EventScript Language Documentation

EventScript is the rule-based automation language used by EventRunner6 for creating home automation rules on Fibaro HC3 controllers. It provides an intuitive syntax for defining triggers, conditions, and actions in automation scenarios.

Table of Contents

Language Overview

EventScript uses a simple `triggerExpression => action` syntax where:

- **Triggers** define when a rule should execute
- **Actions** define what should happen when triggered
- **Properties** provide access to device states and controls

Basic Syntax

```
rule("triggerExpression => action")
```

Rules are defined using the `rule()` function with a string containing the trigger-action pattern. The trigger is an expression returning true or false, and when true the action is executed. It can thus be thought of as

```
IF trigger THEN action END
```

The trigger must be an "pure" expression and not contain any control statements or side effects. Ex. assignments or print statemenets. The reason being that while compiling the rules, the trigger part may be evaluated multiple times. The trigger part is inspected during compilation to find out what events causes the rule to be triggered. Ex. if an fibaro global variable or a device property is used as part of the expression, the rule will trigger when those change in the system.

Control Structures

EventScript supports standard control flow structures for implementing complex logic within rules.

Conditional Statements

Use conditional statements to execute code based on conditions:

```
-- Simple if statement
if <test> then
  <statements>
end

-- If-else statement
if <test> then
  <statements>
else
  <statements>
end

-- If-elseif-else statement (elseif can be repeated)
if <test> then
  <statements>
elseif <test2> then
  <statements>
else
```

```
  <statements>
end

case
  || <test> >> <statements>
  || <test> >> <statements>
  :
  || <test> >> <statements>
end
```

Examples:

```
rule("sensor:breached => if luxSensor:value < 100 then light:on end"
rule("@sunset => if house:isAllOff then alarm:arm else log('House no
```

Loop Statements

EventScript supports various loop constructs:

```
-- Numeric for loop
for i = 1, n[, step] do
  <statements>
end

-- Iterator for loop (arrays)
for _, v in ipairs(<list>) do
  <statements>
end

-- Iterator for loop (tables)
for k, v in pairs(<table>) do
  <statements>
end

-- While loop
while <test> do
  <statements>
end

-- Repeat-until loop
repeat
  <statements>
until <test>
```

Examples:

```
rule("@08:00 => for i=1,5 do lights[i]:on end")
rule("motionDetected => for _,light in ipairs(hallwayLights) do ligh
```

Assignment

EventScript supports various assignment patterns for working with variables and values.

Simple Assignment

Assign values to variables using the assignment operator:

```
var = <expr>
```

Examples:

```
rule("sensor:temp => temperature = sensor:temp")
rule("@morning => lightLevel = 80")
```

Multiple Assignment

Assign multiple values in a single statement:

```
var1, var2, ..., varn = expr1, expr2, ...
```

Functions can return multiple values, with the last expression supporting multiple return values:

```
var1, var2, var3 = 42, (function() return 3, 4 end)()
```

Examples:

```
rule("weatherUpdate => temp, humidity = weatherStation:temp, weather
```

Tables

Tables are the primary data structure in EventScript, used for arrays, dictionaries, and complex data organization.

Table Creation

Create tables using various syntaxes:

```
-- Array-style table
local v = { <expr1>, <expr2>, ..., <exprn> }

-- Dictionary-style table
local v = { <key1> = <expr1>, <key2> = <expr2>, ..., <keyn> = <exprn

-- Mixed table with computed keys
local v = { [<expr1>] = <expr2>, [<expr3>] = <expr4>, ..., [<exprn>]
```

Examples:

```
-- Device groups
livingRoomLights = {66, 67, 68}
deviceStates = { motion = false, door = "closed", temp = 22 }
sensorMap = { [101] = "kitchen", [102] = "bedroom" }
```

Table Access

Access and modify table values:

```
-- Dot notation (for string keys)
<table>.<key> = <expr>
value = <table>.<key>

-- Bracket notation (for any key type)
<table>[<expr>] = <expr>
value = <table>[<expr>]
```

Examples:

```
rule("motion:breached => deviceStates.motion = true")
rule("temp:value => sensorData[temp:id] = temp:value")
```

Expressions

Expressions in EventScript are used to create complex trigger conditions and perform calculations within rules.

