DataWeave 2.x Cheatsheet

> A comprehensive quick-reference for DataWeave 2.0 syntax, functions, and patterns.

Table of Contents

Basic Structure

```
%dw 2.0
input payload application/json          // optional: declare input MIME type
output application/json                 // required: declare output MIME type
var myVar = "hello"                     // variables
fun myFun(x) = x + 1                   // functions
type MyType = String                    // custom types
ns myns http://example.com             // XML namespaces
---
// body expression (the transformation)
payload
```

Data Types

Type	Literal	Example
String	`"..."`	`"hello world"`
Number	digits	`42` , `3.14` , `-7`
Boolean	`true` / `false`	`true`
Null	`null`	`null`
Array	`[...]`	`[1, 2, 3]`
Object	`{...}`	`{name: "Alice"}`
Date	`|yyyy-MM-dd|`	`|2026-02-15|`
DateTime	`|yyyy-MM-dd'T'HH:mm:ssZ|`	`|2026-02-15T14:30:00Z|`
LocalDateTime	`|yyyy-MM-dd'T'HH:mm:ss|`	`|2026-02-15T14:30:00|`
Time	`|HH:mm:ss|`	`|14:30:00|`
Period	`|P...|`	`|P1Y2M3D|`
Duration	`|PT...|`	`|PT1H30M|`
Regex	`/pattern/`	`/\d{3}-\d{4}/`
Range	`start to end`	`1 to 10`
Binary	—	Binary content (files, images)
CData	`"text" as CData`	XML CDATA section
Key	—	Object key type

Operators

Arithmetic

Operator	Description	Example
`+`	Add	`5 + 3` → `8`
`-`	Subtract	`5 - 3` → `2`
`*`	Multiply	`5 * 3` → `15`
`/`	Divide	`10 / 3` → `3.3333...`
`mod`	Modulo	`10 mod 3` → `1`

Comparison

Operator	Description	Example
`==`	Equal	`5 == 5` → `true`
`!=`	Not equal	`5 != 3` → `true`
`>`	Greater than	`5 > 3` → `true`
`<`	Less than	`3 < 5` → `true`
`>=`	Greater or equal	`5 >= 5` → `true`
`<=`	Less or equal	`3 <= 5` → `true`
`~=`	Similarity (coerces then compares)	`"1" ~= 1` → `true`

Logical

Operator	Description	Example
`and`	Logical AND	`true and false → false`
`or`	Logical OR	`true or false → true`
`not`	Logical NOT	`not true → false`

String / Array

Operator	Description	Example
`++`	Concatenate (strings, arrays, objects)	`"a" ++ "b" → "ab"`
`--`	Remove items from array	`[1,2,3] -- [2] → [1,3]`
`-`	Remove key from object	`{a:1, b:2} - "b" → {a:1}`
`<<`	Append to array	`[1,2] << 3 → [1,2,3]`
`>>`	Prepend / timezone shift	`dt >> |+05:00|`
`contains`	Check membership	`[1,2,3] contains 2 → true`

Other

Operator	Description	Example
`default`	Fallback for null	`null default "N/A" → "N/A"`
`as`	Type coercion	`"42" as Number → 42`
`is`	Type check	`42 is Number → true`
`..`	Descendants selector	`payload..name`

Selectors

Selector	Description	Example
`.field`	Single value	`payload.name`
`.*field`	Multi-value (all matches)	`payload.*item`
`..field`	Descendants (recursive)	`payload..id`
`.@attr`	XML attribute	`element.@id`
`.@`	All attributes	`element.@ → {id: "1", ...}`
`[n]`	Index access	`payload[0]`
`[-n]`	Negative index (from end)	`payload[-1]` (last element)
`[n to m]`	Range slice	`payload[0 to 2]`
`[?(...)]`	Filter selector	`payload[?($.age > 30)]`
`.#`	Namespace	`element.#`
`*:name`	Wildcard namespace	`payload.*:Element`

