    - Realisierung zentraler Dienste wie Logging, Auditing, Authentifizierung
  - Die Route implementiert durch Filter/Transformationen eine echte Geschäftslogik
    - Vorsicht: Dies ist sicherlich nicht der eigentliche Sinn einer Route!



- ein Producer erzeugt einen Exchange
- ein Consumer konsumiert einen Exchange
  - Event Driven
  - Polling
- Endpoints sind häufig in beiden Ausprägungen vorhanden
  - Der File Endpoint als Consumer liest Dateien
  - Der File Endpoint als Producer schreibt Dateien

- 
- Fuse Diagram Editor
  - Component-Palette
  - Properties-View zur Konfiguration



C2.3

## **PROCESSORS**

- Ein eigener Prozessor implementiert die Camel\_Schnittstelle Processor
- Der Prozessor hat dabei Zugriff auf das Exchange-Objekt
  - Properties
  - CamelContext
  - In- und Out-Message
    - Out-Message nur für synchrone Routen!
  - Messages haben
    - Header
    - Body
    - Attachment
- Prozessoren
  - werden erzeugt
  - der Routen-Definition eingetragen
  - können beliebig verkettet werden



- Definition in XML

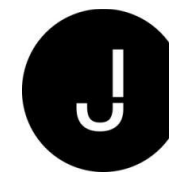
```
<camelContext id="JavacreamCamelRoutes"
xmlns="http://camel.apache.org/schema/spring">
<route id="file_to_stream">
<from uri="file:src/data?noop=true" />
<process ref="simpleProcessor" />
<to uri="stream:out" />
</route>
</camelContext>
```

- Definition in Java

```
from("file:src/data?noop=true").process(new  
SimpleProcessor()).to("stream:out");
```



# Beispiel für einen Prozessor



```
public class SimpleProcessor implements Processor {  
  
    @Override  
    public void process(Exchange exchange) throws Exception {  
        System.out.println(exchange.getProperties());  
        System.out.println(exchange.getExchangeId());  
        System.out.println(exchange.getFromRouteId());  
        System.out.println(exchange.getFromEndpoint());  
        System.out.println(exchange.getPattern());  
        System.out.println(exchange.getIn().getHeaders());  
        System.out.println(exchange.getIn().getBody());  
        System.out.println(exchange.getOut().getHeaders());  
        System.out.println(exchange.getOut().getBody());  
    }  
}
```

- Prozessoren können jederzeit den aktuellen Exchange ändern
  - und damit den Ablauf der Route beeinflussen

```
@Override
```

```
public void process(Exchange exchange) throws Exception {  
    exchange.setProperty("NEW HEADER", "Header-Value");  
    Message message = new DefaultMessage();  
    message.setBody("NEW BODY");  
    exchange.setIn(message);  
}
```

- Für Standard-Aufgaben zur Exchange-Manipulation können bereits fertige Implementierungen der Camel-Distribution benutzt werden
  - Details der Arbeitsweise sind in der Camel-Dokumentation

## Transformation

 ConvertBody

 Enrich

 InOnly

 InOut

 Marshal

 PollEnrich


 RemoveHeader

 RemoveHeaders

 RemoveProperties

 RemoveProperty

 SetBody

 SetExchangePattern

 SetFaultBody

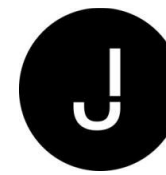
 SetHeader

 SetOutHeader

 SetProperty

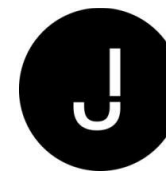
 Transform

 Unmarshal



C3

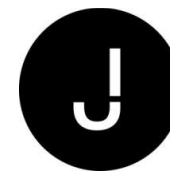
# **SPEZIELLE PROCESSORS**



C3.1

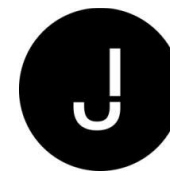
# **EXPRESSIONS**

- Für die Implementierung von Logik in den Routen stellt Camel praktisch alle Sprachen zur Verfügung, die irgendwie in einer Java Runtime unterstützt werden können
  - Java selber
  - XML-Sprachen wie XPath, XQuery
  - Skript-Sprachen
- Allen Sprachen sind bestimmte Sprachfeatures gemeinsam
  - Zugriff auf das Exchange-Objekt innerhalb einer Route
    - Java-Methoden `body()` oder `header()`
  - Zugriff auf den `CamelContext`
    - und damit auf die gesamte Laufzeitumgebung
    - Modifikation des Contexts zur Laufzeit
  - Übersicht
    - <http://camel.apache.org/simple.html>

