Amtiri

AMTIRI SCRIPT

Errors and missing values

Generated on November 25, 2026

A script can fail in two ways: it may not compile, or it may fail while it runs. And a value can be missing in two ways: it may be `null`, or the variable may never have been written. This chapter says what each place does with a failure, how to guard a value with `evl`, `nvl`, `ivl` and `inn`, and which names to keep clear of. The examples use the sample variables.

Compile errors

A script is compiled before it runs. It does not compile when its text breaks the syntax, when a literal is refused (a whole number out of range, an unknown escape), when it calls a function that does not exist, or when it gives a function the wrong number of arguments. A script that does not compile never runs at all. The message starts with `Syntax error at` and marks where compiling stopped; see Syntax.

Variable names are not checked when compiling. A name that is not a variable is found only when the script reads it.

- **In the formula editor**, the status line shows the message as you type.
- **In a variable's formula**, the engine logs an error with the variable's code and the message when the configuration loads, and that variable is not calculated.

Run-time errors

A script that compiles can still fail while it runs:

Cause	Example
Reading a name that is not a variable, or one never written	`NOPE`
Reading a field the record does not hold	`SETPOINTS.mid`
Reading past the end of a list	`REGISTERS[4]`
A condition that is not a boolean	`if(COUNT, 1, 2)`
A value of the wrong kind	`1 + "a"`, `"1" == 1`, `year(TAG)`
`null` where a value is needed	`LEVEL + null`
Text that does not convert	`to_double("1,5")`
A date that does not exist	`date(2026, 2, 30)`
A gradient the engine does not provide	`gradient(nope, 0.5)`

An error goes up through every operator and function around it, and the whole script fails, unless evl catches it on the way. `&&`, `||`, `if` and `case` evaluate only what they need, so an error in a part they skip never happens.

Some results look wrong but are not errors: a division by zero gives `Infinity` or `NaN`; `+`, `-`, `*` or `/` with `null`, a boolean or a date on its left gives `null`; and a comparison with `null` gives `true` or `false`.

```
NOPE # error
SETPOINTS.mid # error
REGISTERS[4] # error
if(COUNT, 1, 2) # error
year(TAG) # error
LEVEL + null # error
to_double("1,5") # error
date(2026, 2, 30) # error
gradient(nope, 0.5) # error
round(NOPE) + 1 # error
if(PUMP_ON, LEVEL, NOPE) # 42.5
1 / 0 # Infinity
null + 1 # null
COUNT == null # false
```

Where errors go

Place	When the script fails	When it does not compile
Variable formula	A warning in the log, with the variable's code and the message. The value stays as it was	An error in the log when the configuration loads. The variable is not calculated
Initial value	An error in the log. The variable starts without a value	The same
Merger filter	A warning in the Merger's messages. Nothing is written	The Merger does not run: `Filter does not compile`
Mailer enabler	The Mailer reports it in its status and messages	The same
WebService Out	No request is sent, and the module reports it in its status and messages	The module does not open: `Configuration invalid`
Dashboard widget property	Nothing is reported. The property falls back: a blank label, a visible widget, zero offsets	Nothing is reported, and the widget is not updated
Action button script	An error dialog	The same, on the click
Peer `script.execute`	The peer receives an error	The same

A dashboard shows no error, so a widget that stays blank or does not move is often a failing formula. Open its formula in the editor to see whether it compiles, and try it in a variable if it does.

The module pages list the exact messages: Logic modules, Mailer and WebService Out.

Guarding a value

Four functions give a default in place of a problem. Each takes the value to guard and the default:

Function	Gives the default when the value	Does not help with
evl	Fails to evaluate, for any reason: a never-written variable, a missing field, a bad conversion	`null`, `Infinity` and `NaN`: they are values, not errors
nvl	Is `null`	Errors, such as reading a never-written variable
ivl	Is `Infinity`, `-Infinity` or `NaN`. Any other number comes back as a decimal	Text or `null`: both are errors
inn	Is not a number (text, `null`, a boolean) or is `NaN`	`Infinity`

The default is evaluated only when it is needed, and an error in the default itself is not caught.

