Class	Utility description	functions	function description	Syntax	Input (arguments)	Output	Comments	Example		
								Input	Source code	output
Coercions		toArray	Splits a String value into an Array of characters.	toArray(@StreamCapable text: String): Array<String>	string or number or boolean	Array of (characters as string)	can be used for Date as well. example: toArray (payload.message as Date)	{ "message": "Hello" }	%dw 2.0 import * from dw::util::Coercions output application/json --- toArray(payload.message)	["H", "e", "l", "l", "o"]
		toBinary	Transform a String value into a Binary value using the specified encoding.	toBinary(str: String, encoding: String): Binary	string or number or boolean and encoding scheme	Binary value	Accepts encodings that are supported by your JDK. Use output type application/dw and then decode the value of output in online base64 decoder with same encoding scheme - you will get the input string	{ "a": "hello" }	%dw 2.0 import * from dw::util::Coercions output application/dw --- toBinary(payload.a, "UTF-16")	"/v8AaABlAGwAbABv" as Binary {base: "64"}
		toBoolean	Transform a String value into a Boolean value.	toBoolean(str: String): Boolean	string(true or false - case insensitive e.g "tRuE") or condition	Boolean	argument can be condition as well e.g toBoolean(payload == "hello") . We cannot pass any function or lambda as a input argument	{ "message": "trUE" }	%dw 2.0 import * from dw::util::Coercions output application/dw --- toBoolean(payload.message)	TRUE
		toDate	Transforms a String value into a Date value and accepts a format and locale.	toDate(str: String, format: String | Null = null, locale: String | Null = null): Date	date as string(any format), format(optional or null), locale(optional - country code as string e.g "US", "UK", "EU")	Date (yyyy-MM-dd)	whatever the format of the date string you are sending as the first argument needs to be added as a second argument. refer example for the same	{ "message": "2022/09/03" }	%dw 2.0 import * from dw::util::Coercions output application/java --- toDate(payload.message, "uuuu/MM/dd" ,"UK")	2022-09-03

		toDateTime	Transforms a String value into a DateTime value and accepts a format and locale.	toDateTime(str: String, format: String | Null = null, locale: String | Null = null): DateTime	Number(epochTime) and unit as string ("milliseconds" or "seconds")
					date as string(any format), format(optional or null), locale(optional - country code as string e.g "US", "UK", "EU")
						DateTime
						DateTime
							get the epoch time - now() as Number ....you will get the time in seconds...if you have a Epoch time in miliseconds (toNumber() can be used to get time in ms)..then you can convert it to Date Time as given in example
							format of datetime will remain same in output if output type. But the typeOf(data) will be DateTime
								{ "message": 1643459378000 }
								{ "message": "2022-1-29 23:57:59Z" }
									%dw 2.0
									import * from dw::util::Coercions
									output application/json
									--- 	
									//now() as Number
									toDateTime(payload.message, "milliseconds")
									%dw 2.4
									import * from dw::util::Coercions
									output application/json
									--- 	
									toDateTime(payload.message,"uuuu-MM-dd HH:mm:ssz")
										"2022-01-29T12:29:38Z"
										"2022-01-29 23:57:59Z"

		toLocalDateTime	Transforms a String value into a LocalDateTime value and accepts a format and locale.	toLocalDateTime(str: String, format: String | Null = null, locale: String | Null = null): LocalDateTime	dateTime as String	LocalDateTime			%dw 2.0
									import * from dw::util::Coercions
									output application/json
									--- 	
									{
									"LocalDateTime" : toLocalDateTime(now() as String),
									"LocalTime" : toLocalTime(now() as String)
									}
										{
										"LocalDateTime": "2022-01-30T15:37:27.210781",
										"LocalTime": "15:37:27.210866"
										}

		toLocalTime	Transforms a String value into a LocalTime value and accepts a format and locale.	toLocalTime(str: String, format: String | Null = null, locale: String | Null = null): LocalTime	dateTime as String	LocalTime				
		toNumber	A variant of toNumber that transforms a DateTime value into a number of seconds or milliseconds, depending on the selected unit.	toNumber(dateTime: DateTime, unit: MillisOrSecs | Null = null): Number	DateTime or a number as string	Number	you can also convert a number having string type to number type. eg: toNumber("67")	{ "message": "2022-01-29T13:32:40.545348Z" }	%dw 2.4 import * from dw::util::Coercions output application/json --- toNumber(payload.message as DateTime, "milliseconds")	1643463160545