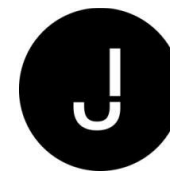


- Bean Language
  - Zugriff auf Java-Objekte und Methoden
- Constant
- Unified Expression Language
- Header
- JsonPath
- XPath
- Mvel
- OGNL
- Ref Language
- ExchangeProperty / Property





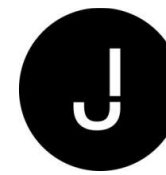
- Skript-Sprachen wie
  - BeanShell
  - JavaScript
  - Groovy
  - Python
  - PHP
  - Ruby
- Simple
  - File Language
- Spring Expression Language
- SQL
- Tokenizer
- XPath
- XQuery
- VTD-XML



- Eine (ursprünglich) einfache Skript-Sprache
- ANT-ähnliche Syntax mit Ausdrücken der Form
  - `${bean.property}`
  - `${bean.method}`
- `simple` unterstützt
  - Literale
  - Arithmetische und logische Operatoren
  - String-Operationen
  - Array-Operationen
  - Einige vordefinierte Funktionen

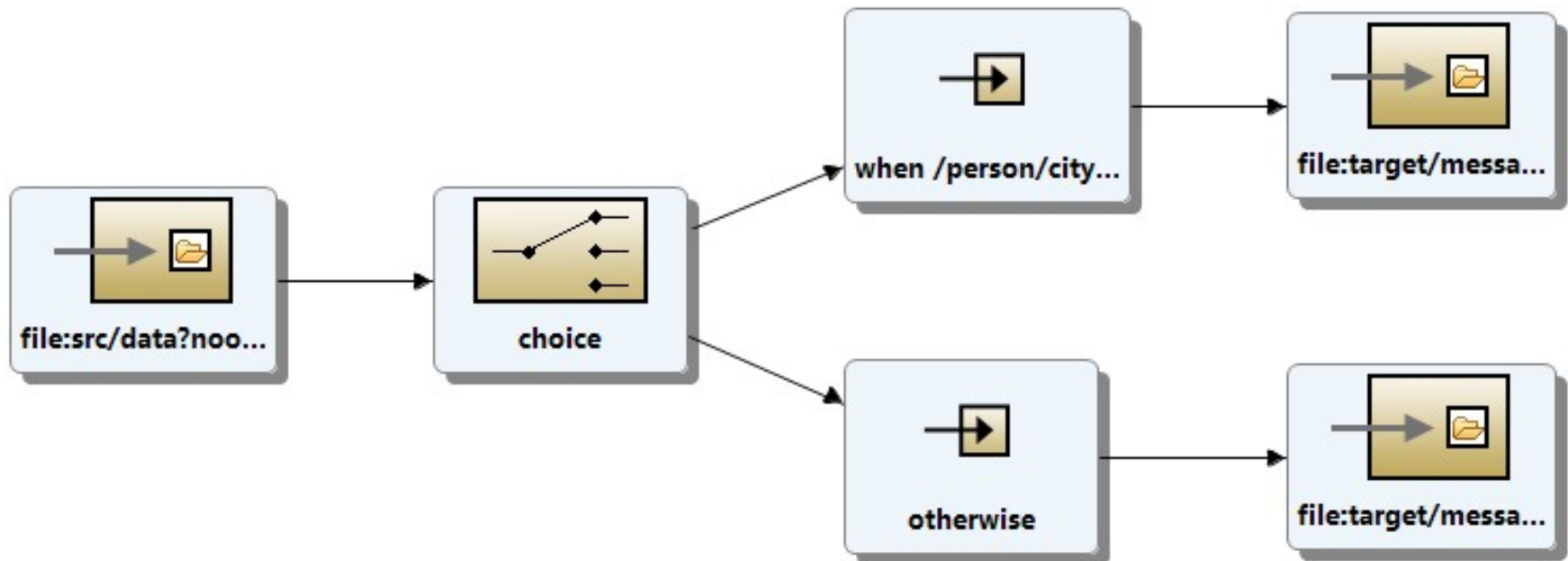
- Mit bean wird eine Methode einer Spring-Bean aufgerufen

```
<bean ref="simpleSpringBean"
method="executeMyLogic"></bean>
```
- Diese Methode kann bestimmte Parametertypen deklarieren, die von Camel übergeben werden
  - `org.apache.camel.Exchange`
  - `org.apache.camel.Message`
  - `org.apache.camel.CamelContext`
  - `org.apache.camel.TypeConverter`
  - `org.apache.camel.spi.Registry`
  - `java.lang.Exception`
- Der erste Parameter ist stets der Body der Message



