11/08/2012
Christian Müller

Apache Camel in Action
Common problems, solutions and best practices

ApacheCon Europe 2012 - Sinsheim

Transactional services. Powering progress

Who is Christian Müller

- 10+ years of experience in software development
  - Focus on Java and Java related technologies
  - Build enterprise integration solutions for telcos and financial institutes
- Senior Software Developer at Atos Worldline
  - Responsible for the technical implementation our integration solutions based on Camel and ServiceMix
- Apache Camel
  - Apache Camel Committer and PMC chair
- Other Apache projects
  - Partly involved in related projects like Apache Karaf, Apache ServiceMix, …

Agenda

11/08/2012
Christian Müller

- Which runtime I should use?
- Why does my headers disappear?
- How to handle (or not) errors in my route?
- Why does my routes and contexts have unpredictable names?
- How to start/stop or suspend/resume routes at runtime?
- How to configure routes at runtime?
- How transactions work in Camel?
- How to configure transactions in Camel?

- Q & A

- **Bonus**: How to separate my Camel routes?
- **Bonus:** Pipeline vs. Multicast & To vs. InOut/InOnly

Which runtime I should use?

- Standalone

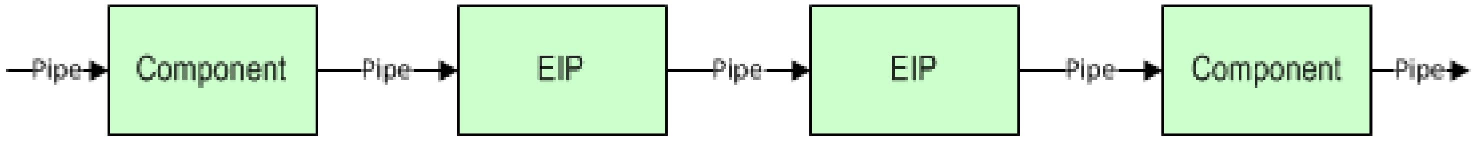
```
// create a Main instance
Main main = new Main();
// enable hangup support so you can press ctrl + c to terminate the JVM
main.enableHangupSupport();
// bind MyBean into the registery
main.bind("foo", new MyBean());
// add routes
main.addRouteBuilder(new MyRouteBuilder());
main.run();
```

- Tomcat, Jetty, …
- Karaf, ServiceMix, …
- Java EE server

Why does my headers disappear?

- How does the Camel pipeline work:

```
from(“cxf:bean:orderEntry”)
   .to(“bean:myService”)
   .to(“cxf:bean:mySoapService”)
   .to(“activemq:queue:ORDER_ENTRY”);
```



- The outcome of the previous “box” is the input for the next one
- The pipeline takes care of it
  - E.g. copy the out message of the previous “box” (if present) into the in message before calling the next “box”.

Why does my headers disappear?

- You set the result in your processor/bean into the out message without copy the in headers (and attachments):

```
public class MyProcessor implements Processor {

  public void process(Exchange exchange) throws Exception {
      Object result = …
      …
      exchange.getOut().setBody(result);
   }
}
```

Why does my headers disappear?

- Set the result into the in message and let Camel's pipeline do the work:

```
public class MyProcessor implements Processor {

  public void process(Exchange exchange) throws Exception {
    Object result = ...
    ...
    exchange.getIn().setBody(result);
  }
}
```

Why does my headers disappear?

11/08/2012
Christian Müller

- Or copy the in message headers (and attachments) into the out message if the exchange is out capable:

```
public class MyProcessor implements Processor {

  public void process(Exchange exchange) throws Exception {
    Object result = …

    if (exchange.getPattern().isOutCapable()) {
      exchange.getOut().setHeaders(exchange.getIn().getHeaders());
      exchange.getOut().setAttachments(exchange.getIn().getAttachments());
      exchange.getOut().setBody(result);
    } else {
      …
    }
  }
}
```

Exception handling

- Camel supports global (per Camel context) and route scoped error handling
- By default, Camel use the DefaultErrorHandler to handle exceptions which:
  - Do not redeliver the exchanges
  - Propagates the exceptions back to the caller
- Camel also provides the following error handlers:
  - NoErrorHandler
  - LoggingErrorHandler
  - DeadLetterErrorHandler
  - TransactionErrorHandler
- You can configure the behavior of the error handlers like:
  - redelivery count / redelivery while (Expression)
  - redelivery delay
  - redelivery back off multiplier
  - use the original message
  - ...

Exception handling

- Configure the DeadLetterErrorHandler as global error handler:
  - The exception will be handled and not propagated back to the caller
  - It will redeliver the exchange at max. 5 times
  - It will wait 1 second for the next redelivery
  - The failed exchange will be moved into the dead letter endpoint

```
errorHandler(
  deadLetterChannel("activemq:queue:DLQ")
    .maximumRedeliveries(5)
    .redeliveryDelay(1000));

from("activemq:queue:start").routeId("route-1")
  .to("bean:service1") // throws Service1Exception
  .to("…");
```

Exception handling

- For the next examples, assume we have the following route:

```
from("cxf:bean:mySoapService").routeId("route-1")
 .to("bean:service1") // throws Service1Exception
 .to("direct:sub");

from("direct:sub").routeId("route-2")
 .to("bean:service2") // throws Service2Exception
 .to("...");
```