		toPeriod	Transform a String value into a Period value.	toPeriod(str: String): Period	period as string	Period			%dw 2.0
									import * from dw::util::Coercions
									output application/dw
									--- 	
									{
									"$(typeOf(toPeriod("PT1H1M")))" : toPeriod("PT1H1M")
									}
										{
										Period: |PT1H1M|
										}

		toRegex	Transforms a String value into a Regex value.	toRegex(str: String): Regex	string	regex			%dw 2.0
									import * from dw::util::Coercions
									output application/dw
									--- 	
									{
									"$(typeOf(toRegex("a-z")))" : toRegex("a-z"),
									}
										{
										Regex: /a-z/
										}

		toString	A variant of toString that transforms a Number value (whole or decimal) into a String value and accepts a format, locale, and rounding mode value.	toString(number: Number, format: String | Null = null, locale: String | Null = null, roundMode: RoundingMode | Null = null): String toString(date: Date | DateTime | LocalDateTime | LocalTime | Time, format: String | Null = null, locale: String | Null = null): String toString(binary: Binary, encoding: String): String	number,format, roundingmode date,format(optional), locale(optional) binary,encoding	string string string	you can use 0 or # for rounding off		%dw 2.0 import * from dw::util::Coercions output application/json --- { "Number_Eg1" : toString(payload.testnumber), "Number_Eg2": toString(payload.testnumber, "0"), "Number_Eg3": toString(payload.testnumber, ".##"), "Boolean_Eg4": toString(true),	{ "Number_Eg1": "5.951", "Number_Eg2": "6",

	This utility is used for type conversion. Introduced in DataWeave version 2.4.0.			toString(data: TimeZone | Uri | Boolean | Period | Regex | Key): String	TimeZone | Uri | Boolean | Period | Regex | Key	string		{
								"testnumber": 5.951,
								"testArray": ["h","e","l","l","o"]
								}
									"Array_Eg5": toString(payload.testArray),
									"Binary_Eg6": toString("/v8AaABlAGwAbABv" as Binary {base: "64"},"UTF-16"),
									"DateTime_Eg7": toString(|2003-10-01T23:57:59|, "uuuu-MM-dd HH:mm:ss a")
									}
										"Number_Eg3": "5.95",
										"Boolean_Eg4": "true",
										"Array_Eg5": "hello",
										"Binary_Eg6": "hello",
										"DateTime_Eg7": "2003-10-01 23:57:59 PM"
										}

				toString(arr: Array<String>): String	string array	string				

		toTime	Transforms a String value into a Time value and accepts a format and locale.	toTime(str: String, format: String | Null = null, locale: String | Null = null): Time	time(string),format (optional),locale (optional)	time		{
								"stringTime" : "23:57:59Z",
								"stringTimeZone": "IST",
								"url" : "https://www.google.com/"
								}
									%dw 2.0
									import * from dw::util::Coercions
									output application/json
									--- 	
									{
									"inputUrlType": typeOf(payload.stringTime),
									"time": toTime(payload.stringTime),
									"outputTimetype": typeOf(toTime(payload.stringTime)),
									
									"timezone": toTimeZone(payload.stringTimeZone),
									
									"inputUrlType": typeOf(payload.url),
									"url": toUri(payload.url),
									"outputUrltype": typeOf(toUri(payload.url))
									
									}
										{
										"inputUrlType": "String",
										"time": "23:57:59Z",
										"outputTimetype": "Time",
										"timezone": "Asia/Kolkata",
										"inputUrlType": "String",
										"url": "https://www.google.com/",
										"outputUrltype": "Uri"
										}

		toTimeZone	Transform a String value into a TimeZone value.	toTimeZone(str: String): TimeZone	timezone(string)	timezone				
		toUri	Transforms a String value into a Uri value.	toUri(str: String): Uri	url(string)	uri				