Core Array Functions

Function	Signature	Example	Result
`map`	`Array.map((item, idx) -> R)`	`[1,2,3] map $ * 2`	`[2,4,6]`
`filter`	`Array.filter((item, idx) -> Bool)`	`[1,2,3] filter $ > 1`	`[2,3]`
`reduce`	`Array.reduce((item, acc) -> R)`	`[1,2,3] reduce ($ + $$)`	`6`
`flatMap`	`Array.flatMap((item) -> Array)`	`[[1,2],[3]] flatMap $`	`[1,2,3]`
`flatten`	`flatten(Array<Array>)`	`flatten([[1],[2,3]])`	`[1,2,3]`
`groupBy`	`Array.groupBy((item) -> Key)`	`arr groupBy $.type`	`{type: [...]}`
`distinctBy`	`Array.distinctBy((item) -> Key)`	`arr distinctBy $.id`	deduplicated
`orderBy`	`Array.orderBy((item) -> Comparable)`	`arr orderBy $.name`	sorted asc
`zip`	`Array.zip(Array)`	`[1,2] zip ["a","b"]`	`[[1,"a"],[2,"b"]]`
`sizeOf`	`sizeOf(Array)`	`sizeOf([1,2,3])`	`3`
`isEmpty`	`isEmpty(Array)`	`isEmpty([])`	`true`
`contains`	`Array contains value`	`[1,2] contains 2`	`true`
`indexOf`	`Array indexOf value`	`["a","b"] indexOf "b"`	`1`
`every`	`Array every ((item) -> Bool)`	`[2,4] every ($ mod 2 == 0)`	`true`
`some`	`Array some ((item) -> Bool)`	`[1,3] some ($ > 2)`	`true`
`sum`	`sum(Array<Number>)`	`sum([1,2,3])`	`6`
`avg`	`avg(Array<Number>)`	`avg([1,2,3])`	`2`
`min`	`min(Array<Comparable>)`	`min([3,1,2])`	`1`
`max`	`max(Array<Comparable>)`	`max([3,1,2])`	`3`
`maxBy`	`Array maxBy (fn)`	`arr maxBy $.price`	item with max price
`minBy`	`Array minBy (fn)`	`arr minBy $.price`	item with min price
`countBy`	`Array countBy (fn)`	`[1,2,3] countBy ($ > 1)`	`{true:2, false:1}`
`take`	`Array take n`	`[1,2,3] take 2`	`[1,2]`
`drop`	`Array drop n`	`[1,2,3] drop 1`	`[2,3]`
`splitAt`	`splitAt(Array, n)`	`splitAt([1,2,3], 2)`	`{l:[1,2], r:[3]}`

Import for extended functions

```
import * from dw::core::Arrays
// Adds: divideBy, drop, dropWhile, every, indexOf, join, leftJoin,
// outerJoin, partition, slice, some, splitAt, sumBy, take, takeWhile
```

Core Object Functions

Function	Signature	Example
`mapObject`	`Obj.mapObject((v, k, idx) -> Obj)`	`obj mapObject {(upper($$)): $}`
`filterObject`	`Obj.filterObject((v, k) -> Bool)`	`obj filterObject ($ != null)`
`pluck`	`Obj.pluck((v, k, idx) -> R)`	`obj pluck {key: $$, val: $}`
`keysOf`	`keysOf(Obj)`	`keysOf({a:1}) → ["a"]`
`valuesOf`	`valuesOf(Obj)`	`valuesOf({a:1}) → [1]`
`sizeOf`	`sizeOf(Obj)`	`sizeOf({a:1, b:2}) → 2`
`++`	merge objects	`{a:1} ++ {b:2} → {a:1, b:2}`
`-`	remove key	`{a:1, b:2} - "a" → {b:2}`
`update`	modify nested fields	`obj update {case .a.b -> 42}`
`namesOf`	`namesOf(Obj)`	`namesOf({a:1}) → ["a"]`

Dynamic keys

```
{(myVar): myValue}              // key from variable
{(items map {($.key): $.value})} // key from expression
```

