DataWeave Delight

Experience the DataWeave programming language through code snippets

NOTE
BOOK

Hubert A. Klein Ikkink

DataWeave Delight Notebook

Experience the DataWeave programming language through code snippets

Hubert Klein Ikkink

This book is for sale at http://leanpub.com/dataweave-delight-notebook

This version was published on 2023-05-21

Contents

Strings

Define Multi Line Strings

To define a string value in DataWeave we can use both double quotes or single quotes. To define a multi line string we simply define a string value over multiple lines. We don't have to do any strange concatenation, but simply define the value over multiple lines. DataWeave will add an end-of-line character automatically after each line.

In the following example we use different ways to defined single and multi line strings:

Source

```
%dw 2.0

import lines from dw::core::Strings

output application/json
---
{
    doubleQuotes: "String value defined using double quotes",
    singleQuotes: 'String value defined using single quotes',

    // Multi line strings can be defined as well.
    multiLineDoubleQuotes: "This is a multi line
string value
with double quotes",
    multiLineSingleQuotes: 'This is a multi line
string value with
single quotes',

    // We can use the lines function to transform
    // each line into an element in an array.
    multiLines: lines("Multiline
string transformed
to array of strings")
}
```

Output

```
{
  "doubleQuotes": "String value defined using double quotes",
  "singleQuotes": "String value defined using single quotes",
  "multiLineDoubleQuotes": "This is a multi line\nstring value\nwith double quotes",
  "multiLineSingleQuotes": "This is a multi line\nstring value with \nsingle quotes",
  "multiLines": [
    "Multiline",
    "string transformed",
    "to array of strings"
  ]
}
```

Written with DataWeave 2.4.

Original post written on July 12, 2022

Trimming string values

In DataWeave we can use the `trim` function from the `dw::Core` module to remove any space character before and after a string value. The characters that are removed are the space, tab and newline characters. There is an overloaded function definition that will accept a `null` value and returns a `null` value.

In the following example we trim string values that start and end with spaces, tabs and newline characters:

Source

```
%dw 2.0

output application/json
---
{
    // Spaces at the start and end ar removed.
    spaces: trim("   mrhaki  "),

    // Also tabs are removed.
    tabs: trim("\t mrhaki"),

    // And newline characters are removed
    newline: trim("\tmrhaki \r\n"),

    // trim will return null when a null value is used.
    nullValue: trim(null),
}
```

Output

```
{
  "spaces": "mrhaki",
  "tabs": "mrhaki",
  "newline": "mrhaki",
  "nullValue": null
}
```

Written with DataWeave 2.4.

Original post written on February 15, 2022

Reverse a string value

DataWeave 2.4 introduced the `reverse` function in the `Strings` module. With this function we reverse the string value that we pass in as argument. The result is the reversed string.

In the following example we reverse the value *DataWeave Delight*:

Source

```
%dw 2.0
import reverse from dw::core::Strings

output text/plain
---
reverse("DataWeave Delight")
```

Output

```
thgileD evaeWataD
```

Written with DataWeave 2.4.

Original post written on February 1, 2022

Repeating a string value

The `Strings` module in DataWeave has some very useful functions to work with string values. For example to repeat a string value we can use the `repeat` function. The first argument is the string value we want to repeat and the second argument the number of times the value must be repeated.

In the following example we use the `repeat` function to get 20 asterisks:

Source

```
%dw 2.0
import repeat from dw::core::Strings

output text/plain
---
repeat("*", 20)
```

Output

```
********************
```

The string value can also be more than one character:

Source

```
%dw 2.0
import repeat from dw::core::Strings

output text/plain
---
repeat("mrhaki ", 5)
```

Output

```
mrhaki mrhaki mrhaki mrhaki mrhaki
```

Written with DataWeave 2.4

Original post written on February 1, 2022

Convert string value to a boolean

If we need to convert a string value `"true"` or `"false"` to a boolean we can use the `toBoolean` function from the `dw::util::Coercions` module. The function will return a boolean `true` if the string value is `"true"`, mixed casing is allowed. And the function returns `false` for a mixed casing string value of `"false"`. Any other string value will throw an exception and will not return a boolean value.

In the following example we coerce some string values to a boolean and also include an example where the input value cannot be coerced:

Source

```
%dw 2.0

import try from dw::Runtime
import toBoolean from dw::util::Coercions

output application/json
---
{
    // Coerce all string values to a boolean with value true.
    trueBooleans: ["TRUE", "true", "True", "trUE"] map (s) -> toBoolean(s),

    // Coerce all string value to a boolean with value false.
    falseBooleans: ["FALSE", "false", "False", "falSE"] map toBoolean($),

    // An exception is thrown when the string value cannot be coerced.
    invalidCoercion: try(() -> toBoolean("Yes"))
}
```

Output

```
{
  "trueBooleans": [
    true,
    true,
    true,
    true
  ],
  "falseBooleans": [
    false,
    false,
    false,
    false
  ],
  "invalidCoercion": {
    "success": false,
    "error": {
      "kind": "InvalidBooleanException",
      "message": "Cannot coerce String (Yes) to Boolean",
      "location": "\n16|     invalidCoercion: try(() -> toBoolean(\"Yes\"))\n
                                                                  ^^^^^",

      "stack": [
        "toBoolean (main:16:42)",
        "main (main:16:32)"
      ]
    }
  }
}
```

Written with DataWeave 2.4.

