Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

LAB- Using Arrays, Objects, Strings modules

In this lab you will understand how to use Arrays, Objects and Strings modules in dwl

Create a new project with any name

Create a mule configuration file with name “usingarraysmodule.xml”

Copy products.xml given to you into src/main/resources/examples folder and open it. observe that it contains details of various products

Configure this input metadata using this products.xml

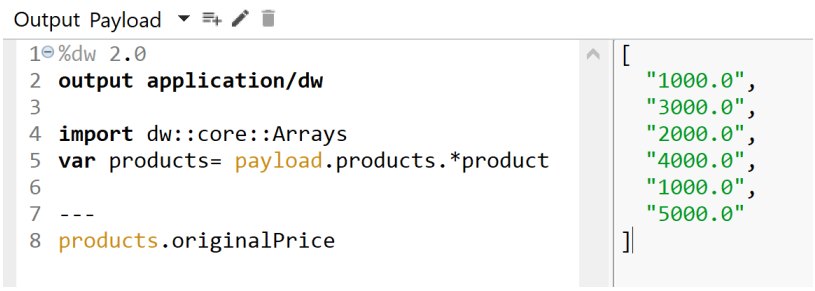
Configure the output as application/dw

Create a variable as shown below:

```
var products= payload.products.*product
```

Change the body expression to products. preview should show an array of products

Now change the the body expression as products.originalPrice to display an array of original prices of all the products



Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

Now we want to check if there is a product who price is less than 1000

Change the body expression to `products.originalPrice Arrays::some($<1000)`
Observe the result is false

Instead of using $, you can replace the expressions as below:

```
products.originalPrice Arrays::some ((price)->price>1000)
```

Actually some is a function which takes 2 arguments.

Now change the body expression to `Arrays::some(products.originalPrice , (price) -> price <1000)` and observe that result is same

Change the expression to `products.originalPrice Arrays::every ((price)->price>100)`

Every function returns true only if all the elements in the arrays satisfy the condition

We want to get the count of products whose original price is greater than 3000

Change the body expression to below:

```
products.originalPrice Arrays::countBy ((price)->price>3000)
```

To get the size of an array, we can use Arrays:sizeOf() function.

Change the body expression as shown below:

```
{
	productscountlessthan3000: products.originalPrice Arrays::countBy( $>3000 ),
	totalProducts:  sizeOf(products.originalPrice )

}
```

Observe the preview and make sure that it is as expected

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

We want the sum of all product prices. We can use Arrays::sumBy

Change the body expression as shown below:

```
{
	productscountlessthan3000: products.originalPrice Arrays::countBy( $>3000 ),
	totalProducts:  sizeOf(products.originalPrice ),
	totalCost: products.originalPrice Arrays::sumBy $

}
```

We want to get the total price of all the products whose price is greater than 3000

Change the body expression as shown below:

```
{
	productscountlessthan3000: products.originalPrice Arrays::countBy( $>3000 ),
	totalProducts:  sizeOf(products.originalPrice ),
	totalCost: products.originalPrice Arrays::sumBy if($>3000) $ else 0

}
```

We want to divide the array into equal sized arrays of size 2

Change the body expression as shown below:

```
products.originalPrice Arrays::divideBy(2)
```

Preview should look like below:

```
[
  [
    "1000.0",
    "3000.0"
  ],
  [
    "2000.0",
    "4000.0"
  ],
  [
    "1000.0",
    "5000.0"
  ]
]
```

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

Now use flatten the array as shown below:

```
%dw 2.0
output application/dw

import dw::core::Arrays

var productsXml= readUrl("classpath://examples/products.xml","application/xml")

var products=productsXml.products.*product

---
flatten(products.originalPrice Arrays::divideBy(2))
```

```
[
  "1000.0",
  "3000.0",
  "2000.0",
  "4000.0",
  "1000.0",
  "5000.0"
]
```

Now let us understand how to use functions present in dw::core:Objects module

Drag another “Transform Message” component into a new flow and rename it as “usingobjectsmoduleFlow”

Open src/main/resources/examples/productswithns.xml and observe it. I have made productId and name as attributes .Observe that tag names as product1,product2,etc.

Now create 2 variables as below:

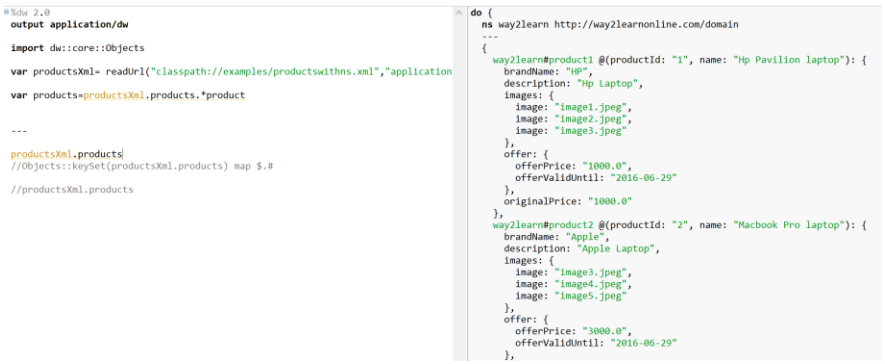
```
var productsXml= readUrl("classpath://examples/productswithns.xml","application/xml")

var products=productsXml.products.*product
```

Make the output as application/dw.

