AnswerHub | Social Q&A for the Enterprise

½ of the Top 10 StackExchange 1.0 Sites Now Run on AnswerHub

Discover Why Now!

DZone Refcardz

CONTENTS INCLUDE:

Updated for Mule 3.3!

Essential Mule 3.3

Simplifying SOA

By John D'Emic

ABOUT MULE

Mule is the world's most widely used open source integration platform and Enterprise Services Bus (ESB). Inspired by the seminal Enterprise Integration Patterns, Mule is designed to support high-performance, multi-protocol transactions between heterogeneous systems and services. It provides the basis for service-oriented architecture.

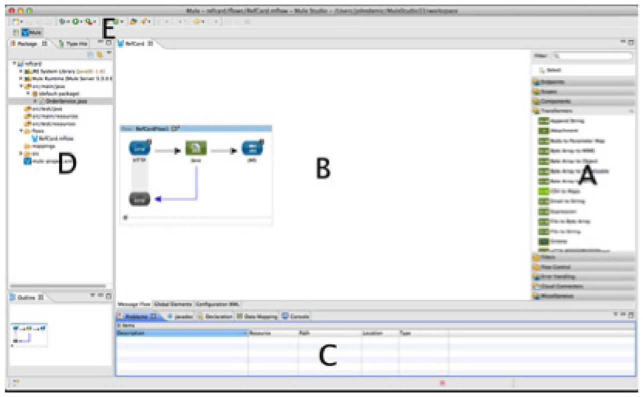
This Refcard covers the use of Mule 3.3. For new users it will serve as a handy reference when building your integration flows in Mule. For existing users of Mule, especially users of previous versions of Mule 3, it will highlight the new features available in Mule 3.3.

WHAT'S NEW IN MULE 3.3?

- Graphical data transformation with DataMapper
- Mule Expression Language, an MVEL based, unified expression language
- Pattern-based exception handling
- Simplified iteration over data structures in message payloads

BUILDING INTEGRATION APPLICATIONS WITH MULE

Mule 3.3 provides a powerful, Eclipse-based authoring environment for developing integration applications.



Mule studio allows you to round-trip between the graphical view of your application and the corresponding configuration XML.

A: The Message Processor pallet displays the available Message Processors for you to use in your flows. They are grouped by function. You can use the filter at the top of the pallet to search for message processors by name.

B: Mule integration flows are built by dragging message processors from the pallet view to the flow view. You can switch between the Message Flow and the Configuration XML at the bottom of this pane.. In this area you can also see Global Elements, like JMS connector configurations.

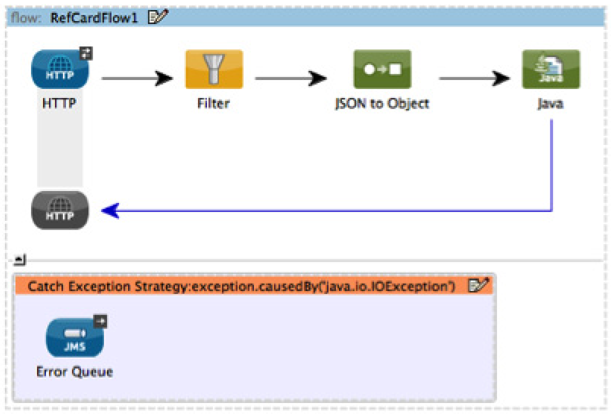
C: These tabs display various status about your Mule project, including any errors, the embedded Mule instance's console log, access to Javadoc and JUnit test results. Its also the place where you can define and edit your transformations for DataMapper.

D: This pane displays the Mule project's directory structure, including the XML configuration files and any Java classes required by your application.

E: You can run and debug your application from here.

Flows

Flows provide a free-form method of orchestrating message processing in Mule. A flow consists of a message source, typically an inbound-endpoint, followed by a sequence of message processors. Message processors, like filters, transformers, or Java components, process a message as it passes through the flow.