Exception handling

- Global exception handling:
  - Both exceptions should be handled in the same way
  - Stop to continue routing the exchange
  - The exception should not be propagated back to the caller

```
onException(Exception.class)
  .handled(true)
  .to("bean:globalExceptionHandler");

from("cxf:bean:mySoapService").routeId("route-1")
  .to("bean:service1") // throws Service1Exception
  .to("direct:sub");

from("direct:sub").routeId("route-2")
  .to("bean:service2") // throws Service2Exception
  .to("…");
```

Exception handling

- Global exception handling:
  - Both exceptions should be handled in the same way
  - Stop to continue routing the exchange
  - The exception should be propagated back to the caller

```
onException(Exception.class)
  .handled(false)
  .to("bean:globalExceptionHandler");

from("cxf:bean:mySoapService").routeId("route-1")
  .to("bean:service1") // throws Service1Exception
  .to("direct:sub");

from("direct:sub").routeId("route-2")
  .to("bean:service2") // throws Service2Exception
  .to("…");
```

Exception handling

- Global exception handling:
  - Both exceptions should be handled in the same way
  - Continue routing the exchange
  - The exception should not be propagated back to the caller

```
onException(Exception.class)
  .continued(true)
  .to(“bean:globalExceptionHandler”);

from(“cxf:bean:mySoapService”).routeId(“route-1”)
  .to(“bean:service1”) // throws Service1Exception
  .to(“direct:sub”);

from(“direct:sub”).routeId(“route-2”)
  .to(“bean:service2”) // throws Service2Exception
  .to(“…”);
```

Exception handling

- Route scoped exception handling:
  - Service1Exception should be handled in a different (not global) way
  - Stop routing the exchange
  - The exception should not be propagated back to the caller

```
onException(Exception.class)
  .handled(true)
  .to("bean:globalExceptionHandler");

from("cxf:bean:mySoapService").routeId("route-1")
  .onException(Service1Exception.class)
    .handled(true)
    .to("bean:service1ExceptionHandler");
  .end()
  .to("bean:service1") // throws Service1Exception
  .to("direct:sub");

from("direct:sub").routeId("route-2")
  .to("bean:service2") // throws Service2Exception
  .to("…");
```

Exception handling

- Route scoped exception handling:
  - Service1Exception and Service2Exception should be handled in a different (not global) way
  - Stop routing the exchange
  - The exception should not be propagated back to the caller

```
onException(Exception.class)
 .handled(true)
 .to("bean:globalExceptionHandler");

from("cxf:bean:mySoapService").routeId("route-1")
 .onException(Service1Exception.class, Service2Exception.class)
   .handled(true)
   .to("bean:serviceExceptionHandler");
 .end()
 .to("bean:service1") // throws Service1Exception
 .to("direct:sub");

from("direct:sub").routeId("route-2")
 .errorHandler(noErrorHandler())
 .to("bean:service2") // throws Service2Exception
 .to("…");
```

Why does my routes and contexts have unpredictable names?

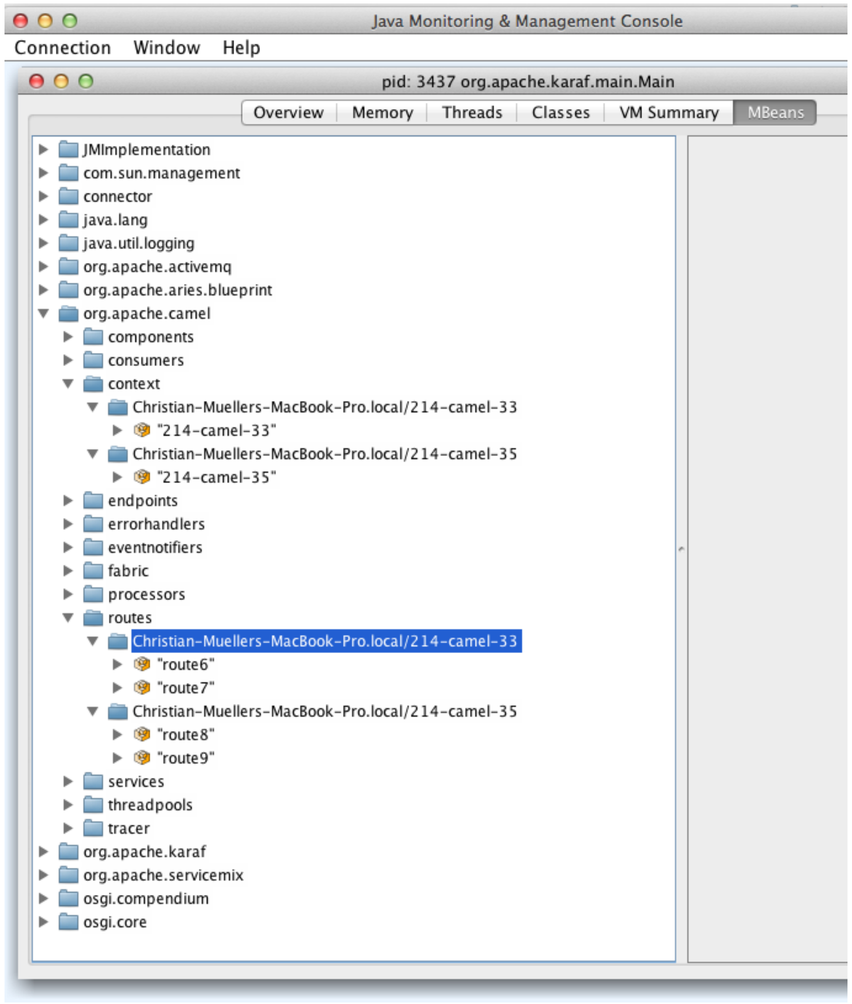
11/08/2012
Christian Müller

```
<?xml version="1.0"  encoding="UTF-8"?>
<beans ...>

 <camel:camelContext>
     <camel:route>
        <camel:from uri="direct:A"/>
        <camel:to uri="direct:B"/>
     </camel:route>
     <camel:route>
        <camel:from uri="direct:B"/>
        <camel:to uri="direct:C"/>
     </camel:route>
   </camel:camelContext>

   <camel:camelContext>
     <camel:route>
        <camel:from uri="direct:C"/>
        <camel:to uri="direct:D"/>
     </camel:route>
     <camel:route>
        <camel:from uri="direct:D"/>
        <camel:to uri="direct:E"/>
     </camel:route>
   </camel:camelContext>

</beans>
```


