

# **Camel-Volante**

## **User's Guide**

**Version 6.2.0**

## Proprietary Notice

**© 2019 Volante Technologies Inc. All rights reserved.**

Information in this document is subject to change without notice and does not represent a commitment on the part of Volante Technologies, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with that agreement. No part of this guide may be reproduced or retransmitted in any form or by any means electronic, mechanical, or otherwise, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Volante Technologies, Inc.

### Restricted Rights Legend

Copyright © 2001-2019 Volante Technologies, Inc. All rights reserved. All Volante products are trademarks or registered trademarks of Volante Technologies, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

### Contact us

In case you need any clarifications or have any queries, please do write to Volante Customer Support executives in [info@volantetech.com](mailto:info@volantetech.com).

### New York/New Jersey Headquarters:

Harborside 5

185 Hudson Street, #1605

Jersey City, NJ 07311, USA

Tel: +1 (201) 258-7459

## Table of Contents

<b>1</b>	<b>Camel-Volante.....</b>	<b>4</b>
1.1	Volante Component.....	4
1.2	Context Properties.....	5
1.3	Message Payload .....	6
1.4	Message Properties & Attachments.....	6
1.5	Streamed Processing .....	7
1.6	Using Envelope.....	8
1.6.1	Output Envelope .....	9
1.6.2	Input Envelope .....	11
1.7	Using user defined Java Objects .....	12
<b>2</b>	<b>Testing Volante Components in OSGI-Camel .....</b>	<b>13</b>

# 1 Camel-Volante

**Apache Camel**™ is a versatile open-source integration framework based on known Enterprise Integration Patterns. Camel empowers you to define routing and mediation rules in a variety of domain-specific languages, including a Java-based Fluent API, Spring or Blueprint XML Configuration files, and a Scala DSL. Refer to [Apache Camel Web Site](http://camel.apache.org/) for more information on Camel.

Apache Camel uses URIs to work directly with any kind of Transport or messaging model such as HTTP, ActiveMQ, JMS, JBI, SCA, MINA or CXF, as well as pluggable Components. The Volante component for Camel, described in detail in this document, allows you to process a Camel message using Volante's message flow.

## See also:

[Volante Component](#)

[Context Properties](#)

[Message Properties & Attachments](#)

[Batched Processing](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.1 Volante Component

Volante provides an adapter (camel-volante.jar) which lets you invoke Volante flows from Apache-Camel. The **volante:** component allows you to process a message using Volante flow. The invoked flow is normally expected to have one input and one output (which represents the body of the Camel message).

### URI format

```
volante:flowName[?options]
```

A flow by the given name should have been defined in a cartridge and the jars from the cartridge should be included in the classpath. The classpath should also include all other dependent jars including camel-volante.jar which defines the Volante component for Camel.

```
<to uri="volante:myflow"/>
```

Example Spring configuration:

```
<camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="activemq:InQueue"/>
    <to uri="volante:myflow"/>
    <to uri="activemq:OutQueue"/>
  </route>
</camelContext>
```

## Additional Options

You can pass additional options to the flow (TransformContext) using the options parameter. The options are name value pair separated by '&' symbol. You can append query options to the URI in the following format, ?option=value&option=value&...

For example,

```
volante:myFlow?message.format=xml
```

See also:

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.2 Context Properties

The following properties are automatically set in the context in which the flow executes. You can use the formula function `getContextObjectProperty()` to access these.

S.No	Context Property	Java Type	Description
1	camel.exchange	org.apache.camel.Exchange	The camel exchange.
2	camel.in	org.apache.camel.Message	Input message received from camel
3	camel.out	org.apache.camel.Message	Output message to be sent out to Camel.
4	camel.in.headers	Map<String, Object>	Input Headers
5	camel.out.headers	Map<String, Object>	Output Headers
6	camel.properties	Map<String, Object>	Exchange Properties

```
def headers = GetContextObjectProperty("camel.in.headers");
```

To access bean properties within these Java objects or Map, use `GetJavaProperty/`  
`SetJavaProperty` functions.

```
def myHeader = headers.GetJavaProperty("myHeader");
```

**See also:**

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.3 Message Payload

The message flow, to be invoked, should be defined with at least one input parameter and one output parameter. The payload of the message received from Camel is passed as the first parameter to the flow. The payload should be convertible to the type of the parameter defined in the flow. On the output side, the first output parameter from the flow is set as the payload of the outbound Camel message.