In the body expression, write productsXml and observe the preview.

Change body to productsXml.products. it should look like below:

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

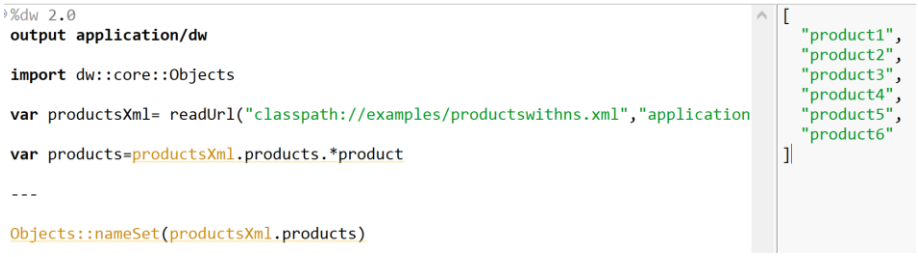


Now we want to view only keyNames .

Import dw::core::Objects

Change the expression to `Objects::nameSet(productsXml.products)`

You should see the preview as shown below:



Now change the body expression as `Objects::keySet(productsXml.products)`

output will be same as before. But what is the difference between nameSet and keyset?

keyset contains all the components of a key – the namespace, the key and attributes

Change the body expression to iterate over each key and extract namesame as shown below:

```
Objects::keySet(productsXml.products) map $.#
```

Preview should look like below:

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

```
%dw 2.0
output application/dw

import dw::core::Objects

var productsXml= readUrl("classpath://examples/productswithns.xml","application

var products=productsXml.products.*product

---

Objects::keySet(productsXml.products) map $.#
```

```
[
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace,
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace,
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace,
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace,
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace,
  {
    uri: "http://way2learnonline.com/domain",
    prefix: "way2learn"
  }as Namespace
]
```

Now change the body expression to below:

```
Objects::keySet(productsXml.products) map (product,productIndex) ->
      "prod$productIndex":product.@
```

Now preview should look like below:

```
%dw 2.0
output application/dw

import dw::core::Objects

var productsXml= readUrl("classpath://examples/productswithns.xml","application

var products=productsXml.products.*product

---

Objects::keySet(productsXml.products) map (product,productIndex) ->
    "prod$productIndex":product.@
```

```
[
  {
    prod0: {
      productId: "1",
      name: "Hp Pavilion laptop"
    }
  },
  {
    prod1: {
      productId: "2",
      name: "Macbook Pro laptop"
    }
  },
  {
    prod2: {
      productId: "3",
      name: "Mac Book  laptop"
    }
  },
```

We can achieve same result using pluck also.

Change the body value to

```
pluck(productsXml.products,(V,K,I)-> K.@)
```

preview should look like below:

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

Now change the body expression to

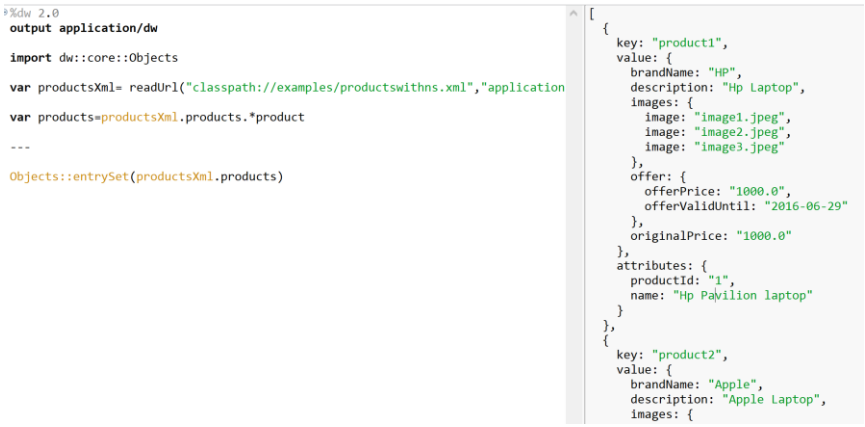
```
pluck(productsXml.products,(V,K,I)->
              ("product" ++ I): K.@
)
```

Result should look like below:

Now change the body expression to display entrySet as shown below:

```
Objects::entrySet(productsXml.products)
```

Preview should look like below:



Now change the body expression to display valueSet as shown below:

Sivaprasad.valluru@gmail.com
Whatsapp: +91 8088910831

```
%dw 2.0
output application/dw

import dw::core::Objects

var productsXml= readUrl("classpath://examples/productswithns.xml","application

var products=productsXml.products.*product

---

Objects::valueSet(productsXml.products)
```

```
[
  {
    brandName: "HP",
    description: "Hp Laptop",
    images: {
      image: "image1.jpeg",
      image: "image2.jpeg",
      image: "image3.jpeg"
    },
    offer: {
      offerPrice: "1000.0",
      offerValidUntil: "2016-06-29"
    },
    originalPrice: "1000.0"
  },
  {
    brandName: "Apple",
    description: "Apple Laptop",
    images: {
      image: "image3.jpeg",
      image: "image4.jpeg",
      image: "image5.jpeg"
    },
```

Now let us see how to merge 2 objects

```
{one:"ONE", one: "One2", two:"TWO"} Objects::mergeWith {one:"Number1"}
```

Output should look like below:

```
{
  two: "TWO",
  one: "Number1"
}
```

This is the end of the Exercise