String Functions

Function	Example	Result
`upper("hello")`		`"HELLO"`
`lower("HELLO")`		`"hello"`
`trim(" hi ")`		`"hi"`
`sizeOf("hello")`		`5`
`contains("hello", "ell")`		`true`
`startsWith("hello", "he")`		`true`
`endsWith("hello", "lo")`		`true`
`"hello" splitBy "l"`		`["he", "", "o"]`
`["a","b"] joinBy ","`		`"a,b"`
`"hello" replace "l" with "L"`		`"heLLo"`
`"hello" match /h(e)(l+)o/`		`["hello","e","ll"]`
`"abc" matches /^[a-z]+$/`		`true`
`"abc123" scan /[a-z]+/`		`[["abc"]]`
`"hello"[0]`		`"h"`
`"hello"[1 to 3]`		`"ell"`

dw::core::Strings module

```
import * from dw::core::Strings
camelize("my_var")       // "myVar"
underscore("myVar")      // "my_var"
capitalize("hello")      // "Hello"
dasherize("myVar")       // "my-var"
pluralize("item")        // "items"
singularize("items")     // "item"
charCode("A")            // 65
charCodeAt("Hello", 0)   // 72
isAlpha("abc")           // true
isAlphanumeric("abc123") // true
isNumeric("123")         // true
isWhitespace("  ")       // true
leftPad("42", 5, "0")    // "00042"
rightPad("hi", 5, ".")   // "hi..."
repeat("ab", 3)          // "ababab"
substringAfter("a-b", "-")  // "b"
substringBefore("a-b", "-") // "a"
withMaxSize("hello world", 5) // "hello"
```

Number Functions

```
import * from dw::core::Numbers

ceil(3.2)      // 4
floor(3.8)     // 3
round(3.5)     // 4
abs(-42)       // 42
sqrt(16)       // 4.0
pow(2, 8)      // 256
mod(10, 3)     // 1
isEven(4)      // true
isOdd(3)       // true
isDecimal(3.14) // true
isInteger(42)  // true
toHex(255)     // "ff"
fromHex("ff")  // 255
toBinary(10)   // "1010"
fromBinary("1010") // 10
random()       // random number 0..1
```

Date/Time

Literals

```
|2026-02-15|                   // Date
|14:30:00|                     // Time
|2026-02-15T14:30:00Z|         // DateTime (UTC)
|2026-02-15T14:30:00-05:00|   // DateTime (offset)
|2026-02-15T14:30:00|         // LocalDateTime
|P1Y2M3D|                      // Period (1 year, 2 months, 3 days)
|PT1H30M|                     // Duration (1 hour, 30 minutes)
```

Operations

```
now()                              // current DateTime
today()                            // current Date (DW 2.4+)

// Arithmetic
|2026-02-15| + |P7D|              // 2026-02-22 (add 7 days)
|2026-02-15| - |P1M|              // 2026-01-15 (subtract 1 month)
|2026-02-15T14:30:00Z| + |PT2H|  // add 2 hours

// Difference
(|2026-03-01| - |2026-02-15|).days  // 14

// Timezone shift
|2026-02-15T14:30:00Z| >> |-05:00|  // shift to EST

// Epoch conversion
now() as Number {unit: "seconds"}      // to epoch seconds
now() as Number {unit: "milliseconds"} // to epoch millis
1708000000 as DateTime {unit: "seconds"} // from epoch

// Formatting
now() as String {format: "yyyy-MM-dd"}
now() as String {format: "MM/dd/yyyy HH:mm:ss"}
now() as String {format: "EEEE, MMMM dd, yyyy"}  // "Saturday, February 15, 2026"

// Parsing
"02/15/2026" as Date {format: "MM/dd/yyyy"}
"20260215" as Date {format: "yyyyMMdd"}
```

Common Format Tokens

Token	Meaning	Example
`yyyy`	4-digit year	2026
`yy`	2-digit year	26
`MM`	Month (01-12)	02
`MMM`	Short month	Feb
`MMMM`	Full month	February
`dd`	Day (01-31)	15
`EEEE`	Full day name	Saturday
`EEE`	Short day name	Sat
`HH`	24-hour (00-23)	14
`hh`	12-hour (01-12)	02
`mm`	Minutes	30
`ss`	Seconds	45
`a`	AM/PM	PM
`XXX`	Offset	+00:00
`Z`	Offset compact	+0000

Type Coercion

```
// String ↔ Number
"42" as Number                          // 42
42 as String                            // "42"
42 as String {format: "000"}            // "042"
1299.5 as String {format: "#,##0.00"} // "1,299.50"

// String ↔ Date
"2026-02-15" as Date                                // Date
"02/15/2026" as Date {format: "MM/dd/yyyy"}       // Date
|2026-02-15| as String {format: "yyyyMMdd"}       // "20260215"

// String ↔ Boolean
"true" as Boolean                       // true
true as String                          // "true"

// Any ↔ CData (XML)
"<html>content</html>" as CData      // wraps in CDATA

// Coercion with default (null-safe)
payload.value as Number default 0
payload.date as Date default |2000-01-01|
```