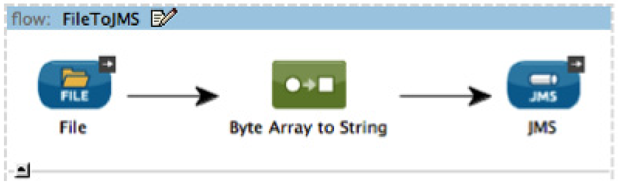

flow: RefCardFlow1
HTTP
Filter
JSON to Object
Java
HTTP
Catch Exception Strategy:exception.causedBy('java.io.IOException')
JMS
Error Queue

An exception strategy can be added to a flow to handle errors that occur during the flow's execution.

Hot Tip End a flow with a router or endpoint to send the message to another flow or external service.

Sending a JMS Message with a Flow

Sending a JMS message is easy with a flow. Here's how you can use a flow to read files from a directory and send their payload to a JMS queue.

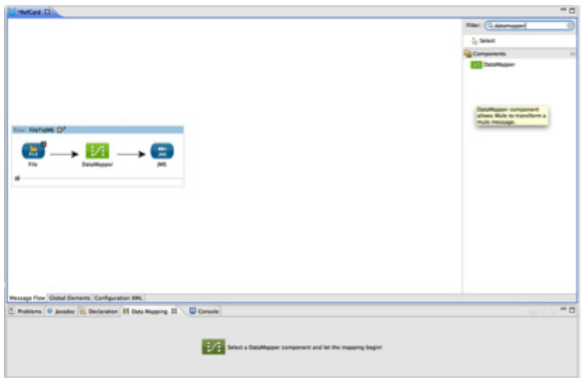


```
<flow name="FileToJMS" doc:name="FileToJMS">
    <file:inbound-endpoint path="/opt/files/in"
responseTimeout="10000" doc:name="File"/>
    <byte-array-to-string-transformer doc:name="Byte Array to
String"/>
    <jms:outbound-endpoint queue="files" connector-ref="Active_MQ"
doc:name="JMS"/>
  </flow>
```

This flow uses a file inbound-endpoint to read files from the specified directory path. Each file is then converted to a string by the byte-array-to-string-transformer. The string is then used as the payload of the JMS message to the "files" queue by the JMS outbound-endpoint.

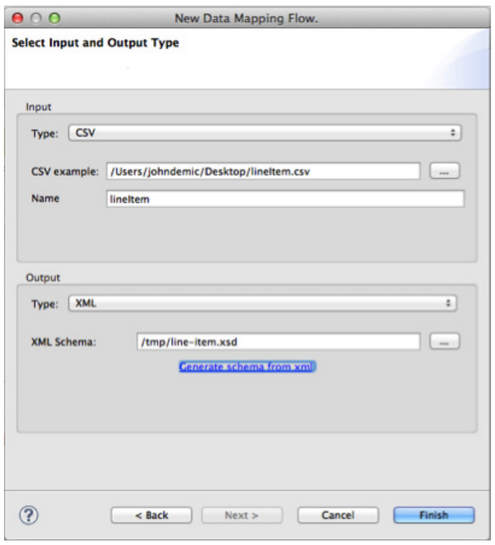
Transforming Payloads with DataMapper

Mule 3.3's DataMapper functionality allows you to easily transform messages from one format to another. To use DataMapper, select the "DataMapper" message processor from the pallet and drag it onto your flow. Then select the DataMapper and launch the "DataMapper Flow Wizard" to define your transformation.



Transforming CSV to XML

Transforming from CSV to XML is simple with DataMapper. After launching the "DataMapper Flow Wizard" the Input Type is set to CSV and the Output Type is set to XML. The CSV Example and XML Schema specify the format of the input and output to the DataMapper.



New Data Mapping Flow.

Select Input and Output Type

Input

Type: CSV

CSV example: /Users/johndemic/Desktop/lineItem.csv

Name: lineItem

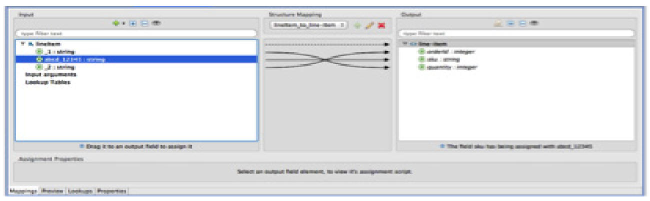
Output

Type: XML

XML Schema: /tmp/line-item.xsd

Hot Tip The fields from the source format can be clicked on and dragged to the output format.