Java Monitoring & Management Console
Connection Window Help
pid: 3437 org.apache.karaf.main.Main
Overview Memory Threads Classes VM Summary MBeans

```
▶ JMImplementation
▶ com.sun.management
▶ connector
▶ java.lang
▶ java.util.logging
▶ org.apache.activemq
▶ org.apache.aries.blueprint
▼ org.apache.camel
  ▶ components
  ▶ consumers
  ▼ context
    ▼ Christian-Muellers-MacBook-Pro.local/214-camel-33
      ▶ "214-camel-33"
    ▼ Christian-Muellers-MacBook-Pro.local/214-camel-35
      ▶ "214-camel-35"
  ▶ endpoints
  ▶ errorhandlers
  ▶ eventnotifiers
  ▶ fabric
  ▶ processors
  ▼ routes
    ▼ Christian-Muellers-MacBook-Pro.local/214-camel-33
      ▶ "route6"
      ▶ "route7"
    ▼ Christian-Muellers-MacBook-Pro.local/214-camel-35
      ▶ "route8"
      ▶ "route9"
  ▶ services
  ▶ threadpools
  ▶ tracer
▶ org.apache.karaf
▶ org.apache.servicemix
▶ osgi.compendium
▶ osgi.core
```

Why does my routes and contexts have unpredictable names?

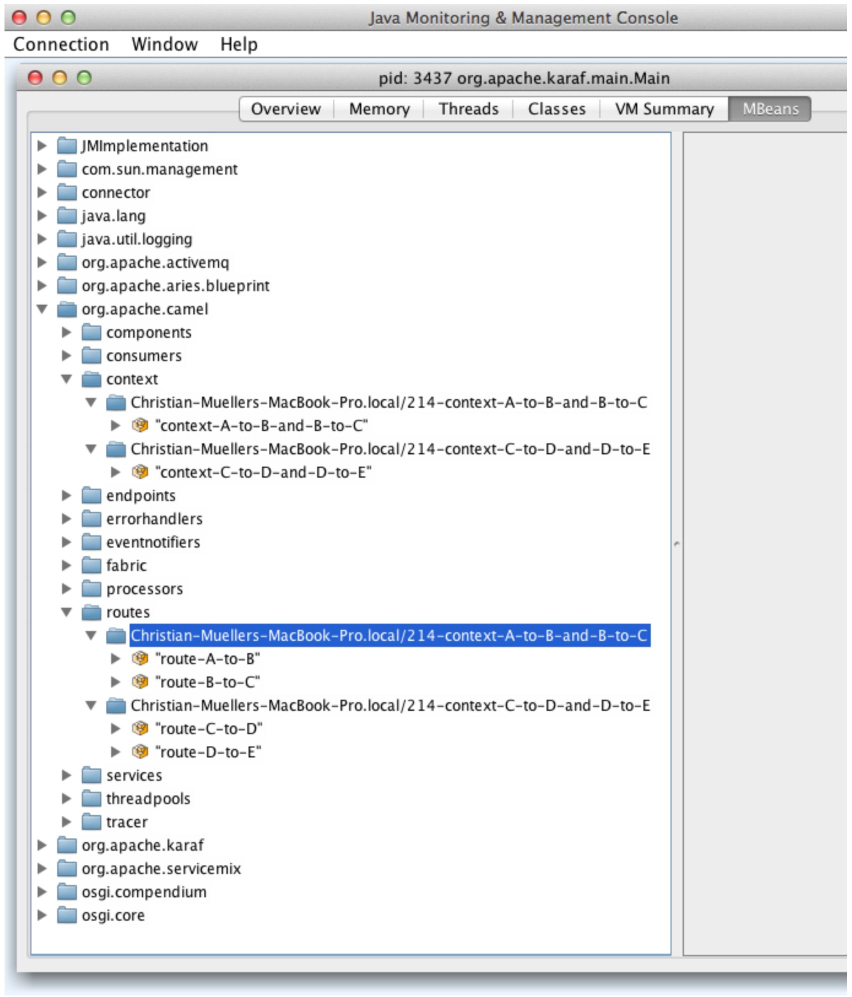
11/08/2012
Christian Müller

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

  <camel:camelContext id="context-A-to-B-and-B-to-C">
    <camel:route id="route-A-to-B">
      <camel:from uri="direct:A"/>
      <camel:to uri="direct:B"/>
    </camel:route>
    <camel:route id="route-B-to-C">
      <camel:from uri="direct:B"/>
      <camel:to uri="direct:C"/>
    </camel:route>
  </camel:camelContext>

