What is DataWeave?

DataWeave is the MuleSoft expression language for accessing and transforming data that travels through a Mule app. DataWeave is tightly integrated with the Mule runtime engine, which runs the scripts and expressions in your Mule app.

**1. Difference between DW 1.0 and DW 2.0?**

**A.)** Few of the changes from Dataweave 1.0 of Mule 3 to Dataweave 2.0 of Mule 4.

* Header changes (%dw 1.0 to %dw 2.0)
* Body changes
* Conditional logics (earlier it was **when-otherwise** and now its **if-else**)
* Function and Variable syntax
* Comments and many more

**2. What is p() in DataWeave and write a syntax.**

**A.) p()** function is used to read the data from properties files such as http host, API Keys, database passwords etc.,

%dw 2.0

output application/json

---

{

"Application Name": p('app.name')

}

**3. How to invoke or call flows from DataWeave?**

A.) We can use **lookup()** to invoke or call the flows from DataWeave. For example, lookup("anotherFlow", payload) executes a flow named anotherFlow and pass the payload to anotherFlow.

%dw 2.0

output application/json

---

{

Mule: lookup('flow2', {test:'hello'}

}

Note: You can call only flows but not sub-flows.

**4. What is the difference between map, mapObject and pluck?**

* **map** iterates over an array and returns an array
* **mapObject** iterates over an object and returns an object
* **pluck** iterates over an object and returns an array of keys, values, or indices in that object.

**5. How to access secure properties inside DataWeave?**

We need to use **secure::** prefix to access secure properties. for example,

%dw 2.0

output application/json

---

{

"ClientSecret" : p('secure::api.clientSecret')

}

**6. How to merge two arrays into single array?**

We can use **flatten** keyword to merge two arrays into single array. for example,

%dw 2.0

output application/json

---

flatten(payload..\*payload)

%dw 2.0

output application/json

var myname = ['Max']

var yourname = ['Mule']

var ourname = [myname,yourname]

---

flatten(ourname)

**Output:**

[

"Max", "Mule"

]

**7. Difference between flatten and flatMap?**

* **flatMap**iterates over each item in an array and flattens the results, can act on values and indices of items in the array.
* **flatten** turns a nested array into a simple array. only acts on the values of the arrays

%dw 2.0

output application/json

---

[ [3,5], [0.9,5.5] ] flatMap (value, index) -> value

**Output:**

[ 3, 5, 0.9, 5.5]

%dw 2.0

output application/json

var myname = ['Max']

var yourname = ['Mule']

var ourname = [myname,yourname]

---

flatten(ourname)

**Output:**

["Max", "Mule"]

**8. How to convert string to number in DataWeave?**

We can use **as Number** to convert string to number in DataWeave. for example,

%dw 2.0

output application/json

---

"Price" : "345.60" as Number

**Output:**

{ "Price" : 345.60 }

**9. What is the use of $ and $$ in DataWeave?**

* **$$** symbol is used to access the **key (or index)** of the key-value pair. In an array it returns **index** and in object it returns**key**.
* **$** symbol is used to access the **value** of the key-value pair.

%dw 2.0

output application/json

var myname =

{ "name": "Max" }

---

myname mapObject {

key: $$,

value: $

}

**Output:**

{

"key": "name",

"value": "Max"

}

**10. How to map Objects elements as an array?**

We can use **map** functions to map Objects elements as an array. for example,

%dw 2.0

output application/json

var BookShop = {

"inventory": {

"book" : {

"title": "Everyday Italian",

"author": "Giada De Laurentiis",

"year": "2005",

"price": "30.00"

},

"book" :{

"title": "Harry Potter",

"author": "J K. Rowling",

"year": "2005",

"price": "29.99"

}

}}

---

items: BookShop.inventory.\*book map (item, index) -> {

"type": "book",

"Id": index,

"title": item.title,

"author": item.author,

"year": item.year,

"price": item.price as Number

}

**Output:**

{

"items": [

{

"type": "book",

"Id": 0,

"title": "Everyday Italian",

"author": "Giada De Laurentiis",

"year": "2005",

"price": 30.00

},

{

"type": "book",

"Id": 1,

"title": "Harry Potter",

"author": "J K. Rowling",

"year": "2005",

"price": 29.99

}

]

}

**11. What is the DataWeave expression to log the current time?**

We can use **now()** function to log the current time.

%dw 2.0

output application/json

---

{

"transactionTime" : now()

}

**12. How to skip null values in DataWeave?**

We can use **skipNullOn**to skip the null values. It will skip the NULL value fields.