Variables

EventScript supports both local and global variables with a specific scope resolution order.

Variable Declaration

```
-- Local variables (scoped to the current rule)
local v1, ..., vn [= expr1, ..., exprn]

-- Global variables (accessible across all rules)
v1, ..., vn [= expr1, ..., exprn]
```

Variable Resolution Order

When accessing a variable, EventScript checks in this order:

1. **Local EventScript variable** (rule-scoped)
2. **Global EventScript variable** (system-wide)
3. **Global Lua variable** (built-in functions and constants)

Variable Assignment

When assigning to a variable that doesn't exist, EventScript creates an EventScript Global variable by default.

Examples:

```
rule("@08:00 => local brightness = 80; lights:value = brightness")
rule("sensor:temp => temp = sensor:temp")  -- Creates global variabl
rule("motion:breached => if temp > 25 then fan:on end")  -- Uses glo
```

Constants

EventScript provides various types of constants for use in expressions.

Time Constants

Time values can be specified in `HH:MM:SS` or `HH:MM` format:

```
rule("sensor:breached & 23:00..05:00 => log('Breached at night')")
rule("@@00:00:10 => log('Ping every 10 seconds')")
```

Time Representation

- **Short times**: Times between 00:00 and 24:00, represented as seconds after midnight
- **Long times**: Epoch times (like Lua's `os.time()`) for absolute timestamps

An absolue long time [YEAR]/MONTH/DAY/HOUR.MIN[:SEC]

```
rule("post(#futureEvent,2027/10/04/23:00)")  -- Post October 4, 23:0
rule("post(#futureEvent,/12/23/18:00)")  -- Post on Xmas eve, this y
rule("#futureEvent => log('Future event'))")
```

Note that long times can't be compared to short times. To convert a long time to a short time, subtract midnight. Normally, need for such calculations are rare.

Predefined Constants

Constant	Type	Description
`sunset`	Short time	Sunset time, updates daily at midnight
`sunrise`	Short time	Sunrise time, updates daily at midnight
`dawn`	Short time	Dawn time, updates daily at midnight
`dusk`	Short time	Dusk time, updates daily at midnight
`now`	Short time	Current time (HH:MM:SS)
`midnight`	Long time	Midnight timestamp, updates daily
`wnum`	Number	Current week number

Examples:

```
rule("@sunset => outdoorLights:on")
rule("sensor:breached & sunrise..sunset => securityAlert()")
rule("wnum % 2 == 0 => weeklyMaintenance()")  -- Every other week
```

Operators

EventScript supports various operators for building complex expressions.

Logical Operators

Operator	Description	Example
`&`	Logical AND	`sensor:breached & 22:00..06:00`

`|`	Logical OR	`door:open | window:open`
`!`	Logical NOT	`!alarm:armed`

Arithmetic Operators

Operator	Description	Example
`+`	Addition	`temp1:value + temp2:value`
`-`	Subtraction	`sunset - 00:30`
`*`	Multiplication	`price * quantity`
`/`	Division	`total / count`
`%`	Modulo	`minute % 15 == 0`
`^`	Exponentiation	`base ^ power`

Comparison Operators

Operator	Description	Example
`==`	Equal	`temp:value == 22`
`!=` or `~=`	Not equal	`door:state != "closed"`
`<`	Less than	`lux:value < 100`
`<=`	Less or equal	`humidity <= 60`
`>`	Greater than	`temp:value > 25`
`>=`	Greater or equal	`battery >= 20`

Assignment Operators

Operator	Description	Example
`+=`	Add and assign	`counter += 1`
`-=`	Subtract and assign	`energy -= consumption`
`*=`	Multiply and assign	`scale *= factor`
`/=`	Divide and assign	`average /= count`

Coalesce Operator

Operator	Description	Example
`??`	Assign if not nil	`counter = a ??` 7

Examples:

```
rule("temp:value > 25 & humidity < 60 => fan:on")
rule("@sunset-00:30 => lights:on")  -- 30 minutes before sunset
rule("motion:breached => counter += 1; log('Motion count: %d', count
```