In addition to XML and CSV, DataMapper also supports POJO, JSON, Maps and Excel spreadsheets.



MESSAGES

Messages encapsulate data entering and leaving Mule. The content of a message is called its payload. The payload is typically a Serializable Java class, an InputStream or an array of bytes.

Attachments

A message can have zero or more MIME attachments in addition to the payload. These can be used to associate files, documents and images with the message.

Properties

Properties, also called headers, are metadata associated with a message. Mule, the various transports, and you, the developer, can add properties to messages. Examples of message properties are JMS message headers, HTTP response headers, or Mule-specific headers like MULE_MESSAGE_ID. The following table contains examples of message properties set by Mule.

Property	Description
MULE_MESSAGE_ID	A GUID assigned to the message.
MULE_CORRELATION_ID	A GUID assigned to a group of messages.
MULE_CORRELATION_GROUP_SIZE	The amount of messages expected in the correlation group.
MULE_CORRELATION_SEQUENCE	The order of a correlation group.
MULE_SESSION	A GUID indicating the session the message belongs to

Scopes

Properties are scoped differently depending on when they're set or accessed during message processing. The following table contains the available scopes.

Scope	Description
inbound	Set by message sources, typically an inbound-endpoint.
outbound	Set on messages leaving a message processor. Properties set by the message-properties-transformer default to the outbound scope.
session	Properties in the session scope are available between processors and services without explicit propagation.
invocation	nvocation properties, or flow variables, contain data that is accessible to a message as it passes through a flow.

Hot Tip Message properties leaving a processor on the outbound scope are available in the inbound scope on the subsequent processor.

CONNECTIVITY

Mule connects to more than 100 applications, protocols and APIs. Mule endpoints enable connectivity to protocols, such as JMS, HTTP and JDBC. Cloud Connectors enable connectivity to applications and social media like SalesForce and Twitter.

Endpoints

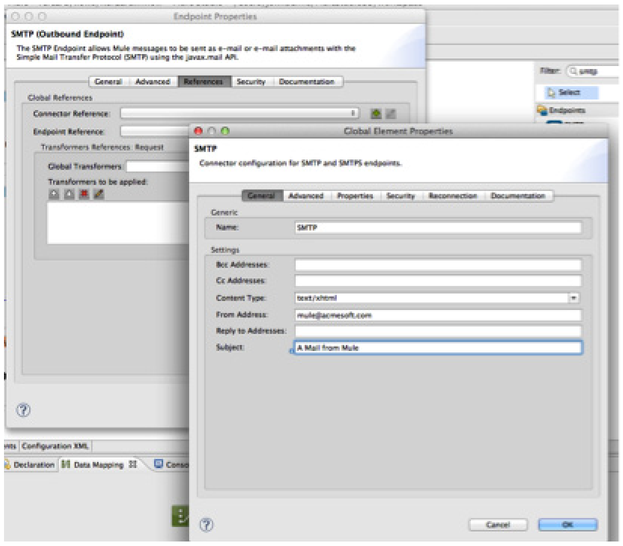
Messages can be received with an inbound endpoint and sent with an outbound endpoint.

Connectors

A connector is used to configure connection properties for an endpoint. Most endpoints don't require a connector, but some like JDBC or JMS, do require connector configuration, as we'll see next.

Configuring an SMTP connector

The following example illustrates how an SMTP connector is configured in Mule Studio as well as in XML.



Endpoint Properties

SMTP (Outbound Endpoint)

The SMTP Endpoint allows Mule messages to be sent as e-mail or e-mail attachments with the Simple Mail Transfer Protocol (SMTP) using the javax.mail API.

Global Element Properties

SMTP

Connector configuration for SMTP and SMTPS endpoints.

