Developer Diary

Story of a techie's life

DataWeave Cheat Sheet

Posted on May 26, 2021June 10, 2021 by Varun Verma

Here is my most frequently used DataWeave cheat sheet that I keep handy.

Access Mule variable without DW expression

On occassions when you do not have the ability to add a DW expression and only have a TextBox in Mule, you can access the variables within by using the following #[varName] notation. e.g.

```
#[vars.sftp_password]
#[payload]
```

DateTime manipulations

The following code block describes how we can effectively accomplish the following:

1. Convert string to DateTime
2. Format current DateTime into the desired string format
3. Combine separate Date and Time components represented as strings, and convert them to a DateTime format
4. Use a string variable and convert it to Period to compute a relative DateTime

```
%dw 2.0
output application/json
var dateTime = "2021-05-26 12:03:00 AM UTC" as String
var downtime_daily_start_at = "1970-01-01T12:00:00"

// Format current DateTime into the desired string format
var curDate = now() as Date{format: "yyyy-MM-dd"}
var startTimeStr = (downtime_daily_start_at splitBy("T"))[1] ++ "
UTC"
var startDateTimeStr = (curDate ++ " " ++ startTimeStr) as String
var duration = "PT3H"

// Convert string to DateTime
var startDateTime = startDateTimeStr as DateTime {format: "yyyy-MM-dd
HH:mm:ss z"}
// Use string variable and convert it to Period to compute a relative
DateTime
var endDateTime = (startDateTime + duration) as DateTime {format:
"yyyy-MM-dd HH:mm:ss z"}
---
{
    "strToDateTime": (dateTime as DateTime {format: "yyyy-MM-dd
hh:mm:ss a z"}),
    "curDate": curDate,
    "startDateTime": startDateTime,
    "endDateTime": endDateTime,
    "downtime": (startDateTime < now()) and (now() < endDateTime)
}
```

Converting Unix timestamp/epoch to Datetime in dw

```
%dw 2.0
output application/java
var epoch = "1609459200000"
---
(epoch/1000) as DateTime  as String {format:'yyyy-MM-dd'}
```

Convert a JSON object to String

```
%dw 2.0
output application/java
---
write(payload, "application/json")
```

Convert a XML to Base64 format

Converting an XML to Base64 is a 2 step process:

1. Convert the XML to String
2. Convert the String to Base64

```
%dw 2.0
import * from dw::core::Binaries

var xmlString = write(payload, "application/xml") //convert xml
object to string
---
toBase64(xmlString)
```

Merge the response from scatter-gather payloads

When using a scatter-gather component in Mulesoft, the payloads from corresponding sub-reference-flows are aggregated together under a common payload. So if you had 2 sub-reference-flows in a scatter gather, the combined payload response is going to look similar to this:

```
{
  "0": {
    "inboundAttachmentNames": [

    ],
    "exceptionPayload": null,
    "inboundPropertyNames": [

    ],
    "outboundAttachmentNames": [

    ],
    "outboundPropertyNames": [

    ],
    "payload": [
      {
        "action_type": "EXPORTED",
        "created_at": "2020-11-05T23:55:48",
        "event_id": "31ce5def-2c48-4095-9a49-3fe59177b2b3",
        "event_type": "eventUpdated",
        "id": 1,
        "status": "RETRY"
      }
    ],
    "attributes": null
  },
  "1": {
    "inboundAttachmentNames": [

    ],
    "exceptionPayload": null,
    "inboundPropertyNames": [

    ],
    "outboundAttachmentNames": [

    ],
    "outboundPropertyNames": [

    ],
    "payload": [
      {
        "action_type": "SUBMITTED",
        "created_at": "2021-05-26T08:54:24",
        "event_id": "89f5671e-5db4-40b4-bb90-06ff2ba71479",
        "event_type": "eventUpdated",
        "id": 4,
        "status": "SUSPENDED"
      },
      {
```

```
            "action_type": "SUBMITTED",
            "created_at": "2021-05-26T08:55:53",
            "event_id": "89f5671e-5db4-40b4-bb90-06ff2ba71481",
            "event_type": "eventUpdated",
            "id": 9,
            "status": "SUSPENDED"
          }
        ],
        "attributes": null
    }
}
```

Most often, the useful piece of information with that object is the payload itself. So to combine the payload, you could use the following dataweave expression:

```
%dw 2.0
output application/json
---
payload[0].payload + payload[1].payload
```

However, if you anticipate the sub-reference-flows to grow gather over time and don't feel like updating the expression to combine them, you could use the mapObject function to gather all the payload response and flatten out the list to get the desired list of payloads.

```
%dw 2.0
output application/json
var mapped = payload mapObject(v,k,i) -> (
    v
)
---
flatten(mapped.*payload)
```

