**karate-config.js**

This is where you can create variables which have a global scope. Karate reads this file before executing any scenario. This comes in very handy when switching environments which specific variables are used for different environments

function() {

var env = karate.env; // get java system property 'karate.env'

karate.log('karate.env selected environment was:', env);

karate.configure("ssl", true)

if (!env) {

env = 'dev'; //env can be anything: dev, qa, staging, etc.

}

var config = {

env: env,

EP\_USERNAME: 'devuser',

EP\_PASSWORD: 'devpass',

EP\_HOST: 'https://am.'+env+'.example.net',

EP\_AUTHENTICATE\_PATH: '/am/json/realms/root/authenticate',

};

if(env == 'qa') {

config.EP\_USERNAME: 'myUserName'

config.EP\_PASSWORD: 'myPa55word'

}

karate.log('OpenAM Host:', config.EP\_HOST);

karate.configure('connectTimeout', 60000);

karate.configure('readTimeout', 60000);

return config;

}

How to send an HTTP Request (Get, Post, Put, Delete, Patch)

Feature: EP elibigle test

Scenario: Login EPeligible system and get eligiblefield

Given header X-OpenEP-Username = EP\_USERNAME

Given header X-OpenEP-Password = EP\_PASSWORD

Given url EP\_HOST + EP\_AUTHENTICATE\_PATH

And request ''

When method POST

Then status 200

\* assert response.eligible != null

\* def iseligible = response.eligible

In the above example, EP\_USERNAME, EP\_PASSWORD, EP\_HOST, and EP\_AUTHENTICATE\_PATH come from the karate-config.js file.

‘**\***‘ can be interpreted as any of Given, When, Then, And, but when an action doesn’t serve a context, we can use ‘\*’.

‘**+**‘ acts as a concatenate operator

The above example sends an empty post body request. We can just use ‘ ‘

The method can be any valid HTTP verb (Get, Post, Put, Patch, Delete)

‘**def**‘ is used to store a value in a variable.

**header**, **url**, **request**, **method**, **status**, **response** are all karate’s keywords forming the DSL. For the full list of keywords, visit Intuit.

In the above example, the response is JSON format, so we can use karate’s builtin JsonPath notation to parse the response.

**Request Chaining with multiple API calls**

Feature: request chaining with multiple api calls

Scenario: chain request demo

\* json req = read('classpath:com/example/templates/idm/post-transaction-template.json')

\* def accid = req.givenid

Given header X-Username = 'anonymous'

Given header X-Password = 'anonymous'

Given url EP\_HOST + '/some/endpoint

And request ''

When method POST

\* def tranid = response.tranid

\* def payload1 =

""" {"authId":"${authId}","callbacks":[{"type":"NameCallback","output":[{"accid":"tid829","amt":"4929" ]}

"""

\* replace payload1

| token | value |

| ${tranid} | tarnid |

| ${accid} | accid |

\* json mypayload1 = payload1

Given header X-Username = 'anonymous'

Given header X-Password = 'anonymous'

Given url EP\_HOST + '/openam/some-other-endpoint

And request mypayload1

When method POST

In the above example, the first call is made and the tranid is parsed from the response and saved in a variable called tranId. We then replace the second payload with the tranid retrieved in the first call. We then use the new payload to send to the next API call.

### How to read request templates and call other feature files

We can make our scenarios reusable and call them from other feature files. In this example, we can create a “generic” post\_transaction.feature file where we can send the transaction request but with a different request body

Feature: **Post transaction in db**

Scenario: **Post transaction with given accid**

Given header X-Requested-With = 'Swagger-UI'

Given header X-EP-Username = EP\_USERNAME

Given header X-EP-Password = EP\_PASSWORD

Given url EP\_HOST + '/some/endpoint/'

And request \_\_arg

When method POST

Then status 201

Note, in the above example, we are using ‘\_\_arg’ as the post body request.

We can then call the above feature file and pass in the required post body, which in turn we can read from a template

Feature: **Transaction post**

Scenario: **Transaction post in db**

\* json myReq = read('classpath:com/example/templates/idm/idm-post transaction.json')

\* call read('classpath:com/example/idm/idm -post\_transaction.feature') myReq

The above code reads a template which is in location `com/example/templates/idm/idm-create-user-template.json` and stores it as a JSON variable called myReq

Then we can send the JSON variable to the other feature file using the call method.

The template looks like

{

"mail" : "david@putsbox.com",

"accid" : "381003",

"tranid" : "tr2933",

"tranamt" : "3920",

"telephoneNumber" : "91234567890",

"dob" : "01/02/2010",

}