Camel Testing

Camel provides many facilities for testing. It includes The Camel Test Kit provided in two jar files containing Junit extensions, Mock component and producer template.

Unit testing:

* With CamelTestSupport calass.  
   Extending this class we have new facilities for testing like producer template ( for messages sending ), enhanced assertions, access to the camel context and others.

Example:

public class CamelFileSystemRouterTest extends CamelTestSupport {  
  
 protected RouteBuilder createRouteBuilder() throws Exception {  
 return new CamelFileSystemRouter();  
 }  
  
 @Test  
 public void testFileMoving() throws InterruptedException {  
 template.sendBodyAndHeader("file://from", "Hello World", Exchange.*FILE\_NAME*, "hello\_world.txt");  
 Thread.*sleep*(1000);  
 File target = new File("out/hello\_world.txt");  
 *assertTrue*("File not moved", target.exists());  
 String content = context.getTypeConverter().convertTo(String.class, target);  
 *assertEquals*("Hello World", content);  
 }  
  
}

* SpringCamelTestSupport is used to test routes based on Spring XML. (page 159)
* with Mock component that allows you to check if:

- the correct number of messages are received on each endpoint

- the messages arrive in the correct order

- the correct payloads are received

- the test ran within the expected time period

Extending CamelTestSupport allows to create mocks very simple using ”getMockEndpoint()” method.  
Example:

public class CamelMQRouterTest extends CamelTestSupport {  
  
 protected RouteBuilder routeBuilder() throws Exception{  
 return new RouteBuilder() {  
 @Override  
 public void configure() throws Exception {  
 from("rabbitmq:foo").to("mock:quote");  
 }  
 };  
 }  
  
 @Test  
 public void testMQRoute() throws InterruptedException {  
 MockEndpoint mockedEndpoint = getMockEndpoint("mock:quote");  
 mockedEndpoint.setExpectedCount(1);  
 template.sendBody("rabbitmq:foo", "Camel");  
 mockedEndpoint.assertIsSatisfied();

List<Exchange> list = mockedEndpoint.getReceivedExchanges();  
 String body = list.get(0).getIn().getBody(String.class);

assertTrue(body.contains("Camel");))

}  
}

**Integration Testing:**

For integration testing purpose you can inspect destination routes using following code:

*BrowsableEndpoint be = context.getEndpoint(“URI”, BrowsableEnpoint.class)*

*List<Exchange> list = be.getEchanges();*

After that you can check all condition about messages and their oreder.

Another approach for integration testing is to use NotifyBuilder. It is used to define condition for messages being routed to an endpoint. Then it offers methods to test whether the conditions have been meet.

Example:

NotifyBuilder notify =

new NotifyBuilder(context).from(“URI”).whenDone(1).create();

**Error generation**

Connectivity errors can be simulated **using processor**:

from(“direct:file”)

.process(new Processor(){

public void process (Exchange exchange) throws Exception {

throw new ConnectException(“Simulated connection error”);

}

})  
.to(“mock:http”);

Also, errors can be simulated using mocks. This advantage of this approach is that you don’t have to alter real route.

MockEndpoint me = getMockEndpoint(“mock:http”);

me.whenAnyExchangeReceived(new Processor(){

public void process (Exchange exchange) throws Exception {

throw new ConnectException(“Simulated connection error”);

})

Another method for errors simulating is using interceptors. An interceptor allows you to intercept any given message and act upon it. Camel provides three types of interceptors for intercepting incoming messages on a particular enpoint { interceptFromEnpoint() }, messages that are about to be sent to a particular enpoint { interceptSendToEndpoint() }, and every single step a message is routed

{ intercept() }. After interceptor running is done, the message would normally be sent to the originally endpoint, but you can instruct Camel to skip this step using the *skipSendToOriginalEndpoint* method:

interceptSendToEnpoint(“http://rider.com/rider”)

.skipSendToOriginalEndpoint();

.process(new SimulateHttpErrorProcessor());

We can add such an interceptor on existing route using adviceWith() method.

RouteDefinition route = context.getRouteDefinitions.get(“routeId”);

route.adviceWith(context, new RouteBuilder() {

//interceptor and processor code

}

});