Diff	calculates the difference between two values and returns a list of differences.	diff	Returns the structural differences between two values.	diff(actual: Any, expected: Any, diffConfig: { unordered?: Boolean } = {}, path: String = "(root)"): Diff	The actual value (Can be any data type) ,The expected value to compare to the actual (Can be any data type) and unordered value (boolean eg. {unordered: true})	Diff = { matches: Boolean, diffs: Array<Difference> }	Example with no differences:
							
							{ "matches": true, "diffs": [ ] }
							
							Example with differences:
							
							{ "matches": false, "diffs": [{"expected": "4", "actual": "2", "path": "(root).a"] }
								{
								"message": "hello"
								}
									%dw 2.4
									import * from dw::util::Diff
									output application/json
									--- 	
									{
									"sameString": diff(payload.message,"hello"),
									"diffKey": diff({a:1}, {b:1}),
									"diffSize": diff({a: 1, b:1}, {b:1}),
									"diffType": diff([1,2], {b:1}),
									"restrictOrder": diff({a: 1, b:1}, {b: 1, a:1}, {unordered: false}),
									"noOrderMatch" : diff({a: 1, b:1}, {b: 1, a:1}, {unordered: true})
									}
										{
										"sameString": {
										"matches": true,
										"diffs": [
										
										]
										},
										"diffKey": {
										"matches": false,
										"diffs": [
										{
										"expected": "Entry (root).b with type Number",
										"actual": "was not present in object.",
										"path": "(root).b"
										}
										]
										},
										"diffSize": {
										"matches": false,
										"diffs": [
										{
										"expected": "Object size is 1",
										"actual": "2",
										"path": "(root)"
										}
										]
										},
										"diffType": {
										"matches": false,
										"diffs": [
										{
										"expected": "Object type",
										"actual": "Array type",
										"path": "(root)"
										}
										]
										},
										"restrictOrder": {
										"matches": false,
										"diffs": [
										{
										"expected": "Entry (root).a with type Number",
										"actual": "was not present in object.",
										"path": "(root).a"
										},
										{
										"expected": "Entry (root).b with type Number",
										"actual": "was not present in object.",
										"path": "(root).b"
										}
										]
										},
										"noOrderMatch": {
										"matches": true,
										"diffs": [
										
										]
										}
										}

		acos	Returns an arc cosine value that can range from 0.0 through pi.	acos(angle: Number): Number | NaN	angle - Number to convert into its arc cosine value.	arc cosine value.	If the absolute value of the input is greater than 1, the result is null.			
		asin	Returns an arc sine value that can range from -pi/2 through pi/2.	asin(angle: Number): Number | NaN	angle - Number to convert into its arc sine value.	arc sine value.	If the absolute value of the input is greater than 1, the result is null.			

Math	provides mathematical functions. Introduced in DataWeave version 2.4.0.	atan	Returns an arc tangent value that can range from -pi/2 through pi/2.	atan(angle: Number): Number	angle - Number to convert into its arc tangent value.	arc tangent value.		{ "radian": 2, "degree": 90, "valueForArc" : 0.5, "logOpn": 10 }	%dw 2.4
									import * from dw::util::Math
									output application/json
									--- 	
									{
									"acos": acos(payload.valueForArc),
									"asin": asin(payload.valueForArc),
									"atan": atan(payload.valueForArc),
									"cos": cos(payload.radian),
									"log10": log10(payload.logOpn),
									"logn": logn(payload.logOpn),
									"sin": sin(payload.radian),
									"tan": tan(payload.radian),
									"toDegrees": toDegrees(payload.radian),
									"toRadians": toRadians(payload.degree)
									}
										{ "acos": 1.0471975511965979, "asin": 0.5235987755982989, "atan": 0.4636476090008061, "cos": -0.4161468365471424, "log10": 1.0, "logn": 2.302585092994046, "sin": 0.9092974268256817, "tan": -2.185039863261519, "toDegrees": 114.59155902616464175369272868838 22, "toRadians": 1.5707963267948966192 30 }