##Triggers
Triggers define the conditions under which rules should execute. The triggerExpression part of a rule can be a complex expression of triggers returning true or false

Daily Triggers

Execute rules at specific times during the day:

```
rule("@time => action")                         -- Trigger at specific ti
rule("@{time1,time2,...} => action")          -- Trigger at multiple tim
rule("@{time,catch} => action")               -- Catchup: Run if deploye
rule("12:00..sunset => action")               -- Time interval guard, mo
```

Daily triggers can only specify a time during the day. To invoke the rule and specific days add a guard to the triggerExpression to test that it is the right day.

Examples:

```
rule("@08:00 => lights:on")                   -- Turn on lights at 8 AM
rule("@{07:00,19:00} => securityCheck()")   -- Check security at 7 A
rule("@sunset => outdoorLights:on")          -- Turn on outdoor lights
rule("@15:00 & day('mon-fri') => outdoorLights:on") -- Turn on outdo
```

Interval Triggers

Execute rules at regular intervals:

```
rule("@@00:05 => action")      -- Every 5 minutes
rule("@@-00:05 => action")     -- Every 5 minutes, aligned to clock
```

Examples:

```
rule("@@00:15 => temperatureCheck()")       -- Check temperature eve
rule("@@-01:00 => hourlyReport()")          -- Generate report on th
```

Event Triggers

Respond to custom events:

```
rule("#myEvent => action")                    -- Trigger on custom event
rule("#myEvent{param=value} => action")      -- Trigger on event with p
```

Examples:

```
rule("#myEvent => temperatureCheck()")        -- Check temperature wh
rule("@sunset => post(#myEvent)")          --POst #MyEvent at suns
```

> **Note:** `#event` is shorthand for `{type='event'}`, and `#event{k1=v1,...}` expands to `{type='event', k1=v1, ...}`

Event matching

```
rule("#myEvent{x=42} => log('x is %s',env.event.x)")  -- Trigger on
rule("post(#myEvent{x=42})")                        -- Post event with
```

Note: the event that triggers the rule is available in the local variable ev.event.

```
rule("#myEvent{x='$v'} => log('x is %s',env.p.v)")  -- Trigger on ev
rule("post(#myEvent{x=42})")                        -- Post event with
```

Values of type string and starting with prefix '$' is considered a pattern, and will bind the variable after the '$' to the value if there is a match. Matched variables are available in env.p.*

```
rule("#myEvent{x='$v>8'} => log('x is %s',env.p.v)")  -- Trigger on
rule("post(#myEvent{x=9})")                          -- Post event with
```

Patterns can also have an operator constraints applied. The trigger event only match, and the local variable is bound, if the constraint is true. In the above example v must be greater than 8 for the trigger event to match. Possible operators are

Operator	Description	Example
`$var>value`	value greater than	`#ev{key="$v>8"}`
`$var<value`	value less than	`#ev{key="$v<8"}`
`$var>=value`	value greater or equal than	`#ev{key="$v>=8"}`
`$var<=value`	value greater or less than	`#ev{key="$v<=8"}`
`$var~=value`	value differs from	`#ev{key="$v~=8"}`
`$var==value`	value equal	`#ev{key="$v==8"}`
`$var<>value`	value string match	`#ev{key="$m<>date.*"}`

Device Triggers

React to device state changes:

```
rule("device:property => action")          -- Single device trigger
rule("{dev1,dev2,...}:property => action") -- Multiple device trigge
```

Examples:

```
rule("motionSensor:value => hallLight:on")
rule("{door1,door2,window1}:breached => alarm:on")
```

Trigger Variables

Use custom variables as triggers:

```
er.triggerVariables.x = 9     -- Define trigger variable
rule("x => action")           -- Trigger when x changes
rule("x = 42")               -- Change x to trigger above rule
```

Functions

trueFor Function

Execute actions when conditions remain true for a specified duration:

```
rule("trueFor(duration, condition) => action")
```

Examples:

```
rule("trueFor(00:05, sensor:safe) => light:off")
-- Turn off light when sensor has been safe for 5 minutes

rule("trueFor(00:10, door:open) => log('Door open for %d minutes', 1
-- Log message with again(n) re-enabling the condition n times
```

Date Functions

Date functions allow you to test properties of the current day and time ranges.