To guard	Write
A reading that may not have arrived yet	`evl(PT_101, 0.0)`
A field that may be missing	`evl(STATE.o, 0.0)`
A value that may be `null`	`nvl(MODE, "auto")`
A division whose divisor can reach zero	`ivl(FLOW / AREA, 0.0)`
A square root or logarithm of a reading that can go negative	`inn(sqrt(DP), 0.0)`
A condition over a reading that may not exist yet	`evl(PT_101 > 4.5, false)`

`evl` hides every error, a misspelt variable name included. Keep the expression it guards small, so that it catches only the failure you expect.

```
evl(NOPE, 0) # 0
nvl(NOPE, 0) # error
nvl(null, 0) # 0
evl(null, 0) # null
nvl(COUNT, 0) # 3
evl(SETPOINTS.mid, 50) # 50
evl(REGISTERS[4], 0) # 0
evl(to_double("1,5"), 0.0) # 0.0
evl(NOPE > 10, false) # false
evl(LEVEL / 0, 0.0) # Infinity
ivl(LEVEL / 0, 0.0) # 0.0
ivl(0 / 0, 0.0) # 0.0
ivl(COUNT, 0) # 3.0
ivl(null, 0) # error
inn(sqrt(-1), 0.0) # 0.0
inn(TAG, 0.0) # 0.0
inn(null, 0.0) # 0.0
inn(COUNT, 0) # 3
inn(1 / 0, 0.0) # Infinity
evl(NOPE, NOPE_TOO) # error
```

Never-written variables

In the engine, a variable exists from the moment it is first written: by its initial value, its formula, a module or a dashboard's input widget. Until then it is not `null`: it does not exist, and reading it fails with `Variable 'CODE' not found`, exactly as a name that is no variable at all.

- **After every start**, a variable that is neither persistent nor given an initial value does not exist until something writes it.
- **`null` does not create it.** A formula or a module that writes `null` to a variable that does not exist yet leaves it not existing.
- **`nvl` cannot guard it**: the read fails before `nvl` sees a value. Use `evl`.
- **A formula that reads it fails** each time it runs, with a warning in the log, and a Merger filter that reads it fails the same way. Give the variable an initial value, or guard the read.
- **A formula that reads its own code** reads its previous value, which does not exist on the first calculation.

```
Formula of RUN_MINUTES, recalculated every minute by a Scheduler:
    evl(RUN_MINUTES, 0) + if(PUMP_ON, 1, 0)
```

Loop and binding names

for, with, forEach, arrayFilter and tsFilter bind a name, such as the counter of a loop. They do so by writing it into the table that holds the plant's variables, shared by every formula and dashboard of the instance, and they put the previous value back when they finish.

- **While the call runs, the name hides a variable of the same code.** Inside `forEach(LIST, COUNT, …)`, `COUNT` is the current item, not the variable.
- **A name that was not a variable is left behind** as a variable holding `null`. Afterwards, reading it gives `null` instead of an error.
- **The name is also a trigger** of the formula that binds it, so the log reports it when the configuration loads, as it does any name that is not a variable.

Choose binding names that no variable of the plant uses.

```
evl(i, "missing") # "missing"
(for(i, 0, i < 3, i * 2, i + 1), evl(i, "missing")) # null
forEach([1, 2, 3], COUNT, COUNT * 10) # [10.0, 20.0, 30.0]
(forEach([1, 2], COUNT, COUNT), COUNT) # 3
```

Names to avoid

Every variable shares one table with names the engine and the dashboards use. Do not give a variable any of these codes:

Name	Why
`ORCHESTER`	The engine's own reference, through which the history functions reach the loggers. A variable with this code replaces it, and they fail
`E`, `PI`	The preset constants. A variable with either code hides it
`DEBUG`	The text `"true"` makes every write of a variable print to the server's standard output. Any value that is not text makes every write of every variable fail
`_VALUE_`	The value a Merger filter is given. A filter cannot read a variable with this code
`_VIEW_`, `_CALLER_`	The view and the button an Action button's script is given

The names that loops and bindings use belong on this list too: see Loop and binding names.

Next steps

- Types and conversions
- Values and types
- Patterns
- Logic modules

This page describes Amtiri Script 5.4.1 as shipped with Data Orchester engine 6.12.1.