  <camel:camelContext id="context-C-to-D-and-D-to-E">
    <camel:route id="route-C-to-D">
      <camel:from uri="direct:C"/>
      <camel:to uri="direct:D"/>
    </camel:route>
    <camel:route id="route-D-to-E">
      <camel:from uri="direct:D"/>
      <camel:to uri="direct:E"/>
    </camel:route>
  </camel:camelContext>

</beans>
```


Java Monitoring & Management Console
Connection
Window
Help
pid: 3437 org.apache.karaf.main.Main
Overview
Memory
Threads
Classes
VM Summary
MBeans

```
JMImplementation
com.sun.management
connector
java.lang
java.util.logging
org.apache.activemq
org.apache.aries.blueprint
org.apache.camel
  components
  consumers
  context
    Christian-Muellers-MacBook-Pro.local/214-context-A-to-B-and-B-to-C
      "context-A-to-B-and-B-to-C"
    Christian-Muellers-MacBook-Pro.local/214-context-C-to-D-and-D-to-E
      "context-C-to-D-and-D-to-E"
  endpoints
  errorhandlers
  eventnotifiers
  fabric
  processors
  routes
    Christian-Muellers-MacBook-Pro.local/214-context-A-to-B-and-B-to-C
      "route-A-to-B"
      "route-B-to-C"
    Christian-Muellers-MacBook-Pro.local/214-context-C-to-D-and-D-to-E
      "route-C-to-D"
      "route-D-to-E"
  services
  threadpools
  tracer
org.apache.karaf
org.apache.servicemix
osgi.compendium
osgi.core
```

Why does my routes and contexts have unpredictable names?

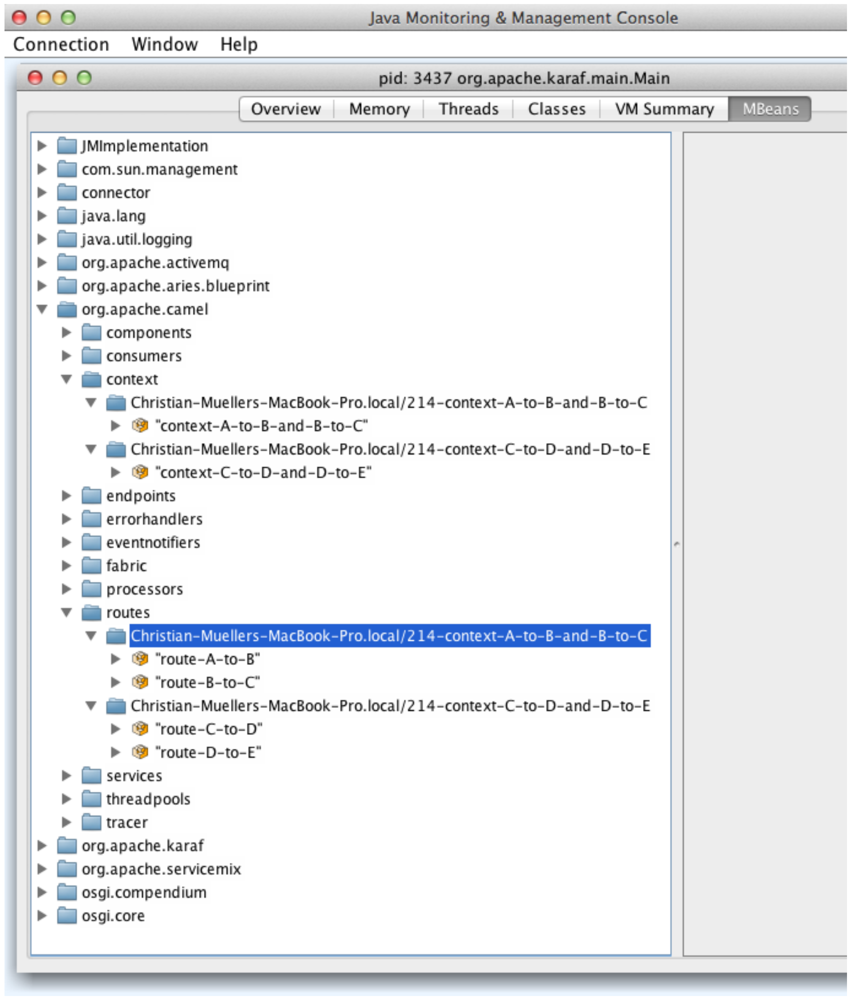
11/08/2012
Christian Müller

```
public class SampleRoute extends RouteBuilder {
 public void configure() throws Exception {
     from("direct:A").routeId("route-A-to-B")
         .to("direct:B");
     …
  }
}

<?xml version="1.0" encoding="UTF-8"?>
<beans … >
  <bean id="sampleRoute" class="…SampleRoute" />

  <camel:camelContext id="context-A-to-B-and-B-to-C">
    <camel:routeBuilder ref="sampleRoute" />
    …
  </camel:camelContext>
</beans>

public class Main {
  public static void main(String… args) {
    …
    DefaultCamelContext ctx = new DefaultCamelContext();
    ctx.setManagementName("context-A-to-B-and-B-to-C");
    …
  }
}
```


Java Monitoring & Management Console
Connection Window Help
pid: 3437 org.apache.karaf.main.Main
Overview Memory Threads Classes VM Summary MBeans