Typically, the Camel payload is String, byte [], File or an InputStream with corresponding type for first parameter in the flow. It is also possible to pass user defined objects and DataObjects as parameter as long as the flow is capable of handling it.

If the flow has more than one input/output parameter the Camel adapter binds the parameter to Camel message header with same name.

**See also:**

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.4 Message Properties & Attachments

If the input/output message of the flow is of type RawMessage, headers and attachments from the corresponding Camel message (input/output) are automatically copied. Camel headers are represented as properties in Volante message with same name and value. Attachments are also copied with same name and value.

### Message Header

You can use the message header of Camel message to pass or receive additional parameters to the flow. The first parameter in the flow always corresponds to the body of the message; if the flow takes additional parameters they are obtained from the header of input message.

```
<setHeader      headerName="OrderType"><constant>Buy</constant></setHeader>
<to uri="volante:OrderFlow"/>
```

In above example, the OrderFlow takes two parameters, one the body of the message and an additional parameter by name "OrderType". The constant value "Buy" is passed as the value of this parameter.

Similarly, if flow returns multiple outputs, the first one is bound to the body of the output message; Rest of the output parameters are set as headers of the output message.

This can be particularly useful if the flow produces multiple types of outputs. The first output parameter can return the body and second parameter can specify the type of the output.

```
<route>
  <to uri="volante:myflow"/>
  <choice>
    <when>
      <simple>${in.header.outputType} == 'MT202'</simple>
      <to uri="direct:b"/>
    </when>
    <when>
      <simple>${in.header.outputType} == 'MT202COV'</simple>
      <to uri="direct:c"/>
    </when>
    <otherwise>
      <to uri="direct:d"/>
    </otherwise>
  </choice>
</route>
```

In the above example, flow has two outputs; the second output parameter is named "outputType" and has a value "MT202" or MT202COV". In the camel route depending on this value, you can redirect the main output (first output parameter) to appropriate destination.

#### See also:

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.5 Streamed Processing

Volante has built-in support handling large input and outputs. The main difference between normal messages and a batched message is that the batch messages are huge in size. Hence, loading the entire input into memory is not desirable. Instead the input is "streamed" in and output is streamed out. At any point during execution the input/output is fully loaded in memory, instead we would be working with parts or input/output called records.

Input is parsed in a continuous process, record at time, the record is immediately consumed (mapped etc.) and the output is streamed out.

For Batched processing/Streaming, the flow should comply with the following requirements,

- The first input to the flow should be a RawMessage, representing the body of the incoming message.

- Flow should be written in such a way that the entire message is not read into memory; instead it should be processed, record at a time (Refer Batch/Phased Parse).
- The first output from the flow should be a RawMessage, with In/Out scope. The Camel Adapter would pass a cached RawMessage (possibly backed by temporary file) as input and the flow should append the output to this message.
- When the flow returns, the output is passed to the next component in the route as a stream. When the output is consumed (output stream is closed), the temporary file is closed and deleted.

**Note:** For Camel Version 2.12 and above, `streamCache="true"` has to be set in the camelContext.xml to execute Batch Processing.

## URI format

```
<camelContext streamCache="true" xmlns="http://camel.apache.org/schema/spring">
```

Or

```
<route streamCache="true">
```

**See also:**

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.6 Using Envelope

- Instead of directly passing input/output parameters to the flow, the contents of the message can be enveloped in an XML message and passed.
- Supports multi-input/output. The Input/output envelope can be use where the number of Input/output parameters are more than one. All the input/output variables can be enveloped in an XML message and then passed to the flow.
- Need to set an option in the URI to use the envelope instead of the actual message. Since you have two separate options to enable envelope, it is possible to use envelope on one side (input/output) alone.

Structure of input and output envelope is document below.

**See also:**

[Apache Camel](#)

[Message Payload](#)

[Testing Volante Components in OSGI-Camel](#)



## 1.6.1 Output Envelope

If the flow has multiple outputs, by passing the “output-as-xml” as an additional option to the flow, an XML envelope containing the output variables defined in the flow is returned.



