Asynchronous message processing with Mule

Asynchronous flows

Setting exchange-pattern of a message source to “one-way” enables asynchronous processing for a flow. Some transports and connectors, like JMS or the VM transport, are asynchronous by default. Other transports which are inherently synchronous, like HTTP, need there exchange pattern explicitly set. Setting one-way exchange patterns on these transports allows you to simulate asynchronous behavior with protocols that would otherwise not be asynchronous.

```
<flow name="HTTP to JMS Flow">
       <http:inbound-endpoint address="http://localhost:8080/foo" exchange-pattern="one-way"/>
       <jms:outbound-endpoint queue="messages"/>
</flow>
```

Asynchronously bridging an HTTP request to JMS

Message aggregation

You can process asynchronously dispatched messages in groups by using the collection-aggregator. Message groups are defined by setting the correlationId property of a MuleMessage or by setting the MULE_CORRELATION_ID outbound header. The correlationGroupSize property of MuleMessage, or the MULE_CORRELATION_GROUP_SIZE header, define the amount of messages in a group.

```
<flow>
 <vm:inbound-endpoint path="foo.bar" exchange-pattern=”one-way”/>
 <collection-aggregator-router timeout="6000" failOnTimeout="false">
   <payload-type-filter expectedType="org.foo.some.Object"/>
 </collection-aggregator-router>
</flow>
```

Asynchronously processing a message group

Message chunking

Split message payloads can be reassembled by using the message-chunking-aggregator-router. By default the message-chunking-aggregator-router will use the correlationId and correlationGroupSize propertis of the MuleMessage for reassembly. You can define an optional “correlationIdExpression” to reassemble with a different message property.

```
<flow>
 <vm:inbound-endpoint path="foo.bar"/>
 <message-chunking-aggregator-router>
   <expression-message-info-mapping correlationIdExpression="#[header:INBOUND:myCustomHeader]"/>
   <payload-type-filter expectedType="org.foo.some.Object"/>
 </message-chunking-aggregator-router>
</flow>
```

Split a java.util.List and route to a JMS queue based on type

Message splitting

Some message payloads, like collections or XML documents, can be split and dispatched asynchronously. Available message splitters:

`list-message-splitter-router`	Split a list
`expression-splitter-router`	Split a message using the Mule Expression Language
`mulexml:filter-based-splitter`	Split an XML document based on an XPath Expression

```
<outbound>
 <list-message-splitter-router>
   <jms:outbound-endpoint queue="order.queue">
     <payload-type-filter expectedType="com.foo.Order"/>
   </jms:outbound-endpoint>
   <jms:outbound-endpoint queue="item.queue">
     <payload-type-filter expectedType="com.foo.Item"/>
   </jms:outbound-endpoint>
   <payload-type-filter expectedType="java.util.List"/>
 </list-message-splitter-router>
</outbound>
```

Asynchronously bridging an HTTP request to JMS

Tuning

Asynchronous processing for a flow can be tuned by defining a queued-asynchronous-processing-strategy.

Multiple queued-asynchronous-processing-strategy can be defined and set using the flow's "processingStrategy" attribute.

`maxBufferSize`	Determines how many requests are queued when the pool reaches maximum capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.
`maxQueueSize`	The maximum number of messages that can be queued
`maxThreads`	The maximum number of threads that can be used.
`minThreads`	The number of idle threads kept in the pool when there is no load.
`poolExhaustedAction`	When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Either "WAIT", "DISCARD", "DISCARD_OLDEST", "ABORT" or "RUN"
`queueTimeout`	The timeout used when taking events from the queue.
`threadTTL`	Determines how long an inactive thread is kept in the pool before being discarded.
`threadWaitTimeout`	How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, the wait is infinite.

```
<queued-asynchronous-processing-strategy name="allow500Threads" maxThreads="500"/>
<flow name="acceptOrders" processingStrategy="allow500Threads">
 <vm:inbound-endpoint path="acceptOrders" exchange-pattern="one-way"/>
 <vm:outbound-endpoint path="commonProcessing" exchange-pattern="one-way"/>
</flow>
```

Configuring a flow to use up to 500 threads to asynchronous process messages arriving a VM inbound-endpoint