Name:	SMTP
Bcc Addresses:	
Cc Addresses:	
Content Type:	text/xhtml
From Address:	mule@acmesoft.com
Reply to Addresses:	
Subject:	A Mail from Mule

```
<smtp:connector name="SMTP" contentType="text/xhtml"
fromAddress="mule@acmesoft.com" subject="A Mail from Mule"
doc:name="SMTP"/>
```

The SMTP connector allows you to specify properties that will be shared across SMTP endpoints. In this case, the connector sets the Content-Type and "from" address as well as the subject of the messages. A connector is referenced by its name, allowing you to define multiple connectors for the same transport.

Hot Tip Endpoints can be generically referenced using an endpoint URI.

The following table contains some common endpoints supplied by Mule.

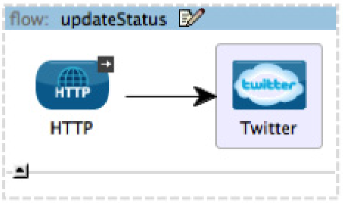
	Endpoint	Description
HTTP	http:// [host]:[port]:[path]?[query]	Send and receive data over HTTP.
AJAX	ajax://[channel]	Pub / Sub to browser apps using CometD.
File	File://[path]	Read and write files.
S/FTP	ftp:// [user]@[host]:[port]/[path]	Read and write files over FTP or SFTP.
JMS	jms:// [type]:[destination]?[options]	Full support for JMS topics and queues.
SMTP	smtp://[user]@[host]:[port]	Send email over SMTP.
IMAP	Imap:// [user]@[host]:[port]/[folder]	Receive email via IMAP
JDBC	jdbc://[sql query]	Send and receive data from a SQL database.
VM	vm://[path]	Uses memory-based queues to send messages between services and flows.

The full list of transports is available in the Mule documentation.

Hot Tip Use exchange patterns to define how a message is received by an endpoint. For endpoints that generate a response (synchronous), use the request-response. For asynchronous endpoints, use the one-way exchange pattern.

Cloud Connectors

Introduced in Mule 3, cloud connectors enable easy access to SaaS, social media and infrastructure services, such as Twilio and Facebook.
Cloud Connectors can be used anywhere in a flow to invoke a remote service. A cloud connector usually has a 'config' element where service credentials are set and one or more elements that invoke a service method. The following will make it possible to publish a tweet using curl:http://localhost?status=gomule!

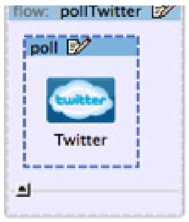


```
<twitter:config name="twitter" format="JSON"
  accessKey="${twitter.consumer.key}"
  accessSecret="${twitter.consumer.secret}"
  oauthToken="${twitter.access.token}"
  oauthTokenSecret="${twitter.access.secret}" />
```

```
<flow name="updateStatus" doc:name="updateStatus">
    <http:inbound-endpoint exchange-pattern="one-way"
host="localhost" port="8081" doc:name="HTTP"/>
    <twitter:update-status config-ref="Twitter"
status="#[payload]" doc:name="Twitter"/>
  </flow>
```

Polling

Mule has a poll tag that allows data from a remote service of a Cloud Connector to be received periodically. To get updates from a Twitter timeline every minute:



```
<flow name="pollTwitter" doc:name="pollTwitter">
    <poll frequency="60000">
      <twitter:get-home-timeline config-ref="Twitter"
doc:name="Twitter"/>
    </poll>
  </flow>
```

MODULES

Modules extend Mule's functionality by providing namespace support for a certain set of message processors. The following table contains some of the modules provided by Mule.

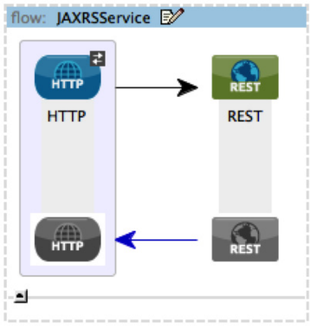
Module	Description
JSON	JSON support, including marshalling, transformation and filtering.
CXF	SOAP support via Apache CXF.
Jersey	JAX-RS support for publishing RESTful services.
Scripting	Support for JSR-223 compliant scripting language, like Groovy or Rhino.
XML	XML support, including XML marshalling, XPath and XSLT support.