		cos	Returns the trigonometric cosine of an angle from a given number of radians.	cos(angle: Number): Number	angle - Number of radians in an angle.	cosine of angle				
		log10	Returns the logarithm base 10 of a number.	log10(a: Number): Number | NaN	Number	logarithm base 10 of a number.				
		logn	Returns the natural logarithm (base e) of a number.	logn(a: Number): Number | NaN	Number	natural logarithm (base e) of a number				
		sin	Returns the trigonometric sine of an angle from a given number of radians.	sin(angle: Number): Number	Number of radians in an angle.	sine of angle				
		tan	Returns the trigonometric tangent of an angle from a given number of radians.	tan(angle: Number): Number	Number of radians in an angle.	tangent of angle				
		toDegrees	Converts an angle measured in radians to an approximately equivalent number of degrees.	toDegrees(angrad: Number): Number	Number of radians	number of degrees.				
		toRadians	Converts a given number of degrees in an angle to an approximately equivalent number of radians.	toRadians(angdeg: Number): Number	number of degrees.	Number of radians				

Timer	contains functions for measuring time.	currentMilliseconds	Returns the current time in milliseconds.	currentMilliseconds(): Number	no argument	time in milliseconds.		{ "message" : "hello" }	%dw 2.4
									import * from dw::util::Timer
									import * from dw::Runtime
									output application/json
									
									fun timeCheck() = (payload dw::Runtime::wait 1000)
									--- 	
									{
									"test" : "hello" wait 400,
									"currentMilliseconds": currentMilliseconds(),
									"duration":duration(()->timeCheck()),
									"time": time(()->timeCheck()),
									"toMilliseconds": toMilliseconds(now()),
									}
										` "test": "hello", "currentMilliseconds": 1643469942105, "duration": { "time": 1000, "result": { "message": "hello" } }, "time": { "start": "2022-01-29T15:25:43.105264Z", "result": { "message": "hello" }, "end": "2022-01-29T15:25:44.105391Z" }, "toMilliseconds": 1643469944105 }

		duration	Executes the input function and returns an object with execution time in milliseconds and result of that function.	duration<T>(valueToMeasure: () -> T): DurationMeasurement<T>	lambda	execution time of function/lambda in milliseconds	This gives the execution time of lambda not for the whole code			
		time	Executes the input function and returns a TimeMeasurement object that contains the start and end time for the execution of that function, as well the result of the function.	time<T>(valueToMeasure: () -> T): TimeMeasurement<T>	lambda	object that contains the start and end time for the execution of that function, as well the result of the function.				
		toMilliseconds	Returns the representation of a specified date-time in milliseconds.	toMilliseconds(date: DateTime): Number	DateTime	epoch in milliseconds				
		asExpressionString	Transforms a Path value into a string representation of the path.	asExpressionString(path: Path): String	Path = Array<PathElement = {| kind: "Object" | "Attribute" | "Array", selector: String | Number, namespace: Namespace | Null |}>	string representation of the path		[{ "kind":"Object", "selector":"customer", "namespace": null }, { "kind":"Attribute", "selector":"name", "namespace": null }]	%dw 2.4 import * from dw::util::Tree output application/json --- asExpressionString(payload)	".customer.@name"

		filterArrayLeafs	Applies a filtering expression to leaf or Path values of an array.	filterArrayLeafs(value: Any, criteria: (value: Any, path: Path) -> Boolean): Any	value (Any type - string, number, object, array, etc), lambda with 2 arguments- value and path	filtered data	filtration will be applied only on the values inside array...others will remain same...here in the example for "conditiononPath" we are filtering data to get all the values inside array which have type object	{ "k1": ["", true, {"k2":" test"},{"k3": 0} ,"hello"], "k4": "val", "k5": [3,8] }	%dw 2.4
									import * from dw::util::Tree
									output application/json
									--- 	
									{
									"conditionOnValue":payload filterArrayLeafs ((value, path) -> ! (value is String)),
									"conditionOnPath":payload filterArrayLeafs ((value, path) -> isObjectType(path))
									}
										{ "conditionOnValue": { "k1": [true, { "k2": "test" }, { "k3": 0 }], "k4": "val", "k5": [3, 8] }, "conditionOnPath": { "k1": [{ "k2": "test" }, { "k3": 0 }], "k4": "val", "k5": [ ] } }