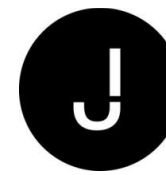
C3.2

## **CHOICE UND PREDICATES**



```
<camelContext
xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="file:src/data?noop=true" />
    <choice>
      <when>
        <xpath>/person/city = 'London'</xpath>
        <to uri="file:target/messages/uk" />
      </when>
      <otherwise>
        <to uri="file:target/messages/others" />
      </otherwise>
    </choice>
  </route>
</camelContext>
```

- Ein Predicate bestimmt ein logisches Ergebnis zur Verwendung
  - `choice`
  - `filter`
  
- `interface Predicate` mit
  - Methode `boolean matches(Exchange exchange)`
  
- Implementierungen
  - `el`
  - `groovy`
  - `javaScript`
  - ...
  - `xpath`

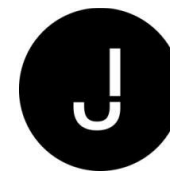


C3.3

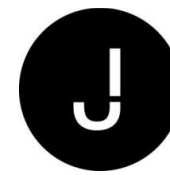
# TRANSFORMER



- Transformationen erfolgen
  - innerhalb einer Route
  - innerhalb einer Komponente
  - Automatisch durch registrierte Type-Converter
- Mechanismen
  - Data Formats
  - Templates
- Transformer ermöglichen eine Datenumwandlung in die von Camel unterstützten Datentypen
- Befehle
  - `transformer`
  - `marshal`
  - `unmarshal`

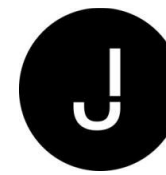


- Standard Java
  - Serialization
  - String
- Object Marshalling
  - Avro
  - Boon
  - JSON
  - Protobuf
- Object/XML Marshalling
  - Castor
  - JAXB
  - XmlBeans
  - XStream
  - JiBX



- Web Services
  - SOAP
- JSON / XML
  - XmlToJson
- Flache Datenstrukturen
  - BeanIO
  - Bindy
  - CSV
  - EDI
  - Flatpack
  - uniVocity
- Domain specific marshalling
  - HL7

- **Kompression**
  - GZip data format
  - Zip DataFormat
  - Zip File DataFormat
- **Security**
  - Crypto
  - PGP
  - XMLSecurity DataFormat
- **Verschiedenes**
  - Base64
  - Custom DataFormat RSS
  - TidyMarkup
  - Syslog
  - ICal
  - Barcode



C3.4

## **FEHLERBEHANDLUNG**

- Globale Error Handlers
  - Logging
  - Dead Letter
  - Default
- `onException`
  - Fängt einen Ausnahme-Typen
- try-catch-Blöcke
  - `doTry`
  - `doCatch`
  - `doFinally`
  - `throwException`
- Kompensation
  - Hinterlegen von Completion-Handlern, die je nach Erfolg der Routenausführung aufgerufen werden
    - Damit können verschachtelte try-catch-Blöcke vermieden werden

- Definition innerhalb des CamelContext
  - `<errorHandler level="DEBUG" type="LoggingErrorHandler" logName="ErrorLog"></errorHandler>`
  - `<errorHandler level="DEBUG" type="DeadLetterChannel" deadLetterUri="direct:errors"></errorHandler>`
- Error Handler definieren optional einen Retry-Mechanismus
  - Redelivery
  - Benutzung eines Predicate-Ausdrucks in `retryWhile`  
`<errorHandler id="myRouteSpecificErrorHandler" type="DefaultErrorHandler">`  
**`<redeliveryPolicy maximumRedeliveries="2"/>`**  
`</errorHandler>`

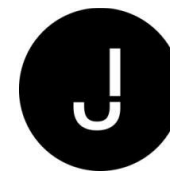
- Definition innerhalb des CamelContexts

```
public class ExceptionRouteBuilder extends RouteBuilder {  
    @Override  
    public void configure() throws Exception {  
        onException(MyException.class, AnotherException.class)  
        .to("direct:error");  
        from("...") //...  
    }  
}
```



- Hier werden analog dem Java-Exception-Mechanismus innerhalb der Routen-Definition try-catch-finally-Bereiche definiert

```
<route>
  <from uri="..."/><to uri="..."/>
  <doTry>...
    <doCatch>
      <exception>MyException</exception>
      <to uri="..."/>
    </doCatch>
    <doFinally>
      <to uri="..."/>
    </doFinally>
  </doTry>
  <to uri="..."/>
</route>
```



- `onCompletion`
- `onException`