The full list of available modules is in the official Mule documentation. Additional modules are available on MuleForge.

Hot Tip Use MuleForge to locate community written extensions.

Hosting a JAX-RS Web Service

The following demonstrates how the Jersey module can be used to host a JAX-RS annotation service classusing Mule.



```
<flow name="JAXRSService" doc:name="JAXRSService">
    <http:inbound-endpoint exchange-pattern="request-response"
host="localhost" port="8080" path="orders" doc:name="HTTP"/>
    <jersey:resources doc:name="REST">
      <component class="com.acmesoft.service.OrderService"/>
    </jersey:resources>
  </flow>
```

MESSAGE PROCESSORS

Message Processors are used in flows to route, transform, filter and perform business logic on messages.

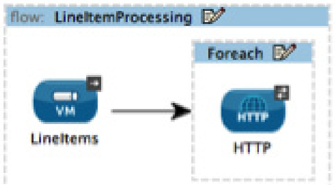
Flow Control

Flow Control Message Processors, or routers, implement the popular Enterprise Integration patterns (EIP) and determine how messages are directed in a flow.

The following table contains commonly used routers.

Router	Description
all	JSON support, including marshalling, transformation and filtering.
choice	Send the message to the first endpoint that matches.
round-robin	Each message received by the router is sent to alternating endpoints.
wire-tap	Sends a copy of the message to the supplied endpoint, then passes the original message to the next processor in the chain.
first-successful	Sends the message to the first endpoint that doesn't throw an exception or evaluates the failureExpression to true.
until-successful	Redelivers a message until it's successfully delivered or gives up after a certain amount of attempts.
foreach	Iterates over a collection in the payload of a message.

The following flow demonstrates how the Foreach processor can iterate over a collection present in a message payload. This flow accepts a List of LineItem objects and sends each to an outbound HTTP endpoint for processing.



```
 <flow name="LineItemProcessing" doc:name="LineItemProcessing">
    <vm:inbound-endpoint exchange-pattern="one-way"
path="lineItems" doc:name="LineItems"/>
    <foreach collection="payload" doc:name="Foreach">
      <http:outbound-endpoint exchange-pattern="request-response"
host="localhost" port="8081" doc:name="HTTP"/>
    </foreach>
  </flow>
```

Transformers

Transformers modify the message and pass it to the next message in the chain. The following table contains commonly used transformers.

Name	Description
message-properties-transformer	Add and remove properties from a message, optionally specifying their scope when different from the default outbound scope.
byte-array-to-string-transformer	Transforms a byte array to a String.
byte-array-to-object-transformer	Transforms a byte array to an Object.
xml:object-to-xml	Using XStream, this transforms message payloads to and from XML.
xml:xslt-transformer	Transforms a message using the given stylesheet.
json:object-to-json-transformer	Using Jackson, this transforms message payloads to and from JSON.

Hot Tip Endpoints often include their own transformers. JMS, for instance, provides transformers to convert message payloads to and from JMS messages automatically.

Components

Components allow business logic to be executed in a flow. Any Java object or script can be used as a component. Components are configured by either identifying the class or providing a reference to a Spring bean for dependency injection.

The following snippet shows how a class called MyService can be configured as a component using a class and via dependency injection via Spring.

```
<bean id="myService" class="com.acmesoft.service.MyService"/>

<flow name="test">
    <http:inbound-endpoint host="foo.com">
    <component>
        <spring-object bean="myService"/>
    </component>
</flow>
```

Mule will use the type of payload that is in the message being processed to determine what method to invoke. It's often necessary, however, to explicitly specify the method to invoke. Entry point resolvers are used for this purpose. The following table contains a list of available resolvers.

Resolver	Description
method-entry-point-resolver	Resolves the method using the specified name.
property-entry-point-resolver	Resolves the method using the specified message property.
custom-entry-point-resolver	A Java class that implements org.mule.api.model. EntryPointResolver or extends org.mule.model.resolvers. AbstractEntryPointResolver.

The use of entry point resolvers allows you to use POJO's as components, decoupling your code from Mule.