	provides functions for handling values as tree-data structures. Introduced in DataWeave version 2.2.2.	filterObjectLeafs	Applies a filtering expression to leaf or Path values of keys in an object.	filterObjectLeafs(value: Any, criteria: (value: Any, path: Path) -> Boolean): Any	value (Any type - string, number, object, array, etc), lambda with 2 arguments- value and path	filtered data	filtration will be applied only on the values inside object...others will remain same..here in the example for "conditiononPath" we are filtering data to get all the values inside object which have type array	{
								"k1": ["", true, {"k2":"test"},{"k3": 0} ,"hello"],
								"k4": "val",
								"k5": [3,8]
								}
									%dw 2.4
									import * from dw::util::Tree
									output application/json
									--- 	
									
									{
									"conditionOnValue":payload filterObjectLeafs ((value, path) -> !(value is String)),
									"conditionOnPath":payload filterObjectLeafs ((value, path) -> isArrayType(path))
									}
										{
										"conditionOnValue": {
										"k1": [
										"",
										true,
										{
										
										},
										{
										"k3": 0
										},
										"hello"
										],
										"k5": [
										3,
										8
										]
										},
										"conditionOnPath": {
										"k1": [
										"",
										true,
										{
										
										},
										{
										
										},
										"hello"
										],
										"k5": [
										3,
										8
										]
										}
										}

		filterTree	Filters the value or path of nodes in an input based on a specified criteria.	filterTree(value: Any, criteria: (value: Any, path: Path) -> Boolean): Any	value (Any type - string, number, object, array, etc), lambda with 2 arguments- value and path	filtered data	filtration will be applied whole input (array+object) ...others will remain same...here in the example for "conditiononPath" we are filtering data to get all the values which have kind - Object type	{
								"k1": ["", true, {"k2":"test"},{"k3": 0} ,"hello"],
								"k4": "val",
								"k5": [3,8]
								}
									%dw 2.4
									import * from dw::util::Tree
									output application/json
									--- 	
									
									{
									"conditionOnValue":payload filterTree ((value, path) -> !(value is String)),
									"conditionOnPath":payload filterTree ((value, path) -> isObjectType(path))
									}
										{
										"conditionOnValue": {
										"k1": [
										true,
										{
										
										},
										{
										"k3": 0
										}
										],
										"k5": [
										3,
										8
										]
										},
										"conditionOnPath": {
										"k1": [
										
										],
										"k4": "val",
										"k5": [
										
										]
										}
										}

		isArrayType	Returns true if the provided Path value is an ARRAY_TYPE expression.	isArrayType(path: Path): Boolean	path = Array<PathElement = {| kind: "Object" | "Attribute" | "Array", selector: String | Number, namespace: Namespace | Null |}>	boolean	the true or false value will be based on 'kind' of last pathElement in path		%dw 2.4
									import * from dw::util::Tree
									output application/json
									--- 	
									
									{
									"checkArray1": isArrayType([{kind: "Object", selector: "user", namespace: null},
									{kind: "Attribute", selector: "name", namespace: null}]),
									"checkArray2": isArrayType([{kind: "Object", selector: "user", namespace: null},
									{kind: "Array", selector: "name", namespace: null}]),
									
									"checkObject1": isObjectType([{kind: "Object", selector: "user", namespace: null},
									{kind: "Attribute", selector: "name", namespace: null}]),
									"checkObject2" : isObjectType([{kind: "Array", selector: "user", namespace: null},
									{kind: "Object", selector: "name", namespace: null}]),
									