```
▶ JMImplementation
▶ com.sun.management
▶ connector
▶ java.lang
▶ java.util.logging
▶ org.apache.activemq
▶ org.apache.aries.blueprint
▼ org.apache.camel
  ▶ components
  ▶ consumers
  ▼ context
    ▼ Christian-Muellers-MacBook-Pro.local/214-context-A-to-B-and-B-to-C
      ▶ "context-A-to-B-and-B-to-C"
    ▼ Christian-Muellers-MacBook-Pro.local/214-context-C-to-D-and-D-to-E
      ▶ "context-C-to-D-and-D-to-E"
  ▶ endpoints
  ▶ errorhandlers
  ▶ eventnotifiers
  ▶ fabric
  ▶ processors
  ▼ routes
    ▼ Christian-Muellers-MacBook-Pro.local/214-context-A-to-B-and-B-to-C
      ▶ "route-A-to-B"
      ▶ "route-B-to-C"
    ▼ Christian-Muellers-MacBook-Pro.local/214-context-C-to-D-and-D-to-E
      ▶ "route-C-to-D"
      ▶ "route-D-to-E"
  ▶ services
  ▶ threadpools
  ▶ tracer
▶ org.apache.karaf
▶ org.apache.servicemix
▶ osgi.compendium
▶ osgi.core
```

How to start/stop or suspend/resume routes at runtime?

11/08/2012
Christian Müller

- Use the RoutePolicy/RoutePolicySupport.
  - ThrottlingInflightRoutePolicy
  - SimpleScheduledRoutePolicy/CronScheduledRoutePolicy
  - create your own RoutePolicy
- Use the Camel API (we will see it later).

```
public interface RoutePolicy {
  void onInit(Route route);
  void onRemove(Route route);
  void onStart(Route route);
  void onStop(Route route);
  void onSuspend(Route route);
  void onResume(Route route);
  void onExchangeBegin(Route route, Exchange exchange);
  void onExchangeDone(Route route, Exchange exchange);
}
```

How to start/stop or suspend/resume routes at runtime?

- Assume the following requirement for your scheduled route:
  - The route has to be scheduled from an external scheduler via JMS command messages.

```
RoutePolicy policy = new MyCustomRoutePolicy(“activemq:queue:command”);

from("seda:start”).routeId(”scheduledRoute”)
  .noAutoStartup()
  .routePolicy(policy)
  ...
  .to("mock:end");
```

How to start/stop or suspend/resume routes at runtime?

11/08/2012
Christian Müller

```
public class MyCustomRoutePolicy extends RoutePolicySupport {
  private String endpointUrl;
  public MyCustomRoutePolicy(String endpointUrl) {
    this.endpointUrl = endpointUrl;
  }
  public void onInit(final Route route) {
    CamelContext camelContext = route.getRouteContext().getCamelContext();
    Endpoint endpoint = camelContext.getEndpoint(endpointUrl);
    endpoint.createConsumer(new Processor() {
      public void process(Exchange exchange) throws Exception {
        String command = exchange.getIn().getBody(String.class);
        if ("start".equals(command)) {
          startRoute(route);
        } else if ("resume".equals(command)) {
          resumeRoute(route);
        } else if ("stop".equals(command)) {
          stopRoute(route);
        } else if ("suspend".equals(command)) {
          suspendRoute(route);
        }
      }).start();
  }
}
```

How to configure routes at runtime?

- Camel has a Java API which allows you to add/modify/remove routes at runtime
  - context.addRoutes(routeBuilderInstance)
  - context.getRoute(“routeId”)
  - context.removeRoute(“routeId”)

- And as mentioned before also to start/stop and resume/suspend routes
  - context.startRoute(“routeId”)
  - context.stopRoute(“routeId”)
  - context.resumeRoute(“routeId”)
  - context.suspendRoute(“routeId”);

How to configure routes at runtime?
Sample 1

▶ Modifying endpoints at runtime:

```
from("cxf:bean:ORDER_ENTRY").routeId("orderEntrySOAP")
 ...
 .setHeader("ENQUEUE_TIME", System.currentTimeMillies())
 .to("seda:ORDER_ENTRY")


from("seda:ORDER_ENTRY").routeId("orderEntry")
 .setHeader("DEQUEUE_TIME", System.currentTimeMillies())
 .to("bean:orderEntryService?method=timeConsumingProcessing")
 .to("bean:performanceMonitor?method=adjustConcurrentConsumers");
```

How to configure routes at runtime?
Sample 1

```
public class PerformanceMonitor {

  public void adjustConcurrentConsumers(Exchange exchange) throws Exception {
    long enqueueTime = exchange.getIn().getHeader(" ENQUEUE_TIME", Long.class);
    long dequeueTime = exchange.getIn().getHeader(" DEQUEUE_TIME", Long.class);

    if ((dequeueTime – enqueueTime) > 5000) {
      CamelContext context = exchange.getContext();
      // only stopping/starting the consumer doesn't work (yet)
      context.stopRoute("orderEntry");
      Route orderEntryRoute = context.getRoute("orderEntry");
      SedaEndpoint endpoint = (SedaEndpoint) orderEntryRoute.getEndpoint();
      int consumerCount = endpoint.getConcurrentConsumers();
      endpoint.setConcurrentConsumers(consumerCount * 2);
      context.startRoute("orderEntry");
    }
  }
}
```

How to configure routes at runtime?
Sample 2

- Dedicated processing route per customer in a static fashion:

```
from(“activemq:queue:ORDER_ENTRY").routeId(“orderEntry”)
 .routingSlip(simple(“activemq:queue:ORDER_ENTRY.${header.COMPANY}”))
 .end();

from(“activemq:queue:ORDER_ENTRY.BANK1”).routeId(“orderEntryBank1”)
 .to(“bean:orderEntryService?method=timeConsumingProcessing”)
 ...
 .end();