Sometimes, though, you will need to operate on more then just a message's payload. Mule's annotations give your components runtime access to a MuleMessage without coupling your component code at compile time to Mule's API. The following table contains a list of commonly used annotations.

Name	Type	Description
@Payload	Parameter	Can be specified on the component entry point and transformer method parameters to show the parameter that indicates the message payload.
@InboundHeaders	Parameter	Specifies the component-entry-point or transformer-method parameter that the inbound headers should be mapped to.
@OutboundHeaders	Parameter	Specifies the component-entry-point or transformer-method parameter that the outbound headers should be mapped to.
@InboundAttachments	Parameter	Specifies the component-entry-point or transformer-method parameter that the inbound attachments should be mapped to.
@OutboundAttachments	Parameter	Specifies the component-entry-point or transformer-method parameter thatthe outbound attachments should be mapped to.

Implementing a Component with Annotations

Here's an example of a component that accesses the message's payload and an inbound header with annotations:

```
public class LineItemService {
    public void process(@Payload Object lineItem,
                        @InboundHeaders("LINE_ITEM_PRIORITY")
String priority) {
        // perform processing
    }
}
```

MULE EXPRESSION LANGUAGE

Mule provides a rich expression language based on MVEL to evaluate data at runtime using the message currently being processed.

Context Objects

The following are commonly used variables on context objects available in MEL expressions.

Name	Description
message	The MuleMessage that gives you access to the payload, the i.d., and the various properties.
flowVars	The flow variables, or invocation properties, available on the flow.
sessionVars	The session variables, or session properties, available on the flow.
server	Information about the server Mule is running on, including its FQDN.
mule	Information about the current Mule instance, including its home.
app	Information about the currently running Mule application, including its name.

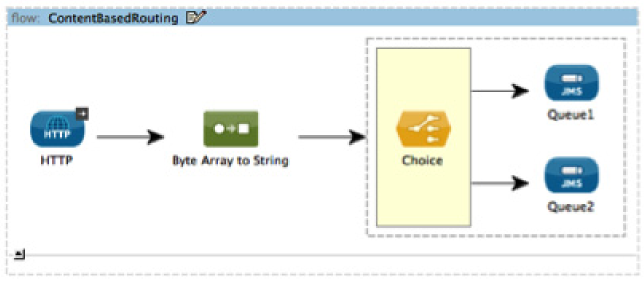
Here are some examples of Mule Expressions

Name	Description
message.inboundProperties['filename'].endsWith('.jpg')	Check if the inbound property 'filename' ends with 'jpg.
<logger message="File Received (size = #[message.inboundProperties['fileSize']/1024] kb)" level="INFO" />	Embed an expression in a logger's message.
xpath('/order/@type') == 'book'	Evaluate an XPath expression.
regex('^(To|From|Cc):')	Evaluate a regular expression.

Content Based Routing and Filtering

The Mule Expression Language enables Mule to perform content-based routing and filtering.

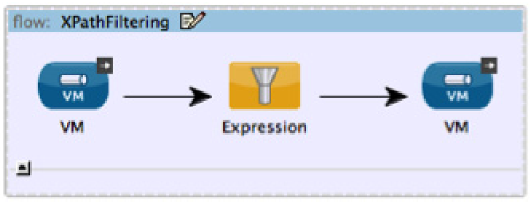
The following illustrates how a message is dynamically routed to a JMS queue by using MEL to evaluate a regular expression against the message's payload.



```
<flow name="ContentBasedRouting" doc:name="ContentBasedRouting">
    <http:inbound-endpoint exchange-pattern="one-way"
host="localhost" port="8081" path="app" doc:name="HTTP"/>
    <byte-array-to-string-transformer doc:name="Byte Array to
String"/>
    <choice doc:name="Choice">
      <when expression="regex('^LineItem') != null">
        <processor-chain>
          <jms:outbound-endpoint queue="queue1" connector-
ref="Active_MQ" doc:name="Queue1"/>
        </processor-chain>
      </when>
      <otherwise>
        <processor-chain>
          <jms:outbound-endpoint queue="queue2" connector-
ref="Active_MQ" doc:name="Queue2"/>
        </processor-chain>
      </otherwise>
    </choice>
  </flow>
```

