MuleSoft Reference Cards

Mule Expression Language (MEL)

Server, Mule, Application and Message variables

Overview of the global contexts and the variables they give access to.

server	mule	app	message
fileSeparator	clusterId	encoding	id
host	home	name	rootId
ip	nodeId	standalone	correlationId
locale	version	workdir	correlationSequence
javaVersion		registry	correlationGroupSize
javaVendor			replyTo
osName			dataType
osArch			payload
osVersion			inboundProperties
systemProperties			inboundAttachments
timeZone			outboundProperties
tmpDir			outboundAttachments
userName			exception
userHome			
userDir			

Create a directory named `target` in the system's temporary directory and set it as the current payload:

```
<expression-component>
    targetDir = new java.io.File(server.tmpDir, 'target');
    targetDir.mkdir();
    payload = targetDir
</expression-component>
```

Set the username and password for an HTTP endpoint at runtime based on inbound message properties:

```
<https:outbound-endpoint address="https://#[message.inboundProperties.username]:↩
               #[message.inboundProperties.password]@api.acme.com/v1/users" />
```

Flow and session variables

Flow variables are available in the `flowVars` context or directly as top level variables (unless `autoResolveVariables` is false or their name violates the MVEL naming conventions). Session variables are available in the `sessionVars` context.

To create a session variable named `sessionId` whose value is the concatenation of the current message ID, "@" and the Mule instance node ID, use either `set-session-variable` or MEL code, both shown hereafter:

```
<set-session-variable variableName="sessionId" value="#[message.id+'@'+mule.nodeId]" />
<expression-component>sessionVars.sessionId = message.id+'@'+mule.nodeId;</expression-component>
```

The following shows how a bean can be dynamically retrieved from the registry, based on a name created by taking a value from a bean payload with a `getTargetService()` accessor:

```
<set-variable variableName="beanName" value="#[message.payload.targetService+'Processor']" />
<set-variable variableName="bean" value="#[app.registry[beanName]]" />
```

Payload and attachments

Copy the current payload in a flow variable named `originalPayload` then restore it:

```
<set-variable variableName="originalPayload" value="#[message.payload]" />
<set-payload value="#[originalPayload]" />
```

To retrieve the message payload in a particular format, using Mule's auto-transformation capability, use `payloadAs`:

```
<logger message="#[message.payloadAs(java.lang.String)]" />
```

To extract all `*.txt` and `*.xml` attachments, use a filtered projection:

```
<expression-transformer expression="#[($.value in ↩
         message.inboundAttachments.entrySet() if $.key ~= '(.*\\.txt|.*\\.xml)')]" />
```

Regex support

Regular expression helper functions retrieve `null`, a single value or an array of values, depending on matches. The forms that take a `melExpression` argument apply the regex to the result of its evaluation instead of `message.payload`.

```
regex(regularExpression [, melExpression [, matchFlags]])
```

For example to select all the lines of the payload that begin with `To:`, `From:`, or `Cc:` use: `regex('^(To|From|Cc):')`

XPath support

XPath helper functions return DOM4J nodes. By default the XPath expression is evaluated on `message.payload` unless an `xmlElement` is specified:

```
xpath(xPathExpression [, xmlElement])
```

To get the text content of an element or an attribute:

```
#[xpath('//title').text]
#[xpath('//title/@id').value]
```

JSON processing

MEL has no direct support for JSON. The `json-to-object-transformer` can turn a JSON payload into a hierarchy of simple data structures that are easily parsed with MEL. For example, the following uses a filtered projection to build the equivalent of the `$..[?(@.title=='Moby Dick')].price` JSON path expression:

```
<json:json-to-object-transformer returnClass="java.lang.Object" />
<expression-transformer
    expression="#[($.price in message.payload if $.title == 'Moby Dick')[0]]" />
```

More MVEL Goodness

Quick access to the MVEL 2.0 Documentation: http://goo.gl/AjceB

Java interoperability, for example to create a random UUID and use it as an XSL-T parameter:

```
<mulexml:context-property key="transactionId"
                          value="#[java.util.UUID.randomUUID().toString()]" />
```

Safe bean property navigation, for example to retrieve `fullName` only if the `name` object is not null:

```
<set-variable variableName="fullName" value="#[message.payload.?name.fullName]" />
```

Local variable assignment, as in this splitter expression that splits a multi-line payload in rows and drops the first row:

```
<splitter expression="#[rows=StringUtils.split(message.payload,'\n\r');↩
                        ArrayUtils.subarray(rows,1,rows.size())]" />
```

"Elvis" operator, to return the first non-null value of a list of values:

```
#[message.payload.userName or message.payload.userId]
```

Global Configuration

Define global imports, aliases and global functions in the global `configuration` element. Global functions can be loaded from the file system, a URL, or a classpath resource (see `extraFunctions.mvel` below). Flow variables auto-binding can be turned off via the `autoResolveVariables` attribute.

```
<configuration>
  <expression-language autoResolveVariables="false">
    <import class="org.mule.util.StringUtils" />
    <import name="rsu" class="org.apache.commons.lang.RandomStringUtils" />
    <alias name="appName" expression="app.name" />
    <global-functions file="extraFunctions.mvel">
      def reversePayload() { StringUtils.reverse(payload) }
      def randomString(size) { rsu.randomAlphanumeric(size) }
    </global-functions>
  </expression-language>
</configuration>
```