...

from(“activemq:queue:ORDER_ENTRY.BANK9”).routeId(“orderEntryBank9”)
 .to(“bean:orderEntryService?method=timeConsumingProcessing”)
 ...
 .end();
```

How to configure routes at runtime?
Sample 2

- Dedicated processing route per customer in a dynamic fashion:

```
from(“activemq:queue:ORDER_ENTRY”).routeId(“orderEntry”)
  .process(new DynamicRouteBuilderProcessor())
  .routingSlip(simple(“activemq:queue:ORDER_ENTRY.${header.COMPANY}”))
  .end();


public class DynamicProcessor implements Processor {
  public void process(final Exchange exchange) throws Exception {
    final String company = exchange.getIn().getHeader(“COMPANY”, String.class);
    Route route = exchange.getContext().getRoute("orderEntry" + company);
    if (route == null) {
      exchange.getContext().addRoutes(new RouteBuilder() {
        public void configure() throws Exception {
          from("activemq:queue:ORDER_ENTRY." + company).routeId("orderEntry" + company)
          .to("bean:orderEntryService?method=timeConsumingProcessing")
        }
      });
    }
  }
}
```

How does transactions work in Camel?

- **NOT** all Camel components are transaction aware!

- Components which supports transactions are:
  - SQL component
  - Ibatis/MyBatis component
  - JPA component
  - Hibernate component
  - JMS component
    - ActiveMQ component
  - SJMS component (Camel 2.11.0)

- Components which mimic transaction behavior:
  - File component
  - FTP/SFTP/FTPS component
  - others…

How does transactions work in Camel?

11/08/2012
Christian Müller

- Camels transaction support leverages on Springs PlatformTransactionManager interface
  - DataSourceTransactionManager
  - JmsTransactionManager
  - JpaTransactionManager/HibernateTransactionManager
  - JtaTransactionManager
  - and others …
- **Important**: One transaction is associated with a single thread of execution!
  - If you use “seda”, “vm”, “jms” or any other protocol in your sub route which will process the exchange in an different thread, this execution will not be part of this transaction context!
- A transaction is **NOT** associated with the exchange itself!
  - We want support asynchronous transactions in Camel 3.0.0.
- Consuming multiple exchanges in one single transaction is not supported yet.
  - The SJMS component supports this (Camel 2.11.0)

How does transactions work in Camel?

- Does your system requires transactions?
  - Do you use components which support transactions?
  - Do you update the content (read only access doesn't requires TX)?
  - Do you update the database content in two or more different places?
  - You read/write from/into multiple JMS destinations?
- Does your system requires XA transactions?
  - You access more than one transactional component and compensations doesn't work for you?
- What are **compensations**?
  - Using a normal TX and "deal" with the errors (e.g. duplicate messages).
    - Write idempotent consumers (which can handle duplicates).
      - Sample: Queue -> DB update -> Queue

How does transactions work in Camel?

- Try to **avoid XA**, because
  - it’s more complex to set up and easy to do it wrong
  - it’s more expensive and difficult to test
    - our unit test “only” test the business exceptions
    - you also have to test the technical exceptions to be sure it will work
    - You may get different results in different environments (OS, disc, TX manager, …)
    - you may have to enable this feature explicitly (like in Oracle)
  - it’s slower (depending on the provider)
  - your resource may doesn’t support it (like HSQLDB)
  - and it’s also not bulletproof…

How to configure transactions in Camel?

- Samples: https://github.com/muellerc/camel-in-transaction
  - JMS TX
  - JDBC TX
  - XA TX
    - Atomicos
    - Bitronix
    - Aries/Geronimo
    - JOTM

How to configure transactions in Camel?

- A typical example with **JMS TX**:
  1. Start a messaging TX
  2. Consume a message from a queue
  3. Execute some business logic
  4. Write the message into another queue
  5. Commit the messaging TX

How to configure transactions in Camel?

11/08/2012
Christian Müller

```
public void configure() throws Exception {
  from("activemqTx:queue:transaction.incoming")
    .transacted("REQUIRED")
    .to("bean:businessService?method=computeOffer")
    .to("activemqTx:queue:transaction.outgoing");
}
```

How to configure transactions in Camel?

11/08/2012
Christian Müller

```
<bean id="txMgr" class="org.springframework.jms.connection.JmsTransactionManager">
  <property name="connectionFactory" ref="connectionFactory"/>
</bean>

<bean id="REQUIRED" class="org.apache.camel.spring.spi.SpringTransactionPolicy">
  <property name="transactionManager" ref="txMgr"/>
  <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
</bean>

<bean id="connectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory">
  <property name="maxConnections" value="8" />
  <property name="connectionFactory">
    <bean class="org.apache.activemq.ActiveMQConnectionFactory">
      <property name="brokerURL" value="tcp://localhost:61616"/>
    </bean>
  </property>
</bean>

<bean id="activemqTx" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory" ref="connectionFactory"/>
  <property name="transacted" value="true"/>
  <property name="transactionManager" ref="txMgr"/>
</bean>
```

How to configure transactions in Camel?

- A typical example with **JDBC TX**:
  1. Receive a message
  2. Start a database TX
  3. Update the database (withdrawal money)
  4. Update the database (deposit money)
  5. Commit the database TX

How to configure transactions in Camel?