Day Testing Functions

```
wday('wed-thu,sun')      -- Test current weekday
day('1,13-last')         -- Test current day of month
month('jul-sep')         -- Test current month
date('* 10-12 * 8 *')    -- Full date/time test (min,hour,day,month,w
```

Day Function Syntax:

- `day('1,13-last')` - 'last' refers to the last day in month
- `day('1,lastw-last')` - First day and last week in month (lastw = last day - 6)

Examples:

```
rule("@15:00 & wday('mon-fri') => workdayRoutine()")     -- Weekday
rule("@08:00 & day('1') => monthlyReport()")             -- First da
rule("@sunset & month('dec-feb') => winterLights:on")    -- Winter m
rule("@12:00 & date('* * 1,15 * *') => biweeklyCheck()") -- 1st and
```

Time Range Testing

```
<time1>..<time2>        -- Test if current time is between times (inc
```

Examples:

```
rule("motion:breached & 22:00..06:00 => nightLight:on")  -- Night ho
rule("door:open & sunrise..sunset => dayAlert()")        -- Daytime
```

Log and Formatting Functions

Functions for logging and string formatting within rules.

Function	Description	Example
`log(fmt, ...)`	Log formatted message	`log('Temperature: %d°C', temp)`
`fmt(...)`	Format string without logging	`message = fmt('Status: %s', status)`
`HM(t)`	Format time as "HH:MM"	`timeStr = HM(os.time())`
`HMS(t)`	Format time as "HH:MM:SS"	`timeStr = HMS(os.time())`

Examples:

```
rule("sensor:temp => log('Temperature changed to %d°C', sensor:temp)
rule("@08:00 => log('Good morning! Time is %s', HM(now))")
rule("alarm:breached => message = fmt('ALERT at %s', HMS(now))")
```

Event Functions

Functions for posting, subscribing to, and managing events.

Function	Description	Example
`post(event, time)`	Post event at specified time	`post(#morningEvent, '08:00')`
`cancel(ref)`	Cancel posted event	`cancel(timerRef)`
`subscribe(event)`	Subscribe to remote events	`subscribe(#remoteEvent)`
`publish(event)`	Publish event to remote systems	`publish(#statusUpdate)`

`remote(deviceId, event)`	Send event to specific QuickApp	`remote(123, #customEvent)`

Examples:

```
rule("@sunset => timerRef = post(#lightsOff, '+01:00')")  -- Post ev
rule("motion:breached => cancel(timerRef)")               -- Cancel
rule("#remoteEvent => log('Received remote event')")     -- Handle r
rule("alarm:armed => remote(456, #securityAlert)")       -- Send to
```

Math Functions

Mathematical and statistical functions for calculations.

Function	Description	Example
`sign(t)`	Return sign of number (-1, 0, 1)	`direction = sign(temperature - 20)`
`rnd(min, max)`	Random number in range	`delay = rnd(5, 15)`
`round(num)`	Round to nearest integer	`temp = round(sensor:temp)`
`sum(...)`	Sum of arguments or table elements	`total = sum(1, 2, 3, 4)`
`average(...)`	Average of arguments or table	`avg = average(temps)`
`size(t)`	Length of array	`count = size(deviceList)`
`min(...)`	Minimum value	`lowest = min(temperatures)`
`max(...)`	Maximum value	`highest = max(temperatures)`
`sort(t)`	Sort table in place	`sort(values)`
`osdate(t)`	Same as os.date	`dateStr = osdate('%Y-%m-%d')`
`ostime(t)`	Same as os.time	`timestamp = ostime()`

Examples:

```
rule("sensors:temp => avgTemp = average(sensors:temp)")
rule("@08:00 => if rnd(1,10) > 5 then specialRoutine() end")
rule("temperatures:change => log('Range: %d to %d', min(temperatures
```

Global Variable Functions

Functions for managing Fibaro global variables.

