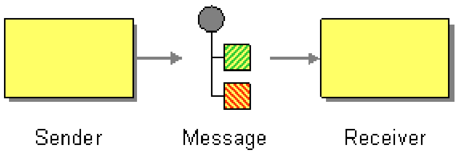
Message

Overview

A *message* is the smallest unit for transmitting data in a messaging system (represented by the grey dot in the figure below). The message itself might have some internal structure—for example, a message containing multiple parts—which is represented by geometrical figures attached to the grey dot in the figure below.

Figure 1. Message Pattern



Types of message

Mediation Router defines the following distinct message types:

- *In* message—a message that travels from a source endpoint to a target endpoint (typically, initiating a message exchange).
- *Out* message—a message that travels from a target endpoint back to a source endpoint (usually in response to an *In* message).
- *Fault* message—a message that travels from a target endpoint back to a source endpoint for the purpose of indicating an exception or error condition (usually in response to an *In* message).

All of these message types are represented internally by the `org.apache.camel.Message` interface.

Message structure

By default, Mediation Router applies the following structure to all message types:

- *Headers*—containing metadata or header data extracted from the message.
- *Body*—usually containing the entire message in its original form.

It is important to bear in mind that this division into headers and body is an abstract model of the message. Mediation Router supports many different compoments, which generate a wide variety of message formats. Ultimately, it is the underlying component implementation that decides what gets placed into the headers and body of a message.

Correlating messages

Internally, Mediation Router keeps track of a message ID, which could be used to correlate individual messages. In practice, however, the most important way that Mediation Router correlates messages is through *exchange* objects.

Exchange objects

An exchange object is an entity that encapsulates related messages, where the collection of related messages is referrred to as a *message exchange* and the rules governing the sequence of messages are referred to as an *exchange pattern*. For example, some common exchange patterns would be: one-way event messages (consisting of an *In* message); request-reply exchanges (consisting of an *In* message, followed by an *Out* message).

Accessing messages

When defining a routing rule in the Java DSL, you can access the headers and body of a message using the following DSL builder methods:

- `header(String name)`, `body()`—return named header and body of the current *In* message.
- `outBody()`—return body of the current *Out* message.
- `faultBody()`—return body of the current *Fault* message.

For example, to populate the *Out* message's `username` header by copying from the equivalent header in the *In* message, you could use the following route defined in Java DSL:

```
from(SourceURL).setOutHeader("username", header("user
name")).to(TargetURL);
```