```
public void configure() throws Exception {
  from("seda:transaction.incoming")
    .transacted(”REQUIRED")
    .to("sql:UPDATE account SET balance = (SELECT balance from account where name = 'foo') - #
WHERE name = 'foo'?dataSourceRef=dataSource")
    .to("sql:UPDATE account SET balance = (SELECT balance from account where name = 'bar') + #
WHERE name = 'bar'?dataSourceRef=dataSource")
    .to("seda:transaction.outgoing");
}
```

How to configure transactions in Camel?

```
<bean id=”txMgr" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>

<bean id=”REQUIRED" class="org.apache.camel.spring.spi.SpringTransactionPolicy">
  <property name="transactionManager" ref=”txMgr"/>
  <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
</bean>

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.apache.derby.jdbc.EmbeddedDataSource40"/>
  <property name="url" value="jdbc:derby:target/testdb;create=true"/>
  <property name="defaultAutoCommit" value="false"/>
</bean>
```

How to configure transactions in Camel?

- A typical financial example with **XA**:
  1. Start a messaging TX
  2. Consume a financial transaction from a queue
  3. Start a database TX
  4. Update the database (withdrawal money)
  5. Update the database (deposit money)
  6. Write the financial transaction into another queue
  7. Commit the database TX
  8. Commit the messaging TX

How to configure transactions in Camel?

```
public void configure() throws Exception {
  from("activemqXa:queue:transaction.incoming")
    .transacted(”REQUIRED")
    .to("sql:UPDATE account SET balance = (SELECT balance from account where name = 'foo') - #
WHERE name = 'foo'?dataSourceRef=dataSource")
    .to("sql:UPDATE account SET balance = (SELECT balance from account where name = 'bar') + #
WHERE name = 'bar'?dataSourceRef=dataSource")
    .to("activemqXa:queue:transaction.outgoing");
}
```

How to configure transactions in Camel?

```
<bean id="jtaTxMgr" class="org.springframework.transaction.jta.JtaTransactionManager">
  <property name="transactionManager" ref=”txMgr"/>
  <property name="userTransaction" ref="userTransaction"/>
</bean>

<bean id="txMgr" class="com.atomikos.icatch.jta.UserTransactionManager" init-method="init"
destroy-method="close">
    <property name="forceShutdown" value="false"/>
</bean>

<bean id="userTransaction" class="com.atomikos.icatch.jta.UserTransactionImp">
    <property name="transactionTimeout" value="120"/>
</bean>

<bean id=”REQUIRED" class="org.apache.camel.spring.spi.SpringTransactionPolicy">
  <property name="transactionManager" ref="jtaTxMgr"/>
  <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
</bean>
```

How to configure transactions in Camel?

11/08/2012
Christian Müller

```
<bean id="resourceManager" class="org.apache.activemq.pool.ActiveMQResourceManager" init-
method="recoverResource">
  <property name="transactionManager" ref=”txMgr" />
  <property name="connectionFactory" ref=”connectionFactory" />
  <property name="resourceName" value="activemq.default" />
</bean>

<bean id=”connectionFactory" class="org.apache.activemq.pool.XaPooledConnectionFactory” init-
method="start" destroy-method="stop”>
  <property name="maxConnections" value="8" />
  <property name="connectionFactory" ref=”xaConnectionFactory" />
  <property name="transactionManager" ref="txMgr"/>
</bean>

<bean id=”xaConnectionFactory" class="org.apache.activemq.ActiveMQXAConnectionFactory”>
    <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>

<bean id="activemqXa" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory" ref=”connectionFactory"/>
  <property name="transacted" value="false"/>
  <property name="transactionManager" ref="jtaTxMgr"/>
</bean>
```

How to configure transactions in Camel?

11/08/2012
Christian Müller

```
<bean id="dataSource" class="org.apache.commons.dbcp.managed.BasicManagedDataSource">
  <property name="transactionManager" ref=”txMgr"/>
  <property name="driverClassName" value="org.apache.derby.jdbc.EmbeddedXADataSource40"/>
  <property name="url" value="jdbc:derby:target/testdb;create=true"/>
  <property name="defaultAutoCommit" value="false"/>
</bean>
```

Q & A

http://camel.apache.org/faq.html

Thank you

Why using multiple Camel contexts?

- In a standalone Camel application, you should only use one CamelContext in general (there is no advantage to use multiple CamelContexts).

- One CamelContext can have as many Camel routes as needed:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
   <camel:camelContext id="main">
      <camel:routeBuilder ref="routeBuilder1" />
      <camel:routeBuilder ref="routeBuilder2" />
      ...
   </camel:camelContext>
</beans>
```

Why using multiple Camel contexts?

- Or use the “packageScan” or “componentScan” feature, e.G.:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
  <camel:camelContext id=“main”>
    <camel:packageScan>
      <camel:package>org.company.product.module</camel:package>
    </camel:packageScan>
  </camel:camelContext>
</beans>
```

Why using multiple Camel contexts?

- If you are deploying your Camel application into an OSGI container or application server, you should consider using multiple CamelContexts.
  - Each of these integration logic is deployable in isolation.
    - They can bee loosely coupled via the different provided VM, JMS, NMR, … components.
  - Each CamelContexts can have multiple Camel routes.
  - A downtime of one part may didn't affect other parts (customers).