									"checkAttribute1": isAttributeType([{kind: "Object", selector: "user", namespace: null},
									{kind: "Attribute", selector: "name", namespace: null}])
									}
										{
										"checkArray1": false,
										"checkArray2": true,
										"checkObject1": false,
										"checkObject2": true,
										"checkAttribute1": true
										}

		isAttributeType	Returns true if the provided Path value is an ATTRIBUTE_TYPE expression.	isAttributeType(path: Path): Boolean	path = Array<PathElement = {| kind: "Object" | "Attribute" | "Array", selector: String | Number, namespace: Namespace | Null |}>	boolean	the true or false value will be based on 'kind' of last pathElement in path			
		isObjectType	Returns true if the provided Path value is an OBJECT_TYPE expression.	isObjectType(path: Path): Boolean	path = Array<PathElement = {| kind: "Object" | "Attribute" | "Array", selector: String | Number, namespace: Namespace | Null |}>	boolean	the true or false value will be based on 'kind' of last pathElement in path			

Tree		mapLeafValues	Maps the terminal (leaf) nodes in the tree.	mapLeafValues(value: Any, callback: (value: Any, path: Path) -> Any): Any	value (Any type - string, number, object, array, etc), lambda with 2 arguments- value and path	mapped value	We can modify the values of object/Array by mapping it to new value	{
								"key1": ["", true, {"key2":"test"},"hello"],
								"key3": "val"
								}
									%dw 2.4
									import * from dw::util::Tree
									output application/json
									--- 	
									
									payload mapLeafValues (value, path) -> upper(value)
										{
										"key1": [
										"",
										"TRUE",
										{
										"key2": "TEST"
										},
										"HELLO"
										],
										"key3": "VAL"
										}

		nodeExists	Returns true if any node in a given tree validates against the specified criteria.	nodeExists(value: Any, callback: (value: Any, path: Path) -> Boolean): Boolean	value (Any type - string, number, object, array, etc), lambda with 2 arguments- value and path	boolean	you can either use path[0] or path[-1] as in example	{
								"username": "mehak",
								"password": "8836t5"
								}
									%dw 2.0
									import * from dw::util::Tree
									output application/json
									--- 	
									payload nodeExists ((value, path) -> path[-1].selector == "password")
										TRUE

		attr	This function creates a PathElement to use for selecting an XML attribute and populates the type's selector field with the given string.	attr(namespace: Namespace | Null = null, name: String): PathElement	namespace(optional), string	PathElement = {| kind: "Attribute", selector: String | Number, namespace: Namespace | Null |}	create Attribute kind PathElement		%dw 2.0
									import * from dw::util::Values
									output application/json
									--- 	
									{
									"attr" : attr(null, "user"),
									"field" : field(null,"user"),
									"index" : index(2)
									}
										{
										"attr": {
										"kind": "Attribute",
										"namespace": null,
										"selector": "user"
										},
										"field": {
										"kind": "Object",
										"namespace": null,
										"selector": "user"
										},
										"index": {
										"kind": "Array",
										"namespace": null,
										"selector": 2
										}
										}

		field	This function creates a PathElement data type to use for selecting an object field and populates the type's selector field with the given string.	field(namespace: Namespace | Null = null, name: String): PathElement	namespace(optional), string	PathElement = {| kind: "Object" , selector: String | Number, namespace: Namespace | Null |}	create Object kind PathElement			
		index	This function creates a PathElement data type to use for selecting an array element and populates the type's selector field with the specified index.	index(index: Number): PathElement	Number	PathElement = {| kind: "Array" , selector: String | Number, namespace: Namespace | Null |}	create Array kind PathElement with index value as a selector			

		mask	This mask function replaces all simple elements that match the specified criteria.	mask(value: Null, fieldName: String | Number | PathElement): (newValueProvider: (oldValue: Any, path: Path) -> Any) -> Null	value, fieldname(having null value) with "value to be replaced with"	updated data with masked values	"with" keyword needs to be used	{
								"username": "mehak",
								"password": "8836t5",
								"testArray": [1,2],
								"testNull": null
								}
									%dw 2.0
									import * from dw::util::Values
									output application/json
									--- 	
									{
									"maskNull" : payload mask "testNull" with "I am null value",
									"maskEg1" : payload mask field("password") with "*******",
									"maskEg2": payload mask "password" with "*******",
									"maskEg3": payload mask 1 with "*"
									}
										{
										"maskNull": {
										"username": "mehak",
										"password": "8836t5",
										"testArray": [
										1,
										2
										],
										"testNull": "I am null value"
										},
										"maskEg1": {
										"username": "mehak",
										"password": "*******",
										"testArray": [
										1,
										2
										],
										"testNull": null
										},
										"maskEg2": {
										"username": "mehak",
										"password": "*******",
										"testArray": [
										1,
										2
										],
										"testNull": null
										},
										"maskEg3": {
										"username": "mehak",
										"password": "8836t5",
										"testArray": [
										1,
										"*"
										],
										"testNull": null
										}
										}