Flow Control

```
// if/else
if (condition) valueA else valueB

// Ternary (same as if/else)
if (x > 0) "positive" else if (x < 0) "negative" else "zero"

// do block (scoped variables)
do {
    var temp = payload.value * 2
    ---
    temp + 1
}

// unless/otherwise
"value" unless condition otherwise "other"
```

Pattern Matching

```
payload match {
    case is String -> "It's a string"
    case is Number -> "It's a number"
    case is Array -> "It's an array"
    case is Object -> "It's an object"
    case is Null -> "It's null"
    else -> "Unknown type"
}

// With variable binding
payload match {
    case str is String -> upper(str)
    case num is Number -> num * 2
    case arr is Array -> sizeOf(arr)
    else -> null
}

// Literal matching
payload.status match {
    case "active" -> "Active"
    case "inactive" -> "Inactive"
    case "pending" -> "Pending"
    else -> "Unknown"
}

// Regex matching
payload.email match {
    case email matches /(.+)@(.+)/ -> {user: email[1], domain: email[2]}
    else -> {error: "Invalid email"}
}
```

Functions & Lambdas

```
// Named function
fun greet(name: String): String = "Hello, $(name)!"

// With default parameter
fun greet(name: String = "World"): String = "Hello, $(name)!"

// Lambda (anonymous function)
var double = (n: Number) -> n * 2

// Higher-order function
fun applyTwice(fn: (Number) -> Number, x: Number): Number = fn(fn(x))

// Function overloading
fun format(d: Date): String = d as String {format: "yyyy-MM-dd"}
fun format(n: Number): String = n as String {format: "#,##0.00"}

// Annotation
@TailRec()
fun factorial(n: Number, acc: Number = 1): Number =
    if (n <= 1) acc
    else factorial(n - 1, acc * n)
```

Variables

```
// Simple variable
var name = "Alice"

// Computed variable
var total = payload.items reduce (item, acc = 0) -> acc + item.price

// Destructuring is not directly supported, but you can:
var first = payload[0]
var rest = payload[1 to -1]

// Constants in header
var TAX_RATE = 0.0875
var MAX_RETRIES = 3
```

Modules & Imports

```
// Import all functions from a module
import * from dw::core::Strings
import * from dw::core::Arrays
import * from dw::core::Objects
import * from dw::core::Numbers
import * from dw::core::Binaries
import * from dw::core::URL
import * from dw::Runtime

// Import specific functions
import camelize, underscore from dw::core::Strings
import divideBy from dw::core::Arrays
import try from dw::Runtime

// Import custom module (from src/main/resources/modules/)
import modules::MyUtils
MyUtils::myFunction(payload)

// Import with alias
import myFunction as mf from modules::MyUtils
```

Useful Modules

Module	Key Functions
`dw::core::Strings`	camelize, underscore, capitalize, leftPad, repeat
`dw::core::Arrays`	divideBy, partition, join, leftJoin, outerJoin
`dw::core::Objects`	mergeWith, namesOf, valuesOf, entrySet
`dw::core::Numbers`	ceil, floor, round, abs, pow, sqrt
`dw::core::URL`	encodeURI, decodeURI, encodeURIComponent
`dw::Runtime`	try, orElse, fail, wait, location
`dw::Crypto`	hashWith, hmacWith, MD5, SHA1

XML Specifics

```
// Namespace declaration
ns soap http://schemas.xmlsoap.org/soap/envelope/
ns ord http://example.com/orders

// Read namespaced element
payload.soap#Envelope.soap#Body.ord#Order

// Read attribute
payload.Order.@id
payload.Order.@              // all attributes

// Set attribute in output
{Order @(id: "123", status: "new"): {
    Item @(sku: "SKU-100"): "Laptop"
}}

// Repeating elements (array of same-named elements)
payload.Items.*Item          // returns array

// Self-closing element
{Contact @(email: "a@b.com"): null}
// Output: <Contact email="a@b.com"/>

// CDATA
{Description: "has <html>" as CData}

// Wildcard namespace
payload.*:Order              // any namespace prefix
```

