|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **TRANSFORM** | **OUTPUT** |
| **Map**  Returns an array that is the result of applying a transformation function (lambda) to each of the elements. The lambda is invoked with two parameters: index and the value. If these parameters are not named, the index is defined by default as $$ and the value as $. | %dw 1.0  %output application/json  ---  users: ["john", "peter", "matt"] map upper $ | {  "users": [  "JOHN",  "PETER",  "MATT"  ]  } |
| **Map**  In the following example, custom names are defined for the index and value parameters of the map operation, and then both are used to construct the returned value. In this case, value is defined as firstName and its index in the array is defined as position. | %dw 1.0  %output application/json  ---  users: ["john", "peter", "matt"] map ((firstName, position) -> position ++ ":" ++ upper firstName)  // Above is a good example of lambda function | {  "users": [  "0:JOHN",  "1:PETER",  "2:MATT"  ]  } |
| **Map**  Returns an array with the values that result out of applying a transformation function (lambda) to each of the values in the object. The keys of the original object are all ignored by this operation and the object is treated as an array. To have access to the keys, you can use the operation mapObject instead. The lambda is invoked with two parameters: index and the value. If these parameters are not named, the index is defined by default as $$ and the value as $. The index refers to the position of a key:value pair when the object is treated as an array. | %dw 1.0  %output application/json  %var conversionRate=13.45  ---  priceList: payload.prices map (  '$$':{  dollars: $,  localCurrency: $ \* conversionRate  }  )  -------------------------------------------------  priceList: payload.prices map ((money, position) ->  '$position':{  dollars: money,  localCurrency: money \* conversionRate  }  ) | {  "priceList": [  {  "0": {  "dollars": "9.99",  "localCurrency": 134.3655  }  },  {  "1": {  "dollars": "53",  "localCurrency": 712.85  }  },  {  "2": {  "dollars": "398.99",  "localCurrency": 5366.4155  }  }  ]  } |
| **Map Object**  Similar to Map, but instead of processing only the values of an object, it processes both keys and values as a tuple. Also instead of returning an array with the results of processing these values through the lambda, it returns an object, which consists of a list of the key:value pairs that result from processing both key and value of the object through the lambda.  The lambda is invoked with two parameters: key and the value. If these parameters are not named, the key is defined by default as $$ and the value as $. | %dw 1.0  %output application/json  %var conversionRate=13.45  ---  priceList: payload.prices mapObject (  '$$':{  dollars: $,  localCurrency: $ \* conversionRate  }  )  priceList: payload.prices mapObject ((money, category) ->  '$category':{  dollars: money,  localCurrency: money \* conversionRate  }  ) | {  "priceList": {  "basic": {  "dollars": "9.99",  "localCurrency": 134.3655  },  "premium": {  "dollars": "53",  "localCurrency": 712.85  },  "vip": {  "dollars": "398.99",  "localCurrency": 5366.4155  }  }  } |
| **Pluck**  Pluck is useful for mapping an object into an array. Pluck is an alternate mapping mechanism to mapObject. Like mapObject, pluck executes a lambda over every key:value pair in its processed object as a tuple, but instead of returning an object, it returns an array, which may be built from either the values or the keys in the object.  The lambda is invoked with two parameters: key and the value. If these parameters are not named, the key is defined by default as $$ and the value as $. | %dw 1.0  %output application/json  ---  result: {  keys: payload.prices pluck $$,  values: payload.prices pluck $  } | {  "result": {  "keys": [  "basic",  "premium",  "vip"  ],  "values": [  "9.99",  "53",  "398.99"  ]  }  } |
| **Filter**  Returns an array that only contains those that pass the criteria specified in the lambda. The lambda is invoked with two parameters: **index** and the **value**. If these parameters are not named, the index is defined by default as **$$** and the value as **$**. | %dw 1.0  %output application/json  ---  {  biggerThanTwo: [0, 1, 2, 3, 4, 5] filter $ > 2  }  %dw 1.0  %output application/xml  ---  filtered: {  aa: "a", bb: "b", cc: "c", dd: "d"  } filter $ == "d"  %dw 1.0  %output application/json  ---  filtered: (payload filter ($.\*data contains "b") map {  data: $.data  }) | "Filtergt2": [  3,  4,  5  ],  "FilterObject": [  "d"  ]  } |
| **Remove**  Returns another array where the specified indexes are removed. | %dw 1.0  %output application/json  ---  {  aa: ["a", "b", "c"] - 1  }  myObject: {aa: "a", bb: "b"} - "aa"  myObject: {aa: "a", aa:"c", bb: "b"} -- { aa:"a"} | {  "aa": [a, c]  }  {  "myObject": {  "bb": "b"  }  }  {  "myObject": {  "aa": "c",  "bb": "b"  }  } |
| **Booloeans**  **AND,OR** | %dw 1.0  %output application/json  ---  {  currency: "USD"  } when payload.country == "USA" and payload.currency == "local"  otherwise  {  currency: "EUR"  }  {  currency: "EUR"  } when payload.country == "Italy" or payload.country == "Germany" or payload.country == "Spain" or payload.country == "Portugal" or payload.country == "France" or payload.country == "Greece"  otherwise  {  currency: "USD"  } |  |
| **Concatenation** | %dw 1.0  %output application/json  ---  {  a: [0, 1, 2] ++ [3, 4, 5]  }  {  name: "Mule" ++ "Soft"  } | {  "a": [0, 1, 2, 3, 4, 5]  }  {  "name": MuleSoft  } |
| **Contains**  Evaluates if an array or list contains in at least one of its indexes a value that validates to true and returns a boolean value. You can search for a literal value, or match a regex too. |  |  |
| **Flatten**  If you have an array of arrays, this function can flatten it into a single simple array. | [  [3,5],  [9,5],  [154,0.3]  ] | [ 3, 5, 9, 5, 154, 0.3] |
| **Size Of**  Returns the number of elements in an array (or anything that can be converted to an array such as a string). | %dw 1.0  %output application/json  ---  {  arraySize: sizeOf [1,2,3],  textSize: sizeOf "MuleSoft",  objectSize: sizeOf {a:1,b:2}  } | {  "arraySize": 3,  "textSize": 8,  "objectSize": 2  } |
| **Array Push/Remove**  Pushes a new element to the end of an array.Removes an element from an array when it matches the specified value. If multiple elements in the array match the value, they will all be removed. | aa: [0, 1, 2] + 5  {  a: [0, 1, 1, 2] - 1,  b: [{a: "a"}] - {a: "a"}  } | {  "aa": [0, 1, 2, 5]  }  {  "a": [0,2],  "b": []  } |