%dw 2.0

output application/json skipNullOn="everywhere"

---

{

"name": "NaGG",

"company": "MuleSoft",

"designation": null

}

{

"name": "NaGG"

,

"company": "MuleSoft"

}

**13. How to Merges elements from two arrays into an array of arrays**?

We can use **zip()** function to merge two arrays into an array of arrays. for example,

%dw 2.0

output application/json

---

{

"empDetails" : ["NaGG","Joe"] zip ["Integration Engineer", "Integration Lead"]

}

**14. How to mask sensitive information in DataWeave?**

We can use**mask**operator to mask the sensitive information. for example,

%dw 2.0

output application/json

import \* from dw::util::Values

var personal ={

"name": "NaGG",

"ssn": "ABCD1234",

"company": "MuleSoft"

}

---

(personal mask "ssn" with "\*\*\*\*") mask "password" with "\*\*\*\*"

{

"name": "NaGG",

"ssn": "\*\*\*\*",

"company": "MuleSoft"

}

**15. How to exclude unnecessary information from the payload? Say I don’t want to send age to external services.**

We can use **-** to remove specific key:value pairs. for example,

%dw 2.0

output application/json

var details = {

"name": "NaGG",

"ssn": "ABCD1234",

"company": "MuleSoft",

"age": "25 Years"

}

---

{

empDetails : details - "age"

}

{

"empDetails": {

"name": "NaGG",

"ssn": "ABCD1234",

"company": "MuleSoft"

}

}

**16. How to log the input as System log in DataWeave?**

We can use **log()**function to log the input as system log. for example,

%dw 2.0

output application/json

---

log ("WISHES", "All the Best Mate")

WISHES - "All the Best Mate"

**17. What is the difference between match and matches?**

* **match** uses regex to match a string and then separates it into capture groups. Returns the results in an array.
* **matches** checks if an expression matches the entire input string and returns Boolean vaue.

**match:**

%dw 2.0

output application/json

---

"me@mulesoft.com" match(/([a-z]\*)@([a-z]\*).com/)

**Output:**

[

"me@mulesoft.com",

"me",

"mulesoft"

]

**matches:**

%dw 2.0

output application/json

---

[ ("admin123" matches /a.\*\d+/), ("admin123" matches /^b.+/) ]

**Output:**

[ true, false ]

**17. What is the difference between read and readUrl?**

* **read ()**reads the a string or binary and returns parsed content.
* **readUrl()**Reads a URL, including a classpath-based URL, and returns parsed content.

**read:**

%dw 2.0

output application/xml

---

read('{ "Hello" : "Vanchiv" }','application/json')

<?xml version='1.0' encoding='UTF-8' ?>

<Hello>Vanchiv</Hello>

**readUrl:**

%dw 2.0

output application/json

---

readUrl("https://jsonplaceholder.typicode.com/posts/1", "application/json")

"userId": 1, "id": 1, "title": "sunt aut", "body": "quia et" }

The major difference is **read()**function reads the string or binary whereas **readUrl()** reads a URL (including classpath)

**18. What is the reduce() function in DataWeave?**

**reduce** can be used to process an :array and operate on each of its elements. It performs an aggregation operation on the elements of the array after performing a lambda operation (optional) on each of its element. for example,

**read:**

%dw 2.0

output application/json

var numbers =[1,2,3,4]

---

{

sum : numbers reduce ($$ + $),

concat: numbers reduce ($$ ++ $)

}

**Output:**

{

"sum": 10,

"concat": "1234"

}

**18. What is the sizeOf() and typeOf() in DataWeave?**

* **sizeOf()** returns no of elements in an array and no of key-value pairs in an object.
* **typeOf()** returns the data type of given input. such as array, object, string etc.,

**sizeOf():**

%dw 2.0

output application/json

var numbers =[1,2,3,4]

---

sizeOf(numbers)

**Output:**

4

**sizeOf():**

%dw 2.0

output application/json

var numbers =[1,2,3,4]

---

typeOf(numbers)

**Output:**

"Array"

**18. What is spiltBy() and write sample script.**

**Splits**a string into a string array based on a value that matches part of that string. It filters out the matching part from the returned array.

%dw 2.0

output application/json

var name = "Max-Mule"

---

name splitBy("-")

**Output:**

[ "Max",

"Mule"

]

**19. How to remove blank spaces from starting and ending of the string?**

We can use **trim()** function to remove any blank spaces from the beginning and end of a string.

%dw 2.0

output application/json

---

trim(" Welcome to Vanchiv ")

**Output:**

"Welcome to Vanchiv"

Note: It won’t remove any spaces from the middle of the string, only the beginning and end

**20. Write a script to convert a string to base64 and base64 to string?**

We can se **toBase64()** and **fromBase64()** functions to encode and decode a string. for example,

%dw 2.0

output application/json

import dw::Crypto

import toBase64 from dw::core::Binaries

import fromBase64 from dw::core::Binaries

var encode = "Secret"

var decode = "U2VjcmV0"

---

{

encyption: toBase64(encode),

decryption: fromBase64(decode)

}

**Output:**

{

"encyption": "U2VjcmV0",

"decryption": "Secret"

}

**21. Write a program to check wither the given number is Even or Odd?**

%dw 2.0

output application/json

var number = attributes.queryParams.number

var result = number mod 2

---

if (result == 0)

{ number: "Even" }

else { number: "Odd" }