CSV Specifics

```
// Input config
input payload application/csv separator="|", header=true, quoteChar="\""

// Output config
output application/csv separator=",", quoteValues=true, header=true

// Access with headers
payload[0].columnName

// Access without headers
// header=false → columns are column_0, column_1, etc.
payload[0].column_0

// Properties
// separator    : delimiter character (default: ",")
// header       : first row is headers (default: true)
// quoteChar    : quote character (default: "\"")
// quoteValues  : quote all output values (default: false)
// escapeChar   : escape character (default: "\\")
// streaming    : enable streaming for large files (default: false)
// bodyStartLineNumber : skip N lines before parsing
```

Error Handling

```
import try, orElse from dw::Runtime

// default (null safety)
payload.field default "fallback"
payload.items default []
payload.count default 0

// try — returns {success: Boolean, result/error: ...}
var result = try(() -> payload.value as Number)
---
if (result.success) result.result else 0

// orElse — inline fallback
try(() -> payload.value as Number) orElse 0

// Chained defaults
payload.primary default payload.secondary default "fallback"

// isEmpty (null + empty check)
isEmpty(payload.name)         // true if null, "", or []
isEmpty(payload.items)

// Conditional with null check
if (payload.field != null) payload.field else "default"

// fail — throw an error
if (payload.amount < 0) fail("Amount cannot be negative")
else payload.amount

// log — debug logging (outputs to Mule console)
log("myLabel", payload)       // logs payload, returns payload
```

Output Formats & MIME Types

Format	MIME Type	Common Options
JSON	`application/json`	`indent=true`, `skipNullOn="everywhere"`
XML	`application/xml`	`indent=true`, `writeDeclaration=true`, `encoding="UTF-8"`
CSV	`application/csv`	`separator`, `header`, `quoteValues`
Java	`application/java`	(for Mule Java objects)
Plain text	`text/plain`	
Binary	`application/octet-stream`	
URL encoded	`application/x-www-form-urlencoded`	
Multipart	`multipart/form-data`	
YAML	`application/yaml`	
Excel	`application/xlsx`	(Mule 4.4+)

Useful output options

```
output application/json indent=true, skipNullOn="everywhere"
output application/xml indent=true, writeDeclaration=true
output application/csv separator="|", quoteValues=true
output application/json deferred=true    // streaming output
```

Common Recipes

Remove nulls from object

```
payload filterObject (v, k) -> v != null
```

Remove nulls recursively

```
fun stripNulls(data) = data match {
    case obj is Object -> obj filterObject (v) -> v != null
        mapObject (v, k) -> {(k): stripNulls(v)}
    case arr is Array -> arr filter ($ != null) map stripNulls($)
    else -> data
}
```

Flatten nested array

```
payload flatMap (parent) -> parent.children map (child) -> child ++ {parentId: parent.id}
```

Convert array to lookup object

```
{(payload map (item) -> {(item.id): item})}
```

Paginate / chunk array

```
import divideBy from dw::core::Arrays
payload divideBy 100    // chunks of 100
```

UUID generation

```
import dw::Crypto
Crypto::randomBytes(16) as String {encoding: "hex"}
```

Conditional field inclusion

```
{
    name: payload.name,
    (email: payload.email) if payload.email != null,
    (phone: payload.phone) if !isEmpty(payload.phone)
}
```

Merge array of objects into one

```
payload reduce (item, acc = {}) -> acc ++ item
```

Count by field value

```
payload groupBy $.status mapObject (items, status) -> {(status): sizeOf(items)}
```

Deep get with null safety

```
payload.customer.address.street default "N/A"
```

Quick Reference Card

```
%dw 2.0                          // version declaration
output application/json           // output format
var x = 42                        // variable
fun f(n) = n + 1                  // function
type T = String                   // type alias
ns ns1 http://example.com         // namespace
---                               // separator (header / body)
payload                           // body expression

// Shorthand parameters
$    = current item (in map, filter, etc.)
$$   = current key/index
$$$  = current index (in mapObject)

// String interpolation
"Hello $(name), you have $(count) items"

// Conditional field
{(fieldName: value) if condition}

// Dynamic key
{(expression): value}
```

Back to all patterns