Either of them resulting in the following output:

```
[
  {
    "action_type": "EXPORTED",
    "created_at": "2020-11-05T23:55:48",
    "event_id": "31ce5def-2c48-4095-9a49-3fe59177b2b3",
    "event_type": "eventUpdated",
    "id": 1,
    "status": "RETRY"
  },
  {
    "action_type": "SUBMITTED",
    "created_at": "2021-05-26T08:54:24",
    "event_id": "89f5671e-5db4-40b4-bb90-06ff2ba71479",
    "event_type": "eventUpdated",
    "id": 4,
    "status": "SUSPENDED"
  },
  {
    "action_type": "SUBMITTED",
    "created_at": "2021-05-26T08:55:53",
    "event_id": "89f5671e-5db4-40b4-bb90-06ff2ba71481",
    "event_type": "eventUpdated",
    "id": 9,
    "status": "SUSPENDED"
  }
]
```

Array list intersection

Given two list/arrays, find the intersection, i.e. elements that are common in both arrays. e.g.

listA = [1,2,3,5]

listB = [2,4,5]

The expected output should be [2,5]

```
%dw 2.0
output application/json
var listA = [1,2,3,5]
var listB = [2,4,5]
---
listA filter((item, index) -> listB contains item)
```

Conditionally add Sibling node to XML

```
%dw 2.0
output application/xml
var payload = payload
var context = true
var soapEnv = {uri: "http://schemas.xmlsoap.org/soap/envelope/",
prefix: "SOAP-ENV"} as Namespace
ns xsd http://www.w3.org/2001/XMLSchema
ns xsi http://www.w3.org/2001/XMLSchema-instance
---
soapEnv#Envelope @('xmlns:xsd': xsd, 'xmlns:xsi': xsi):
{
    soapEnv#Header: vars.jXSoapHeader,
    soapEnv#Body: {
        CustSrch: {
            SrchMsgRqHdr: vars.jXchangeHdr ++ ("MaxRec":"10")
if(context==true),
            UserId : payload.userId,
        },
    },
}
```

Conditionally adding items to JSON

Given a payload which has an optional element metadata, which further down may have some optional items. Build an output JSON with some selected elements from the input payload. e.g. In the given payload, filter out only the primaryUser and secondaryUser elements.

```
{
   "metadata": {
        "primaryUser": {
            "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
            "fullName": "Tommy Jones"
        },
        "secondaryUser": {
            "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
            "fullName": "Bob Smith"
        },
        "primaryPhone": {
            "value": "514-019-9276",
            "type": "CELL"
        }
   }
}
```

Dataweave expression:

```
%dw 2.0
output application/json
---
{
    users: {}
    ++  (if(payload.metadata != null and payload.metadata.primaryUser
!= null) ({"primaryAssignedReviewer": payload.metadata.primaryUser})
else {})
    ++ (if(payload.metadata != null and
payload.metadata.secondaryUser != null) ({"secondaryUser":
payload.metadata.secondaryUser}) else {})
}
```

Resulting output:

```
{
  "users": {
    "primaryAssignedReviewer": {
      "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
      "fullName": "Tommy Jones"
    },
    "secondaryUser": {
      "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
      "fullName": "Bob Smith"
    }
  }
}
```

Extract selective elements from a List of Objects

Given the following input, extract all the document Id’s as a list

```
{
        "documents": [{
                        "id": "1001",
                        "name": "Account Statement"
                },
                {
                        "id": "1002",
                        "name": "Tax Statement"
                },
                {
                        "id": "1002",
                        "name": "Rental Agreement"
                }
        ]
}
```

Dataweave:

```
%dw 2.0
output application/json
---
payload.documents.id
```

Output:

```
[
  "1001",
  "1002",
  "1002"
]
```

Extract selective elements from an Object

For a given object, extract a given set of values. e.g. retrieve all the userId’s from the object.

```
{
  "users": {
    "primaryUser": {
      "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
      "fullName": "Tommy Jones"
    },
    "secondaryUser": {
      "_id": "238d15d9-5843-4c39-a14d-cb0249a0ae53",
      "fullName": "Bob Smith"
    }
  }
}
```

Dataweave:

```
%dw 2.0
output application/json
---
payload.users..*"_id"
```

Output:

```
[
  "238d15d9-5843-4c39-a14d-cb0249a0ae53",
  "238d15d9-5843-4c39-a14d-cb0249a0ae53"
]
```

The DW expression can be adjusted to get a **unique set of values**, e.g.

```
%dw 2.0
output application/json
---
payload.users..*"_id" distinctBy (value) -> { "unique" : value }
```

would output

```
[
  "238d15d9-5843-4c39-a14d-cb0249a0ae53"
]
```

Get a list of keys from object/dictionary to check on a value

To check if a value belongs to a key within an object/dictionary/json – use the *keySet()* method to get a list of keys and *contains* to check containment.

Input json/object

```
{
    "specialtyAccounts": {
        "01NA": "BUSINESS",
        "0178NA": "BUSINESS",
        "0188NA": "BUSINESS",
        "0170NA": "TRUST",
        "0175NA": "TRUST"
        }
}
```

Dataweave:

```
%dw 2.0
import * from dw::core::Objects
output application/json
var accountId = "0188NA"
---
contains(namesOf(vars.specialtyAccounts), accountId)
```

Posted in MuleSoft Tagged CheatSheet, DataWeave, MuleSoft