### URI format

`volante:flowName?output-as-xml=true`

#### Structure of XML Envelope is given below

VolanteParams	Root tag of the returned XML
Output	Enclosing tags for all Output parameters
Param	Tag representing each output parameter in the flow. One param tag is included for each output in the message flow. The text within this tag contains the output value for that parameter.
@name	Name of the output parameter as defined in message flow.
@index	Index of the output parameter as defined in the message flow.
@format	The format type for the output value.
Fault	The text within this tag contains the exception message.

1. For a valid input, the below envelope XML is returned. The number of Param elements would depend on the number of parameters returned from the flow. All the Param elements would have name and index attributes.

 RawOut	OUTPUT	RawMessage
 NumberOfRecords	OUTPUT	Integer

```
<?xml version="1.0" encoding="UTF-8" ?>
<VolanteParams>
  <Output>
    <Param name="RawOut" index="1">123456,B,IBM,80.5,100,20060606
222222,S,IBM,90.0,500,20070102
    </Param>
    <Param name="NumberOfRecords" index="2">2</Param>
  </Output>
</VolanteParams>
```

2. If there are errors when the flow is executed, the execution content is included under the <Fault> tag. The number of TransformExceptions would depend on the number of exceptions encountered.

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

```

<VolanteParams>
  <Fault>
    <TransformException>
      <Type>TransformException</Type>
      <Message>Field 'Order Type' cannot be null.</Message>

      <ErrorCode>SRT129</ErrorCode>
      <Severity>fatal</Severity>
      <Cascadable>true</Cascadable>
      <FieldName>Order Type</FieldName>
      <FieldID>Records[0].Order Type</FieldID>
      <Internal-Code>SRT129</Internal-Code>
      <Error-Record-Index>0</Error-Record-Index>
      <line>1</line>
      <Error-Line>123456,,IBM,80.50,100,20060606</Error-Line>
      <Location>Record</Location>
      <Error-Type>Parsing</Error-Type>
      <Error-Phase>Input</Error-Phase>
      <Trace>at OrderCSV.parse
        at OutputXMLFlow.Parse1(Parse OrderCSV)</Trace>
    </TransformException>
  </Fault>
</VolanteParams>

```

3. If the output is in a format other than String (example Binary), then format tag specifying the format for the output text is displayed.

```

<?xml version="1.0" encoding="UTF-8" ?>
<VolanteParams>
  <Output>
    <Param name="rawOutMessage" index="1"
      format="base64">ewoJImZsZDEiIDogIkFBQUEiLAoJImZsZDIiIDogbnVsbCwKCSJmbGQ
      zIiA6IG51bGwsCgkibmVzdGVkIiA6IG51bGwKfQ==</Param>
    </Output>
  </VolanteParams>

```

#### See also:

[Apache Camel](#)

[Message Payload](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.6.2 Input Envelope

If the flow has multiple inputs, an XML envelope containing all the input variables defined in the flow can be processed by passing "input-as-xml" to the flow. This flow returns the raw message for the given input envelope.



### URI format

`volante:flowName?input-as-xml=true`

Structure of XML Envelope is given below

VolanteParams	Root tag of the returned XML
Input	Enclosing tags for all Input parameters
Param	Tag representing each input parameter in the flow. One param tag is included for each input in the message flow. The text within this tag contains the input value for that parameter.
@name	Name of the input parameter as defined in the message flow.
@index	Index of the input parameter as defined in the message flow.
@format	The format type for the input value.

The input variables are enveloped as shown. The number of Param elements would depend on the number of inputs to the flow. All the Param elements would have name and index attributes.

Field Name	Scope	Type
 RawIn	INPUT	RawMessage
 NumberOfRecords	INPUT	String

```
<?xml version="1.0" encoding="UTF-8" ?>
<VolanteParams>
  <Input>
    <Param name="RawIn" index="1">123456,B,IBM,80.5,100,20060606
222222,S,IBM,90.0,500,20070102
    </Param>
    <Param name="NumberOfRecords" index="2">2</Param>
  </Input>
</VolanteParams>
```

#### See also:

[Apache Camel](#)

[Message Payload](#)

[Testing Volante Components in OSGI-Camel](#)

## 1.7 Using user defined Java Objects

Java Objects can be processed in Camel using a POJO message. A POJO message can be used to convert a user defined Java object into Designer representation (Parsing) which can be processed using Volante flow.

