Amtiri

AMTIRI SCRIPT

Flow

Generated on November 25, 2026

Flow

Case

`case` · `case(reference: any, value: any, sentence: any, [more: any...])`

Compares one value against a series of candidates and yields the matching result - a lookup table. Arguments after the reference are candidate and result in pairs; one final unpaired argument is the default. A candidate matches only when it is exactly equal, type included: case(1.0, 1, ...) does not match, although 1.0 == 1 is true.

Parameters

Name	Type	Description
`reference`	`any`	The value being looked up.
`value`	`any`	A candidate to compare the reference against.
`sentence`	`any`	The result when that candidate matches.
`more`	`any` (0 or more)	Further candidate and result arguments, in pairs. One final unpaired argument becomes the default.

Returns `any`: The matching result, the default, or null when neither exists.

Examples

```
case(COUNT, 1, "one", 2, "two", 3, "three", "many") # "three"
case(COUNT, 1, "one", "many") # "many"
case(TAG, "P-101", "Feed pump", "P-102", "Return pump") # "Feed pump"
```

Do while

`dowhile` · `dowhile(sentences: expression, check: boolean)`

Repeats an expression for as long as a condition holds, testing after each pass - so it always runs at least once. Note the parameters are the other way round from while.

Parameters

Name	Type	Description
`sentences`	`expression`	What to evaluate on each pass.
`check`	`boolean`	The condition, re-evaluated after every pass.

Returns `any`: The last pass's result.

Examples

```
dowhile("once", false) # "once"
```

For

`for` · `for(variable: name, initialize: any, check: boolean, sentences: expression, increment: expression)`

Repeats an expression over a counter: the counter is set once, tested before each pass and advanced after it.

Parameters

Name	Type	Description
`variable`	`name`	The counter's name, written bare rather than quoted.
`initialize`	`any`	The counter's starting value.
`check`	`boolean`	The condition, re-evaluated before every pass.
`sentences`	`expression`	What to evaluate on each pass.
`increment`	`expression`	How the counter advances after each pass.

Returns `any`: The last pass's result.

Examples

```
for(i, 1, i <= 3, i * 10, i + 1) # 30.0
for(i, 0, i < COUNT, REGISTERS[i], i + 1) # 65535
```

If

`if` · `if(condition: boolean, true_sentence: any, [false_sentence: any])`

Chooses between two values on a condition. With no false branch it yields null when the condition does not hold.

Parameters

Name	Type	Description
`condition`	`boolean`	The test that decides the branch.
`true_sentence`	`any`	The value when the condition holds.
`false_sentence`	`any` (optional)	The value when it does not. Defaults to null.

Returns `any`: Whichever branch was taken.

Examples

```
if(LEVEL > 90, "HIGH", "OK") # "OK"
if(PUMP_ON, 1) # 1
if(not(PUMP_ON), 1) # null
if(1, "yes", "no") # error
```

While

`while` · `while(check: boolean, sentences: expression)`

Repeats an expression for as long as a condition holds, testing before each pass.

Parameters

Name	Type	Description
`check`	`boolean`	The condition, re-evaluated before every pass.
`sentences`	`expression`	What to evaluate on each pass.

Returns `any`: The last pass's result, or null when the condition never held.

Examples

```
while(false, 1) # null
```

With

`with` · `with(value: any, paramName: name, function: expression)`

Evaluates an expression with a value bound to a name - a local alias, so an expensive sub-expression is written and computed once. The previous meaning of the name is restored afterwards.

Parameters

Name	Type	Description
`value`	`any`	The value to bind.
`paramName`	`name`	The name to bind it to, written bare rather than quoted.
`function`	`expression`	The expression to evaluate. Refer to the value by paramName.

Returns `any`: The expression's result.

Examples

```
with(LEVEL * 2, x, x + x) # 170.0
with(SETPOINTS, s, s.high - s.low) # 80.0
```

Outside the palette

Kalman filter

`ctrlKalman` · `ctrlKalman(settings: object, state: object, measurement: double)`

Deprecated: use the Kalman module. One step of a one-dimensional Kalman filter, smoothing a noisy measurement. Like ctrlPID it returns the whole state and takes its own previous state back in. Read the filtered value from the o field.

Parameters

Name	Type	Description
`settings`	`object`	The tuning, keys in lower case: q the process noise per second, r the measurement noise.
`state`	`object`	The variable holding the previous state - normally the variable being computed. Missing or null on the first pass, which initialises it.
`measurement`	`double`	The raw reading for this pass. A null reading leaves an existing state unchanged; the first null reading, with no state yet, creates one with o = 0.

Returns `object` : The new state: o the filtered value, p the estimate covariance, ts a whole number from the engine's clock. The first reading is taken as it is: o equals it.

