# Using getIn or getOut methods on Exchange

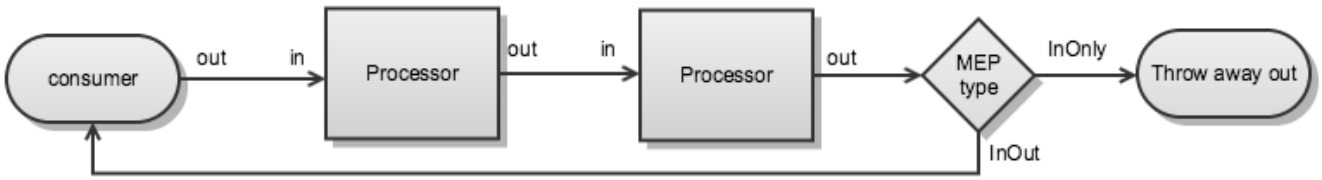
## Message exchange patterns and the Exchange object

The Camel API is influenced by APIs such as JBI specification, CXF which defines a concept called Message Exchange Patterns (MEP for short).

The MEP defines the messaging style used such as one-way (InOnly) or request-reply (InOut),
which means you have IN and optionally OUT messages. This closely maps to other APIs such as WS, WSDL, REST, JBI and the likes.

The Exchange API provides two methods to get a message, either `getIn` or `getOut`.
Obviously the `getIn` gets the IN message, and the `getOut` gets the OUT message.

## Flow of an exchange through a route




- The out message from each step is used as the in message for the next step
- if there is no out message then the in message is used instead
- For the InOut MEP the out from the last step in the route is returned to the producer. In case of InOnly the last out is thrown away

> **Beware of getOut to check if there is an out message**
>
> exchange.getOut creates an out message if there is none. So if you want to check if there is an out message then you should use exchange.hasOut instead

## Using getIn or getOut methods on Exchange

Now suppose you want to use a Camel Processor to adjust a message. This can be done as follows:

```
public void process(Exchange exchange) throws Exception {
   String body = exchange.getIn().getBody(String.class);
   // change the message to say Hello
   exchange.getOut().setBody("Hello " + body);
}
```

This seems intuitive and is what you would expect is the *right* approach to change a message from a Processor.
However there is an big issue - the `getOut` method will create a new Message, which means any other information
from the IN message will not be propagated; which means you will lose that data.
To remedy this we'll have to copy the data which is done as follows:

```
public void process(Exchange exchange) throws Exception {
   String body = exchange.getIn().getBody(String.class);
   // change the message to say Hello
   exchange.getOut().setBody("Hello " + body);
   // copy headers from IN to OUT to propagate them
   exchange.getOut().setHeaders(exchange.getIn().getHeaders());
}
```

Well that is not all, a Message can also contain attachments so to be sure you need to propagate those as well:

```
public void process(Exchange exchange) throws Exception {
   String body = exchange.getIn().getBody(String.class);
   // change the message to say Hello
   exchange.getOut().setBody("Hello " + body);
   // copy headers from IN to OUT to propagate them
   exchange.getOut().setHeaders(exchange.getIn().getHeaders();
   // copy attachements from IN to OUT to propagate them
   exchange.getOut().setAttachments(exchange.getIn().getAttachments());
}
```

Now we ensure that all additional data is propagated on the new OUT message. But its a shame we need 2 code lines to ensure data is propagated.
What you can do instead is to change the IN message instead, as shown below:

```
public void process(Exchange exchange) throws Exception {
   String body = exchange.getIn().getBody(String.class);
   // change the existing message to say Hello
   exchange.getIn().setBody("Hello " + body);
}
```

**Consider using getIn**

As shown above you most often need to alter the existing IN message, than creating a totally new OUT message.
And therefore it's often easier just to adjust the IN message directly.

Changing the IN message directly is possible in Camel as it doesn't mind. Camel will detect that the Exchange has no OUT message and therefore use the IN message instead.

## About Message Exchange Pattern and getOut

If the Exchange is using `InOnly` as the MEP, then you may think that the Exchange has no OUT message.
But you can still invoke the `getOut` method on Exchange; Camel will not barf.

So the example code above is possible for any kind of MEP. The MEP is *just* a flag on the Exchange which the Consumer and Producer adhere to.
You can change the MEP on the Exchange using the `setPattern` method. And likewise there is DSL to change it as well.