If an unrecognized object is received by the Camel adapter, then is wrapped in a Raw Message and passed to the flow. The flow can use the POJO plugin to parse the message and process the inbound object. Similarly, on the outbound side a user defined object can be returned from the flow by wrapping it in a Raw Message. Refer to the POJO plugin documentation for further details.

If the input for the flow has a POJO message then the input variables defined in the flow can be processed by passing "input-as-pojo" to the flow. This flow returns the raw message for the given POJO input.

### URI format

```
volante:flowName?input-as-pojo=true
```

**Note:** This method is not applicable for flow output.

#### See also:

[Apache Camel](#)

[Testing Volante Components in OSGI-Camel](#)

## 2 Testing Volante Components in OSGI-Camel

### 1. Edit Karaf\bin\Setenv.bat and set the JAVA\_HOME property

```
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0_31
```

### 2. Run karaf

```
>bin\karaf.bat
```

#### Install camel in Karaf

```
features:addurl mvn:org.apache.camel.karaf/apache-
camel/2.10.0/xml/features
features:install camel (need internet connection)
```

#### List bundles

```
karaf@root> list
START LEVEL 100, List Threshold: 50
  ID   State      Blueprint      Spring      Level  Name
[ 84] [Active    ] [             ] [        ] [ 50] camel-core
(2.10.0)
[ 85] [Active    ] [Created      ] [        ] [ 50] camel-karaf-
commands (2.10.0)
[ 92] [Active    ] [             ] [        ] [ 50] geronimo-
jta_1.1_spec (1.1.1)
[ 93] [Active    ] [             ] [        ] [ 50] camel-spring
(2.10.0)
[ 94] [Active    ] [Created      ] [        ] [ 50] camel-blueprint
(2.10.0)
karaf@root>
```

### 3. Install core Volante Jars

```
install -s file:F:/Volante/lib/runtime/generalutils.jar
install -s file:F:/Volante/lib/runtime/transformrt.jar
install -s file:F:/Volante/lib/runtime/jta.jar
install -s file:F:/Volante/lib/runtime/resourcemanager.jar
install -s file:F:/Volante/lib/runtime/asciirt.jar
install -s file:F:/Volante/lib/runtime/volante-osgirt.jar
install -s file:F:/Volante/lib/runtime/volante-client.jar
install -s file:F:/Volante/Integration/camel/camel-volante.jar
```

#### 4. Build the example

```
mvn package
```

The file camel-example-volante-2.10.1.jar would be created in Volante\Integration\camel\camel-volante-spring-example\target folder

#### 5. Deploy application specific Jars (cartridges & example Jar)

```
install -s file:F:/Volante/Integration/camel/camel-volante-spring-  
example/cartridge/java/Order.jar
```

```
install -s file:F:/Volante/Integration/camel/camel-volante-spring-  
example/target/camel-example-volante-2.10.1.jar
```

Copy sample for input apache-karaf-2.3.0\input folder. Verify output in output-normal folder

#### Note:

From camel 2.20.0, camel deprecated and removed spring based camelContext (karaf feature spring-dm) support for OSGi environment. Due to this spring based camel volante bundles will not be deployed.

To deploy latest camel volante bundles in karaf, convert spring based camel context (under META-INF/spring to blueprint based camelContext (under OSGI-INF/blueprint).

#### Example

The below snippet is an existing spring based camel context for camel-volante-spring-example project

```
<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://camel.apache.org/schema/spring">  
    <route>  
      <from uri="file:input"/>  
      <convertBodyTo type="byte[]"/>  
      <to uri="volante:OrderFlow"/>  
    </route>  
  </camelContext>  
</beans>
```

The below snippet is the blueprint based camel context for camel-volante-spring-example project

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <camelContext xmlns="http://camel.apache.org/schema/blueprint">
    <route>
      <from uri="file:input"/>
      <convertBodyTo type="byte[]"/>
      <to uri="volante:OrderFlow"/>
    </route>
  </camelContext>
</blueprint>
```

The blueprint camel context file should be placed under OSGI-INF/blueprint folder.

**See also:**

[Apache Camel](#)

[Volante Component](#)

[Message Payload](#)

[Context Properties](#)

[Message Properties & Attachments](#)

[Batched Processing](#)