Function	Description	Example
`global(name)`	Create global variable, returns false if exists	`isNew = global('myVariable')`
`deleteglobal(name)`	Delete global variable	`deleteglobal('oldVariable')`

Examples:

```
rule("@startup => if global('systemStatus') then systemStatus = 'run
rule("@shutdown => deleteglobal('temporaryFlag')")
```

Table Functions

Utility functions for working with tables and arrays.

Function	Description	Example
`adde(t, v)`	Add value to end of table	`adde(logEntries, newEntry)`
`remove(t, v)`	Remove value from table	`remove(activeDevices, deviceId)`

Examples:

```
rule("motion:breached => adde(motionLog, now)")
rule("device:offline => remove(activeDevices, device:id)")
```

Rule Functions

Functions for controlling rule execution.

Function	Description	Example
`enable(rule)`	Enable rule by ID or object	`enable(nightModeRule)`
`disable(rule)`	Disable rule by ID or object	`disable(dayModeRule)`

Examples:

```
rule("@sunset => enable(nightRules); disable(dayRules)")
rule("$vacationMode == true => disable(normalRoutines)")
rule("$maintenanceMode == false => enable(allRules)")
```

Property Functions

Property functions use the syntax `<ID>:<property>` for reading and `<ID>:<property> = <value>` for writing.

Device Properties

Property	Type	Description
`value`	Trigger	Device value property
`state`	Trigger	Device state property
`bat`	Trigger	Battery level (0-100)
`power`	Trigger	Power consumption
`isDead`	Trigger	Device dead status
`isOn`	Trigger	True if device/any in list is on
`isOff`	Trigger	True if device is off/all in list are off
`isAllOn`	Trigger	True if all devices in list are on
`isAnyOff`	Trigger	True if any device in list is off
`last`	Trigger	Time since last breach/trigger
`safe`	Trigger	True if device is safe
`breached`	Trigger	True if device is breached
`isOpen`	Trigger	True if device is open
`isClosed`	Trigger	True if device is closed
`lux`	Trigger	Light sensor value
`volume`	Trigger	Audio volume level
`position`	Trigger	Device position (blinds, etc.)
`temp`	Trigger	Temperature value
`scene`	Trigger	Scene activation event value
`central`	Trigger	Central scene event, {keyId=.., keyAttribute=...}

Device Control Actions

Property	Type	Description
`on`	Action	Turn device on
`off`	Action	Turn device off
`toggle`	Action	Toggle device state
`play`	Action	Start media playback
`pause`	Action	Pause media playback
`open`	Action	Open device (blinds, locks)
`close`	Action	Close device
`stop`	Action	Stop device operation
`secure`	Action	Secure device (locks)
`unsecure`	Action	Unsecure device
`wake`	Action	Wake up dead Z-Wave device
`levelIncrease`	Action	Start level increase
`levelDecrease`	Action	Start level decrease
`levelStop`	Action	Stop level change

Device Assignment Properties

Property	Description
`value = <val>`	Set device value
`state = <val>`	Set device state
`R = <val>`	Set red color component
`G = <val>`	Set green color component
`B = <val>`	Set blue color component
`W = <val>`	Set white color component
`color = <rgb>`	Set RGB color values
`volume = <val>`	Set audio volume
`position = <val>`	Set device position

`power = <val>`	Set power level
`targetLevel = <val>`	Set target dimmer level
`interval = <val>`	Set interval value
`mode = <val>`	Set device mode
`mute = <bool>`	Set mute state
`dim = <table>`	Set dimming parameters
`msg = <text>`	Send push message
`email = <text>`	Send email notification

Partition Properties

Property	Type	Description
`armed`	Trigger	True if partition is armed
`isArmed`	Trigger	True if partition is armed
`isDisarmed`	Trigger	True if partition is disarmed
`isAllArmed`	Trigger	True if all partitions are armed
`isAnyDisarmed`	Trigger	True if any partition is disarmed
`isAlarmBreached`	Trigger	True if partition is breached
`isAlarmSafe`	Trigger	True if partition is safe
`isAllAlarmBreached`	Trigger	True if all partitions breached
`isAnyAlarmSafe`	Trigger	True if any partition is safe
`tryArm`	Action	Attempt to arm partition
`armed = <bool>`	Action	Arm or disarm partition

