Scripting Languages Context

`ScriptContext` Options

The `JSR-223` scripting language's **`ScriptContext`** is pre-configured with the following attributes all set at **`ENGINE_SCOPE`**.

Attribute	Type	Value
`camelContext`	`org.apache.camel.CamelContext`	The Camel Context.
`context`	`org.apache.camel.CamelContext`	The Camel Context (cannot be used in groovy).
`exchange`	`org.apache.camel.Exchange`	The current Exchange.
`properties`	`org.apache.camel.builder.script.PropertiesFunction`	**Camel 2.9:** Function with a **`resolve`** method to make it easier to use Camels Properties component from scripts. See further below for example.
`request`	`org.apache.camel.Message`	The **`IN`** message.
`response`	`org.apache.camel.Message`	**Deprecated**: The **`OUT`** message. The **`OUT`** message is **`null`** by default. Use the **`IN`** message instead.

See Scripting Languages for the list of languages with explicit DSL support.

Passing Additional Arguments to the `ScriptingEngine`

Available from Camel 2.8

You can provide additional arguments to the **`ScriptingEngine`** using a header on the Camel message with the key **`CamelScriptArguments`**.

Example:

```
public void testArgumentsExample() throws Exception {
    getMockEndpoint("mock:result").expectedMessageCount(0);
    getMockEndpoint("mock:unmatched").expectedMessageCount(1);

    // additional arguments to ScriptEngine
    Map<String, Object> arguments = new HashMap<>();
    arguments.put("foo", "bar");
    arguments.put("baz", 7);

    // those additional arguments is provided as a header on the Camel Message
    template.sendBodyAndHeader("direct:start", "hello", ScriptBuilder.ARGUMENTS, arguments);

    assertMockEndpointsSatisfied();
```

Using Properties Function

Available from Camel 2.9

If you need to use the Properties component from a script to lookup property placeholders, then its a bit cumbersome to do so. For example, to set a header name **`myHeader`** with a value from a property placeholder, whose key is taken from a header named **`foo`**.

```
.setHeader("myHeader").groovy("context.resolvePropertyPlaceholders('{{' + request.headers.get('foo') + '}}')")
```

From **Camel 2.9**: you can now use the properties function and the same example is simpler:

```
.setHeader("myHeader").groovy("properties.resolve(request.headers.get('foo'))")
```

Loading Script From External Resource

Available from Camel 2.11

You can externalize the script and have Camel load it from a resource such as **`classpath:`**, **`file:`**, or **`http:`**. This is done using the following syntax: **`resource:scheme:location`** e.g. to refer to a file on the classpath you can do:

```
.setHeader("myHeader").groovy("resource:classpath:mygroovy.groovy")
```

How to Get the Result from Multiple Statements Script

Available from Camel 2.14

The script engine's eval method returns a **`null`** when it runs a multi-statement script. However, Camel can look up the value of a script's result by using the key **`result`** from the value set. When writing a multi-statement script set the value of the **`result`** variable as the script return value.

textbar = "baz"; # some other statements ... # camel take the result value as the script evaluation result result = body * 2 + 1

Dependencies

To use scripting languages in your camel routes you need to add the a dependency on **`camel-script`** which integrates the JSR-223 scripting engine.

If you use maven you could just add the following to your **`pom.xml`**, substituting the version number for the latest & greatest release (see the download page for the latest versions).

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-script</artifactId>
 <version>x.x.x</version>
</dependency>
```