```
from("cxf:bean:orderEntryService").routeId("mainRoute")
 .dynamicRouter(simple("activemq:queue:ORDER_ENTRY_${body.customerId}"));

<?xml version="1.0" encoding="UTF-8"?>
<beans … >
   <bean id="mainRoute" class="…" />
   <camel:camelContext id="main">
      <camel:routeBuilder ref="mainRoute" />
   </camel:camelContext>
</beans>
```

Why using multiple Camel contexts?

11/08/2012
Christian Müller

```
from("activemq:queue:ORDER_ENTRY_BANK1").routeId("bank1Route")
  .to("bean:orderEntryProcessor?method=processBank1");

<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
   <bean id="bank1Route" class="..." />
   <camel:camelContext id="bank1">
      <camel:routeBuilder ref="bank1Route" />
   </camel:camelContext>
</beans>


from("activemq:queue:ORDER_ENTRY_BANK2").routeId("bank2Route")
  .to("cxf:bean:enrichService")
 .to("bean:orderEntryProcessor?method=processBank2");

<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
   <bean id="bank2Route" class="..." />
   <camel:camelContext id="bank2">
      <camel:routeBuilder ref="bank2Route" />
   </camel:camelContext>
</beans>
```

How to separate my Camel routes?

- If you Camel route (your "*configure()*" method) grows
  - to more than you can show on one screen
  - or it becomes difficult to understand

- Split them into separate routes (which share the same CamelContext)
  - If necessary, connect them together via the Camel provided direct or seda components

How to separate my Camel routes?

11/08/2012
Christian Müller

```
from(“activemq:queue:ORDER_PROCESS.STEP_1”).routeId(“step1”)
 .to(“validator:myschema.xsd”)
 .convertBodyTo(Order.class)
 .to(“bean:orderEntryService?method=audit”)
 .enrichRef(“direct:enrichCustomerData", “myAggregationStrategy")
 .to(“bean:orderEntryService?method=process”)
 .to(“activemq:queue:ORDER_PROCESS.STEP_2”);

from(“activemq:queue:ORDER_PROCESS.STEP_2”).routeId(“step2”)
 .enrichRef(“direct:updateLagacySystem", “myLagacyAggregationStrategy“)
 .to(“bean:orderEntryService?method=postProcess”)
 .convertBodyTo(OrderReceipt.class)
 .to(activemq:queue:ORDER_PROCESS.RESPONSE);

from(“direct:enrichCustomerData”).routeId(“enrichCustomerData”)
 .convertBodyTo(QueryCustomerDataRequest.class)
 .setHeader(CxfConstants.OPERATION_NAME, queryCustomerData)
 .to(“cxf:bean:customerService”);

from(“direct:updateLagacySystem").routeId(“updateLagacySystem”)
 .convertBodyTo(UpdateRequest.class)
 .to(“ims:…”);
```

How to separate my Camel routes?

```
public class OrderEntryRoute extends RouteBuilder {

    public void configure() throws Exception {
        configureStep1();
        configureStep2();
        configureEnrichCustomerData();
        configureUpdateLagacySystem();
    }

    public void configureStep1 () throws Exception {
        from(”activemq:queue:ORDER_PROCESS.STEP_1").routeId(“step1”)
          ...
          .to(“activemq:queue:ORDER_PROCESS.STEP_2”);
    }

    public void configureStep2 () throws Exception {
        ...
    }

    ...
}
```

How to separate my Camel routes?

```
public class OrderEntryStep1Route extends RouteBuilder {

    public void configure () throws Exception {
        from(”activemq:queue:ORDER_PROCESS.STEP_1").routeId(“step1”)
          .to(“myschema.xsd”)
          .convertBodyTo(Order.class)
          .to(“bean:orderEntryService?method=audit”)
          .enrichRef(“direct:enrichCustomerData", “myAggregationStrategy")
          .to(“bean:orderEntryService?method=process”)
          .to(“activemq:queue:ORDER_PROCESS.STEP_2”);
    }
}

public class OrderEntryStep2Route extends RouteBuilder {

    public void configure () throws Exception {
        ...
    }
}

...
```

pipeline vs. multicast & to vs. inOut/inOnly!

11/08/2012
Christian Müller

- Pipeline:
  - the result of the previous processor is the input of the next processor:

```
from("direct:start")
 .to("direct:foo")
 .to("direct:bar");

from("direct:start")
 .to("direct:foo", "direct:bar");

from("direct:start")
 .pipeline()
  .to("direct:foo")
  .to("direct:bar");

from("direct:start")
 .pipeline("direct:foo", "direct:bar");
```

pipeline vs. multicast & to vs. inOut/inOnly!

- Multicast:
  - each processor in a multicast will receive the same input:

```
from("direct:start")
  .multicast()
    .to("direct:foo")
    .to("direct:bar");

from("direct:start")
  .multicast()
    .to("direct:foo", "direct:bar");
```

pipeline vs. multicast & to vs. inOut/inOnly!

- Prefer to use inOut/inOnly over to (if possible) to be more explicitly:
- Processing based on the Exchange MEP:

```
from(“direct:start”)
  .to(“direct:foo”, “direct:bar”);

from(“direct:start”)
  .setExchangePattern(ExchangePattern.InOut)
    .to(“direct:foo”, “direct:bar”);
```

~~from(“direct:start”)~~
~~.inOut()~~
~~.to(“direct:foo”, “direct:bar”);~~

- Routing based on the MEP configured on the endpoint:

```
from(“direct:start”)
  .to(ExchangePattern.InOut, “direct:foo”, “direct:bar”);

from(“direct:start”)
  .inOut(“direct:foo”, “direct:bar”);
```