Thermostat Properties

Property	Type	Descript
`thermostatMode`	Trigger/Action	Thermos operating mode
`thermostatModeFuture`	Trigger	Future thermost

		mode
`thermostatFanMode`	Trigger/Action	Fan operatin mode
`thermostatFanOff`	Trigger	Fan off status
`heatingThermostatSetpoint`	Trigger/Action	Heating setpoint
`coolingThermostatSetpoint`	Trigger/Action	Cooling setpoint
`heatingThermostatSetpointCapabilitiesMax`	Trigger	Max heating setpoint
`heatingThermostatSetpointCapabilitiesMin`	Trigger	Min heati setpoint
`coolingThermostatSetpointCapabilitiesMax`	Trigger	Max cool setpoint
`coolingThermostatSetpointCapabilitiesMin`	Trigger	Min cooli setpoint
`thermostatSetpoint = <val>`	Action	Set thermost setpoint

Scene Properties

Property	Type	Description
`start`	Action	Start/execute scene
`kill`	Action	Stop scene execution

Information Properties

Property	Type	Description
`name`	Info	Device name
`roomName`	Info	Room name containing device
`HTname`	Info	HomeTable variable name

`profile`	Info	Current active profile
`access`	Trigger	Access control event
`central`	Trigger	Central scene event
`time`	Trigger/Action	Device time property
`manual`	Trigger	Manual operation status
`trigger`	Trigger	Generic trigger property

List Operations

Operation	Description
`average`	Average of numbers in list
`sum`	Sum of values in list
`allTrue`	True if all values are true
`someTrue`	True if at least one value is true
`allFalse`	True if all values are false
`someFalse`	True if at least one value is false
`mostlyTrue`	True if majority of values are true
`mostlyFalse`	True if majority of values are false
`bin`	Convert to binary (1 for truthy, 0 for falsy)
`leaf`	Extract leaf nodes from nested table

Examples

Basic Device Control

```
rule("@08:00 => livingRoomLights:on")           -- Morning lights
rule("motionSensor:breached => hallwayLight:on") -- Motion activatio
rule("@sunset => {porch,garden,driveway}:on")    -- Evening outdoor l
```

Conditional Logic

```
rule("door:isOpen & @sunset => securityLight:on")      -- Security a
rule("trueFor(00:10, house:isAllOff) => alarm:arm")    -- Auto-arm w
rule("luxSensor:value < 100 & motion:breached => lights:on") -- Smar
```

Time-based Automation

```
rule("@{07:00,19:00} => thermostat:mode='auto'")        -- Twice dai
rule("22:00..06:00 & motion:breached => nightLight:on") -- Night mod
rule("@@00:30 => hvac:refresh")                         -- Regular m
```

List Operations

```
rule("temperatureSensors:average > 25 => fan:on")       -- Climate c
rule("{sensor1,sensor2,sensor3}:someTrue => alert:on")  -- Multi-sen
rule("allLights:isAnyOff => log('Some lights are off')") -- Status m
```

Advanced Scenarios

```
-- Vacation mode
rule("$vacationMode == true & motion:breached => securityAlert")

-- Energy saving
rule("trueFor(01:00, room:isAllOff) => hvac:targetLevel=18")

-- Weather-based automation
rule("weatherStation:temp < 0 & @06:00 => carHeater:on")
```

Rule management functions

Defining a rules returns a rule object, that has methods for managing and controling the rule

```
local r = rule("triggerExpression => actions")

r:disable() -- disables the rule and stops all timers started by the

r:enable() -- enables the rule. A daily or interval rule will start

r:start() -- trigger the rule manually

r:info() -- logs info about the rule
```

Best Practices

1. **Use meaningful device names** in your HomeTable variables
2. **Group related devices** in lists for easier management
3. **Combine time guards** with device triggers for smarter automation
4. **Use trueFor()** to avoid false triggers from brief state changes
5. **Test rules thoroughly** before deploying to production
6. **Document complex rules** with comments in your main function
7. **Use trigger variables** for inter-rule communication

8. **Leverage list operations** for aggregated device control