Original post written on February 14, 2022

Padding strings

In DataWeave we can create a fixed width string value. If the string value is less than the fixed width than the space is padded with spaces or any other character we want. We can add the padding to the left of our string value or to the right. For left padding we use the function `leftPad` from the `dw::core::Strings` module. And for right padding we use the function `rightPad` in the same module. The functions takes as first argument the string value, the second argument is the total width of the resulting string and the third optional argument is the character we want to use for padding. By default the padding character is a space (`" "`), but we can use any character. We must make sure we only use 1 character, because when we use more than 1 character the result is no longer a string value with our defined maximum width.

In the following example we use both functions with several arguments:

Source

```
%dw 2.0

import rightPad, leftPad from dw::core::Strings

var text = "DataWeave"

output text/plain
---
"|$(text)|
|$(rightPad(text, 12))|
|$(rightPad(text, 12, "*"))|
|$(leftPad(text, 12))|
|$(leftPad(text, 12, "-"))|

Using multiple characters for padding
gives unexpected result:
|$(rightPad(text, 12, " * "))|
"
```

Output

```
|DataWeave|
|DataWeave   |
|DataWeave***|
|   DataWeave|
|---DataWeave|

Using multiple characters for padding
gives unexpected result:
|DataWeave *  *  * |
```

In the following example we use the `leftPad` and `rightPad` functions to generate a report as text.

```
%dw 2.0

import rightPad, leftPad from dw::core::Strings

var data = [
    { page: "page1.html", responseTime: 200, size: 1201 },
    { page: "page2.html", responseTime: 42,  size: 8853 },
    { page: "page3.html", responseTime: 98,  size: 3432 },
    { page: "page4.html", responseTime: 432, size: 9081 },
]

fun total(index) = data reduce ((row, result = 0) -> result + row[index] as Number)

var totalTime = total(1)
var totalSize = total(2)

output text/plain
---
"
*** Summary ***

$(rightPad("Total pages", 25)): $(leftPad(sizeOf(data) as String, 6))
$(rightPad("Total response time (ms)", 25)): $(leftPad(totalTime as String, 6))
$(rightPad("Total size (KB)", 25)): $(leftPad(totalSize as String, 6))

*** Details ***

$(data map ((item) -> rightPad(item.page, 14) ++
                      leftPad(item.responseTime, 5) ++
                      leftPad(item.size, 8))
       joinBy "\n")
"
```

Output

```
*** Summary ***

Total pages              :      4
Total response time (ms) :    772
Total size (KB)          :  22567

*** Details ***

page1.html      200    1201
page2.html       42    8853
page3.html       98    3432
page4.html      432    9081
```

Written with DataWeave 2.4.

Original post written on February 10, 2022

Wrapping string values

If we want to wrap a string value with another string value we can use the `wrapWith` and `wrapIfMissing` functions defined in the `dw::core::Strings` module. The first argument of these functions is the string we want to wrap and the second argument is the string value that wraps the first argument. The function `wrapIfMissing` will only apply the wrapper string if it was not already applied.

To remove a wrapped character from a wrapped string we can use the `unwrap` function. The first argument is the string value that is already wrapped and the second argument the character we want to use for unwrapping. The second argument can only be a single character, but we can repeatedly invoke the `unwrap` function to remove multiple wrap characters.

In the following example we use all three functions in different ways:

Source

```
%dw 2.0
import wrapWith, wrapIfMissing, unwrap from dw::core::Strings
output application/json
---
{
  wrapWith: {
    // Wrap with single quotes
    sample1: wrapWith("DataWeave", "'"),

    // Even if already wrapped, keep on wrapping
    sample2: wrapWith("'DataWeave'", "'"),

    // We can use more than one character to wrap
    sample3: wrapWith("DataWeave", "{data}"),

    // Using infix notation
    sample4: "mrhaki" wrapWith "__"
  },

  wrapIfMissing: {
    // Wrap value only if not already wrapped
    sample1: wrapIfMissing("DataWeave", "'"),

    // If already wrapped with given value then don't wrap
    sample2: wrapIfMissing("'DataWeave'", "'")
  },

  unwrap: {
    // Remove wrapping character(s)
    sample1: unwrap("'DataWeave'", "'"),

    // If given wrap characters are not used, return the value
    sample2: unwrap("DataWeave", "'"),

    // Unwrap with more than one character only removes one wrap character
    sample3: unwrap("##DataWeave##", "##"),
    // Unwrap with more than one character gives unexpected results
    sample4: unwrap(unwrap("##DataWeave##", "#"), "#"),

    // With infix notation
    sample5: ("##DataWeave##" unwrap "#") unwrap "#"
  }
}
```

Output

```
{
  "wrapWith": {
    "sample1": "'DataWeave'",
    "sample2": "''DataWeave''",
    "sample3": "{data}DataWeave{data}",
    "sample4": "__mrhaki__"
  },
  "wrapIfMissing": {
    "sample1": "'DataWeave'",
    "sample2": "'DataWeave'"
  },
  "unwrap": {
    "sample1": "DataWeave",
    "sample2": "DataWeave",
    "sample3": "#DataWeave#",
    "sample4": "DataWeave",
    "sample5": "DataWeave"
  }
}
```

Written with DataWeave 2.4.