Examples

```
ctrlKalman({q: 0.01, r: 0.5}, null, LEVEL).o # 42.5
ctrlKalman({q: 0.0, r: 0.5}, {o: 40.0, p: 0.5, ts: 0}, LEVEL).o # 41.25
ctrlKalman({q: 0.01, r: 0.5}, null, null).o # 0.0
```

Data Orchester: available in plant formulas.

PID controller

`ctrlPID` · `ctrlPID(settings: object, state: object, error: double)`

Deprecated: use the PID module. One step of a PID control loop. It returns the whole controller state, output included, so the variable this formula feeds is the state itself and the state parameter is that same variable - that self-reference is how the loop carries its integral and derivative terms between evaluations. Read the output from the o field.

Parameters

Name	Type	Description
`settings`	`object`	The tuning, keys in lower case: p, i and d gains, plus mino and maxo bounding the output. The integral term is held inside those same bounds, which is what stops it winding up.
`state`	`object`	The variable holding the previous state - normally the variable being computed. Missing or null on the first pass, which initialises it.
`error`	`double`	How far the process is from its setpoint, right now.

Returns `object` : The new state: o the output, a the accumulated integral, e the previous error, ts a whole number from the engine's clock. The first pass starts the loop at rest: o, a and e are 0.

Examples

```
ctrlPID({p: 2.0, i: 0.1, d: 0.0, mino: 0.0, maxo: 100.0}, null, 50.0 - LEVEL).o # 0.0
ctrlPID({p: 2.0, i: 0.0, d: 0.0, mino: 0.0, maxo: 100.0}, {a: 0.0, o: 0.0, e: 0.0, ts: 0}, 50.0 - LEVEL).o # 15
```

Data Orchester: available in plant formulas.

PWM controller

`ctrlPWM` · `ctrlPWM(settings: object, state: object, input: double)`

Deprecated: use the PWM module. One step of a pulse-width-modulation loop, turning a continuous input into an on/off duty cycle: in each period the output is on for the input's share of it. Like ctrlPID it returns the whole state and takes its own previous state back in. It only switches when it is evaluated, so drive it with a Scheduler. Read the on/off decision from the o field.

Parameters

Name	Type	Description
`settings`	`object`	The configuration, keys in lower case: period, the cycle length in seconds; minin and maxin, the inputs that give 0 % and 100 % on; and optionally mintoggle, the fewest seconds between two switchings (0 when left out).
`state`	`object`	The variable holding the previous state - normally the variable being computed. Missing or null on the first pass, which initialises it.
`input`	`double`	The demand, somewhere between minin and maxin.

Returns `object` : The new state: o the on/off output (a boolean), cs the cycle start and lt the last switching, both whole numbers from the engine's clock. The first pass returns o = false.

Examples

```
ctrlPWM({period: 60, minin: 0, maxin: 100}, null, LEVEL).o # false
ctrlPWM({period: 60, minin: 0, maxin: 100, mintoggle: 5}, {o: false, cs: 0, lt: 0}, LEVEL).o # true
```

Data Orchester: available in plant formulas.

Sequence

`exec` · `exec([sentences: expression...])`

Evaluates several expressions in order and yields the last one's result. This is the sentence list the grammar builds for an ordinary parenthesised (a, b) group.

Parameters

Name	Type	Description
`sentences`	`expression` (0 or more)	An expression to evaluate. They run in the order given.

Returns `any` : The last expression's result, or null when none were given.

Examples

```
exec(COUNT + 1, COUNT * 2) # 6.0
(COUNT + 1, COUNT * 2) # 6.0
```

List functions

`functions` · `functions()`

Every function name registered on the environment, sorted - an interpreter introspection aid.

Returns `[]` : The registered function names.

Examples

```
functions().arrayFilter(f, f == "TSAVERAGE") # ["TSAVERAGE"]
```

Gradient (old spelling)

`gadient` · `gadient(gradient_name: name, value: double)`

The old, misspelt name of gradient, still accepted so stored formulas keep working. It does exactly what gradient does; write gradient in new formulas.

Parameters

Name	Type	Description
`gradient_name`	`name`	The gradient's name, written bare rather than quoted - it is read as an identifier, not as text. A name the host has not registered is an error.
`value`	`double`	The reading to look up along the gradient.

Returns `colour` : The colour the gradient assigns to the value, as a colour number.

Examples

```
gadient(red2green, 1) == gradient(red2green, 1) # true
```

List variables

`variables` · `variables()`

Every variable name defined on the environment, sorted - an interpreter introspection aid.

Returns `[]` : The defined variable names.

Examples

```
variables().arrayFilter(v, v == "LEVEL") # ["LEVEL"]
```

This page describes Amtiri Script 5.5.0 as shipped with Data Orchester engine 6.22.0 and Site 1.27.0.