RecipientList Annotation

@RecipientList Annotation

We support the use of @RecipientList on a bean method to easily create a dynamic Recipient List using a Java method.

Simple Example using @Consume and @RecipientList

```
package com.acme.foo;

public class RouterBean {

    @Consume(uri = "activemq:foo")
    @RecipientList
    public String[] route(String body) {
        return new String[]{"activemq:bar", "activemq:whatnot"};
    }
}
```

For example if the above bean is configured in Spring when using a **<camelContext>** element as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<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.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
    ">

  <camelContext xmlns="http://activemq.apache.org/camel/schema/spring"/>

  <bean id="myRecipientList" class="com.acme.foo.RouterBean"/>

</beans>
```

then a route will be created consuming from the **foo** queue on the ActiveMQ component which when a message is received the message will be forwarded to the endpoints defined by the result of this method call - namely the **bar** and **whatnot** queues.

How it works

The return value of the @RecipientList method is converted to either a java.util.Collection / java.util.Iterator or array of objects where each element is converted to an Endpoint or a String, or if you are only going to route to a single endpoint then just return either an Endpoint object or an object that can be converted to a String. So the following methods are all valid

```
@RecipientList
public String[] route(String body) { ... }

@RecipientList
public List<String> route(String body) { ... }

@RecipientList
public Endpoint route(String body) { ... }

@RecipientList
public Endpoint[] route(String body) { ... }

@RecipientList
public Collection<Endpoint> route(String body) { ... }

@RecipientList
public URI route(String body) { ... }

@RecipientList
public URI[] route(String body) { ... }
```

Then for each endpoint or URI the message is forwarded a separate copy to that endpoint.

You can then use whatever Java code you wish to figure out what endpoints to route to; for example you can use the Bean Binding annotations to inject parts of the message body or headers or use Expression values on the message.

More Complex Example Using DSL

In this example we will use more complex Bean Binding, plus we will use a separate route to invoke the Recipient List

```
public class RouterBean2 {

    @RecipientList
    public String route(@Header("customerID") String custID String body) {
            if (custID == null)  return null;
        return "activemq:Customers.Orders." + custID;
    }
}

public class MyRouteBuilder extends RouteBuilder {
    protected void configure() {
        from("activemq:Orders.Incoming").recipientList(bean("myRouterBean", "route"));
    }
}
```

Notice how we are injecting some headers or expressions and using them to determine the recipients using Recipient List EIP.
See the Bean Integration for more details.