Original post written on February 2, 2022

Turn String Into Kebab Casing With dasherize

The `dw::core::Strings` module has useful functions for working with string values. One of the functions is `dasherize`. The function takes a string argument and replaces spaces, underscores and camel-casing into dashes. The resulting string value with hyphens is also called kebab-casing. The `dasherize` function also turns any uppercase character to lowercase.

In the following example we have different input arguments with camel-casing, underscores and spaces:

Source

```
%dw 2.0
import dasherize from dw::core::Strings

output application/json
---
{
    // Replaces camel casing with dashes.
    camelCase: dasherize("stringInCamelCase"), // string-in-camel-case

    // Replaces underscores with dashes.
    snake_case: dasherize("string_with_underscores"), // string-with-underscores

    // Replaces spaces with dashes.
    spaces: dasherize("string with spaces"), // string-with-spaces

    // Uppercase is transformed to lowercase.
    upper: dasherize("STRING_WITH_UPPERCASE") // string-with-uppercase
}
```

Output

```
{
  "camelCase": "string-in-camel-case",
  "snake_case": "string-with-underscores",
  "spaces": "string-with-spaces",
  "upper": "string-with-uppercase"
}
```

Written with DataWeave 2.4.

Original post written on November 20, 2022

Using string interpolation

In DataWeave we can use expressions in strings that will be evaluated and inserted into the string value. This is called string interpolation. The expression must be enclosed in parentheses where the first parenthesis is prefixed with a dollar sign: `$(<expression>)`. The expression must return a string value or can be automatically coerced into a string value in order for it to work. The expression can also be a variable.

In the following example we use a variable, selector, the `upper` function and a calculation that returns a number that is coerced into a string:

Source

```
%dw 2.0

var prefix = "Hi"

var user = { alias: "mrhaki", name: "Hubert Klein Ikkink"}

var count = 41

output text/plain
---
"$(prefix),
welcome $(user.name) also known as $(upper(user.alias)).
You're visitor $(count + 1)."
```

Output

```
Hi,
welcome Hubert Klein Ikkink also known as MRHAKI.
You're visitor 42.
```

Written with DataWeave 2.4.

Original post written on February 6, 2022

Turn string value into array of characters

A string value can be seen as an array of characters and if we want to transform our string value to an array we can use the `toArray` function in the `dw::util::Coercions` module. Once we have transformed our string to an array we can use all functions that work on arrays. The nice thing about DataWeave is that some functions that work on arrays already have an overloaded version that accepts a string value. Then we don't have to explicitly use the `toArray` function, but we can simply use our original value when we invoke the function.

In the following example we use the `toArray` function and also see the `groupBy` function that already accepts a string value and treats it is as an array:

Source

```
%dw 2.0

import toArray from dw::util::Coercions
import isUpperCase from dw::core::Strings

output application/json
---
{
    // toArray will convert a string to an array of characters
    toArray: toArray("DataWave"),

    // joinBy function only accepts an array argument
    toArrayJoinBy: toArray("abcd") joinBy "-",

    // groupBy function has been overloaded to accept string,
    // but internally makes it an array of characters
    groupByString: "AbCd" groupBy ((character, index) -> isUpperCase(character))
}
```

Output

```
{
  "toArray": [
    "D",
    "a",
    "t",
    "a",
    "W",
    "e",
    "a",
    "v",
    "e"
  ]
  "toArrayJoinBy": "a-b-c-d",
  "groupByString": {
    "true": "AC",
    "false": "bd"
  }
}
```

Written with DataWeave 2.4.

Original post written on February 10, 2022

Getting The Ordinal Value For A Number

The `dw::core::Strings` has a lot of functions to deal with strings. One of the functions is `ordinalize` that takes a number as argument and returns the ordinal value as string.

The following exmaple shows several numbers transformed to their ordinal values:

Source

```
%dw 2.0

import ordinalize from dw::core::Strings

output application/json
---
{
    "1": ordinalize(1),
    "2": ordinalize(2),
    "3": ordinalize(3),
    "10": ordinalize(10),
    "location": "The restaurant is at the " ++ ordinalize(4) ++ " floor."
}
```

Output

```
{
  "1": "1st",
  "2": "2nd",
  "3": "3rd",
  "10": "10th",
  "location": "The restaurant is at the 4th floor."
}
```

Written with DataWeave 2.4.

Original post written on June 27, 2022