				mask(value: Any, selector: PathElement): (newValueProvider: (oldValue: Any, path: Path) -> Any) -> Any	value, pathElement with "value to be replaced with"					
				mask(value: Any, fieldName: String): (newValueProvider: (oldValue: Any, path: Path) -> Any) -> Any	value, fieldname with "value to be replaced with"					
				mask(value: Any, i: Number): (newValueProvider: (oldValue: Any, path: Path) -> Any) -> Any	value, index with "value to be replaced with"					

				update(objectValue: Object, fieldName: String): UpdaterValueProvider<Object>	Object, fieldname(String)					{
										"updateEg1": {
										"username": "mehak",
										"password": "***",
										"testArray": [
										1,
										2,
										{

				update(objectValue: Object, fieldName: PathElement): UpdaterValueProvider<Object>	Object, fieldname (PathElement - field ("fieldname"))					

Values	This utility module simplifies changes to values. Introduced in DataWeave version 2.2.2.	update	This update function updates a field in an object/array with the specified string value.	update(arrayValue: Array, indexToUpdate: Number): UpdaterValueProvider<Array>	Array,index(number)	updated object		{
								"username": "mehak",
								"password": "8836t5",
								"testArray": [1,2, {"password": 24376}],
								"testObjectArray": [{"password": 24376}],
								"testNull": null
								}
									%dw 2.0
									import * from dw::util::Values
									output application/json
									--- 	
									{
									"updateEg1": payload update "password" with "**",
									"updateEg2": payload update field("password") with "**",
									"updateEg3": payload.testArray update 2 with "**",
									"updateEg4": payload.testObjectArray update "password" with "**",
									"updateEg5": payload.testArray update index(2) with "**",
									"updateEg6": payload update ["testObjectArray", field("password")] with "I am password inside testObjectArray",
									"updateEg7": payload update "testNull" with "I was null",
									
									}
										{
										"updateEg1": {
										"username": "mehak",
										"password": "**",
										"testArray": [
										1,
										2,
										{
										"password": 24376
										}
										],
										"testObjectArray": [
										{
										"password": 24376
										}
										],
										"testNull": null
										},
										"updateEg2": {
										"username": "mehak",
										"password": "**",
										"testArray": [
										1,
										2,
										{
										"password": 24376
										}
										],
										"testObjectArray": [
										{
										"password": 24376
										}
										],
										"testNull": null
										},
										"updateEg3": [
										1,
										2,
										"**"
										],
										"updateEg4": [
										{
										"password": "**"
										}
										],
										"updateEg5": [
										1,
										2,
										"**"
										],
										"updateEg6": {
										"username": "mehak",
										"password": "8836t5",
										"testArray": [
										1,
										2,
										{
										"password": 24376
										}
										],
										"testObjectArray": [
										{
										"password": "I am password inside testObjectArray"
										}
										],
										"testNull": null
										},
										"updateEg7": {
										"username": "mehak",
										"password": "8836t5",
										"testArray": [
										1,
										2,
										{
										"password": 24376
										}
										],
										"testObjectArray": [
										{
										"password": 24376
										}
										],
										"testNull": "I was null"
										}
										}

				update(arrayValue: Array, indexToUpdate: String): UpdaterValueProvider<Array>	Array of Objects, key value					
				update(arrayValue: Array, indexToUpdate: PathElement): UpdaterValueProvider<Array>	Array, index(number)					
				update(value: Array | Object | Null, path: Array<String | Number | PathElement>): UpdaterValueProvider<Array | Object | Null>	Array/Object, [Array/object/PathElement)					
				update(value: Null, toUpdate: Number | String | PathElement): UpdaterValueProvider<Null>	null, string/number/PathElement					