Using Filters with XPath

The following example demonstrates how the expression filter can be used to only pass certain XML documents. In this case, only ordered XML documents containing a certain ZIP code are allowed to pass.



```
  <flow name="XPathFiltering" doc:name="XPathFiltering">
    <vm:inbound-endpoint exchange-pattern="one-way" path="in"
        doc:name="VM"/>
    <expression-filter expression="xpath('/order/@type').text ==
'book'"
        doc:name="Expression"/>
    <vm:outbound-endpoint exchange-pattern="one-way"
        path="out" doc:name="VM"/>
</flow>
```

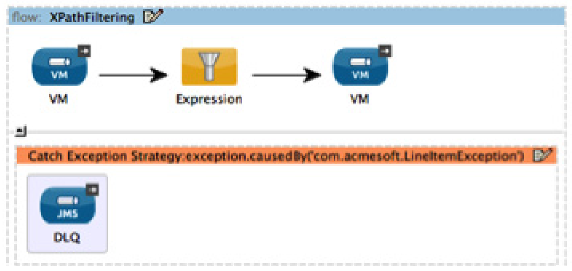
HANDLING ERRORS

Exceptions thrown during message processing are handled by exception strategies. Exception handling has been revamped for Mule 3.3. The available exception strategies are enumerated below.

default-exception-strategy	The default exception strategy used by all flows when an explicit exception strategy isn't defined.
catch-exception-strategy	Selectively handles exceptions based on type.
choice-exception-strategy	Selectively handles exceptions based on an MEL evaluation.
reference-exception-strategy	References an externally defined global exception strategy.
rollback-exception-strategy	Attempts to roll back a message when an exception is thrown.

Catching an Exception

The following example will catch exceptions of the com.acmesoft.LineItemException type and will route them to a JMS queue.


flow: XPathFiltering
VM
Expression
VM
Catch Exception Strategy:exception.causedBy('com.acmesoft.LineItemException')
JMS
DLQ

```
  <flow name="XPathFiltering" doc:name="XPathFiltering">
    <vm:inbound-endpoint exchange-pattern="one-way"
         path="in"   doc:name="VM"/>
     <expression-filter expression="xpath('/order/@type').text ==
'book'"
        doc:name="Expression"/>
    <vm:outbound-endpoint exchange-pattern="one-way" path="out"
        doc:name="VM"/>
    <catch-exception-strategy when="exception.causedBy('com.
acmesoft.LineItemException')"
        doc:name="Catch Exception Strategy">
        <jms:outbound-endpoint queue="dq" connector-ref="Active_
MQ" doc:name="DLQ"/>
     </catch-exception-strategy>
</flow>
```

Hot Tip Exceptions routed by an exception strategy are instances of org.mule.api.message.ExceptionMessage, which gives you access to the Exception that was thrown with the payload of the message.

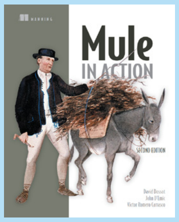
CONCLUSION

This RefCard is just a glimpse at the capabilities of Mule 3.3. The complete documentation for Mule 3.3 is available in the Mule User Guide.

ABOUT THE AUTHORS

John D'Emic is a software developer and author. He is the co-author of both editions of Mule in Action and is currently a Solutions Architect at MuleSoft, Inc. You can see what John's up to by following his Twitter account: @johndemic

RECOMMENDED BOOK



Mule in Action, Second Edition is a totally-revised guide covering Mule 3 fundamentals and best practices. It starts with a quick ESB overview and then dives into rich examples covering core concepts like sending, receiving, routing, and transforming data. You'll get a close look at Mule's standard components and how to roll out custom ones. You'll also pick up techniques for testing, performance tuning, BPM orchestration, and explore cloud API integration for SaaS applications.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome

refcardz@dzone.com

Sponsorship Opportunities

sales@dzone.com

ISBN-13: 978-1-936502-55-4
ISBN-10: 1-936502-55-0

50795

9 781936 502554

$7.95

Version 1.0