Scripting

Scripts can be written using Groovy language and referenced inside resource files. The Console provides an editor for editing and saving Groovy script files.

Referencing Script files in Resources

Let's create a script file and save it with the name `hello` as follows:

hello.groovy

```
response['message'] = 'Hello World';
```

This file can be referenced in the resource XML using a <Script> tag as follows:

script.xml

```
<Request method="GET">
    <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="firstscript" output="true">
                <Arg name="file" value="hello.groovy" />
        </Execute>
</Request>
```

`Response array`

When we make a GET request to the `script` resource given above, the output obtained as follows:

```
{
   "firstscript": {
      "message":"Hello World"
    }
}
```

`Here, we can observer that the values assigned to the response array in the script are printed in the response output.`

Executing functions

Groovy scripts can be used to create functions. Below is a simple factorial function example.

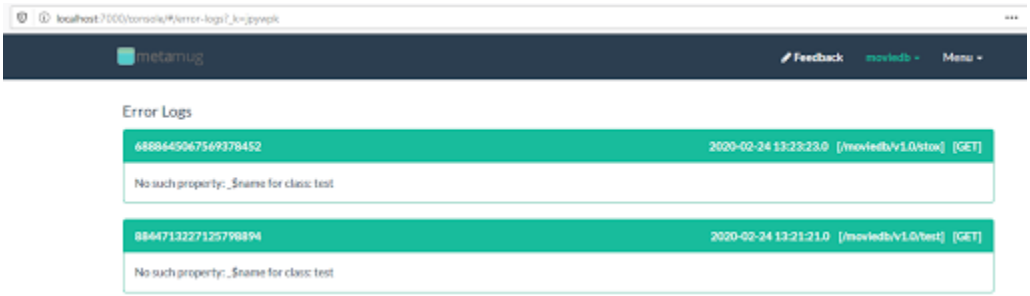
```
def fact(n){
        if(n < 1)
                return 1;
          else
              return fact(n-1)*n;
}

response["output"]=fact(5)
```

> *Note here `print` will not work, like a regular script. The function call must be passed to `response` object.*

Accessing Request parameters

All the request variables can be accessed with _$variable. If the variable is not present in the incoming the request you will get the following error




```
response["message"] = 'Hello ' + _$name;
```

Internal Elements with MPath

In order to access internal elements of the request using MPath. The MPath notation is used.

```
def sqlOutput = _$["sqlElement"].rows[2].name
```

Adding Script to Resource

In order to add a script to your request flow. The script needs to be added as an element inside `Request` tag.

Resource XML

hello.xml

```
<Request method="GET">
        <Desc> Greet with Hello </Desc>
    <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="msg" output="true">
                <Arg name="file" value="greet.groovy" />
        </Execute>
</Request>
```

greet.groovy

```
response['msg'] = 'Hello ' + _$name;
```

The name needs to be passed as a query parameter in case of a GET request

`/script?name=John`

The response object is converted into a json object. In this example, `msg` is assigned with the greeting message

```
{ "script": { "msg": "Hello John" } }
```

Query to Script

One of the regular use-cases of scripts can be post-processing an SQL Result. Groovy has access to all the previous elements defined in the request with MPath.

qrytoscript.xml

```
<Request method="GET">
        <Sql id="q"> SELECT * from movie </Sql>
          <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="greet" output="true">
                <Arg name="file" value="qtos.groovy" />
        </Execute>
</Request>
```

qtos.groovy

we can use `_$[“q“].rows[2].name` to access the name attribute.

```
response['message'] = 'Hello ' +  _$["q"].rows[2].name;
```

Since the query element q is a request element, no parameter is expected in the GET request.

```
/qrytoscript
```

The above request will result in following response

```
{
  "q": [
    {"name":"Superman","rating":2,"id":1},
    {"name":"Spiderman","rating":4,"id":2},
    {"name":"Batman","rating":4,"id":5},
    {"name":"I am legend","rating":5,"id":8}
  ],
  "greet": {"message":"Hello Batman"}
}
```

XRequest to Script

When integrating external APIs, scripts can help add additional processing. No need to upload projects or write extensive code. Formatting, conversion between APIs can be easily handled with scripting.

xreqtoscript.xml

In the below request we have added a script tag after the XRequest.

```
<Request method="GET">
        <XRequest id="xreq" url="https://postman-echo.com/get?foo1=Hello&foo2=World"
                          method="GET" output="true" >
        <Header name="Accept" value="application/json" />
    </XRequest>
        <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="greet" output="true">
                <Arg name="file" value="xtos.groovy" />
        </Execute>
</Request>
```

xtos.groovy

Here since the returned object is a JSON object we use `getJSONObject` method to access the element.

```
response['message'] = 'Hello ' + _$['xreq'].getJSONObject('args').getString('foo2');
```

Request and Response

`/xreqtoscript`

```
{
   "xreq": {

     "args": {"foo1":"Hello","foo2":"World"},
     "headers": {"x-forwarded-proto":"https","host":"postman-echo.com","x-forwarded-port":"443","accept-
encoding":"gzip","accept":"application/json","user-agent":"okhttp/3.10.0"},

     "url":"https://postman-echo.com/get?foo1=Hello&foo2=World"
   },

   "greet": { "message": "Hello World" }
}
```

Using MPath to access Script Output

In the resource XML, standard MPath notation can be used to access script output. In case we want to access script information in a query or XRequest tag.

The below section must be added to MPath documentation

Script to Query

scripttoqry.xml

```
<Request method="GET">
    <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="script" output="true">
                <Arg name="file" value="test.groovy" />
        </Execute>
        <Sql id="q"> SELECT $[script].message as greeting </Sql>
</Request>
```

test.groovy

```
response['message'] = 'Hello ' + _$name;
```

Sending the below request will result in the following response.

`/scripttoqry?name=John`

```
{
        "q": [
                        {"greet":"Hello John"}
                ],
    "script": {
            "message":"Hello John"
    }
}
```

Script to Xrequest

scripttoxreq.xml

```
<Request method="GET">
        <Execute classname="com.metamug.mason.plugin.GroovyRunner" id="script" output="true">
                <Arg name="file" value="test.groovy" />
        </Execute>
        <XRequest id="xreq" url="https://postman-echo.com/post"
                          method="POST" output="true" >
        <Header name="Content-Type" value="application/json" />
        <Body>
                  {
                "foo1": "Welcome",
                "foo2": "$[script].message"
            }
        </Body>
    </XRequest>
</Request>
```

test.groovy

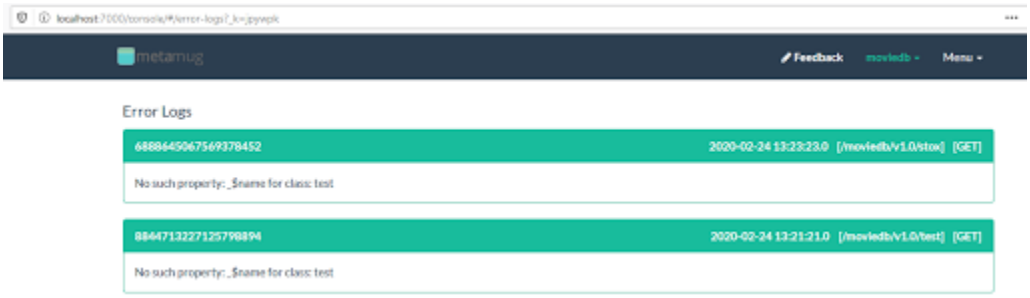
```
response['message'] = 'Hello ' + _$name;
```

Sending the below request will result in the following response.

`/scripttoxreq?name=Anish`

```
{
    "script": {"message":"Hello Anish"},
    "xreq":{"args":{},"headers":{"content-length":"97","x-forwarded-proto":"https","host":"postman-echo.com","x-
forwarded-port":"443","content-type":"application/json; charset=utf-8","accept-encoding":"gzip","user-agent":"
okhttp/3.10.0"},"data":{"foo1":"Welcome","foo2":"Hello Anish"},"form":{},"files":{},"json":{"foo1":"Welcome","
foo2":"Hello Anish"},"url":"https://postman-echo.com/post"}
}
```

Treating _$variable as a request parameter



now imagine if we get an error for _$name. all _$variables are actually coming from the request. So it makes perfect sense to have throw 412 client error. How can we do this for a script. We have achieved it for xml

Throwing error in Script

If the script needs to be halted, exception can be thrown. The thrown exception is visible in the error log. This is helpful in debugging error scenarios.

```
if (orderId) {
   // Work with orderId
}else{
     throw new IllegalArgumentException("The number ${orderId} cannot be empty")
}
```

If exception is not thrown early, the script might fail later without details of the exception.