EventScript Tutorial: Home Automation for Beginners

Welcome to EventScript! This tutorial will teach you how to create powerful home automation rules using EventRunner6's intuitive rule-based language. We'll start with the basics and work our way up to advanced automation scenarios.

Table of Contents

Introduction

EventScript is a simple yet powerful language for creating home automation rules. Think of it as a way to tell your smart home: "When this happens, do that."

The basic structure is:

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

For example:

```
rule("@sunset => 467:on")  -- Turn on lights at sunset
```

This tells your home: "When sunset occurs, turn on the lights controlled by device with deviceID 467."

Note: The trigger (left side) must be a pure expression without side effects (no assignments or logging). Actions (right side) perform changes.

Try this

- Change the example to use `@sunrise` and a different message. Deploy and confirm the log updates at the expected time.
- Replace the action with a list of devices like `{345,467}:on` if you have a another device - this turns on 2 devices at sunset.

Quick Cheat Sheet: 10 essential patterns

These examples requires that you have setup variables for the different devices, like kitchenLight. This will be eplained later in the tutorial.

```
-- 1) Daily time
rule("@08:00 => kitchenLight:on")

-- 2) Multiple times
rule("@{07:00,19:00} => securityCheck()")

-- 3) Aligned interval (on the hour)
rule("@@-01:00 => log('Top of the hour')")

-- 4) Time-guarded device trigger
rule("motion:breached & 22:00..06:00 => nightLight:on")

-- 5) Device property trigger
rule("frontDoor:isOpen => log('Front door opened')")

-- 6) Threshold trigger
rule("tempSensor:value >= 26 => fan:on")

-- 7) When sensor is not breached for 10min, turn off hall light
rule("trueFor(00:10, !hallMotion:breached) => hallLight:off")

-- 8) Post event with delay
rule("@sunset => post(#evening, '+00:15')")

-- 9) List operation (average)
rule("temperatureSensors:value:average > 25 => hvac:on")

-- 10) Offset relative to sunset
rule("@sunset-00:30 => blinds:close")
```

Getting Started

All your rules are defined inside the `main` function of your EventRunner6 QuickApp:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Your rules go here
  rule("@08:00 => log('Good morning!')")
end
```

Let's break this down:

- `rule` - Function to define automation rules
- `var` - Table for storing variables accessible across all rules
- `triggerVar` - Table for variables that can trigger rules when changed

Try this

- Add a second rule in the same `main` for `@sunset` that logs a message.
- Temporarily add `@@00:00:10 => log('Ping every 10s')` and remove it after testing.

Your First Rules

Let's start with some simple rules to get you comfortable:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Turn on lights every morning
  rule("@08:00 => log('Good morning! Time to wake up')")

  -- Turn off lights every night
  rule("@23:00 => log('Good night! Time for bed')")

  -- Log the current time every hour
  rule("@@01:00 => log('The time is now %s', HM(now))")
end
```

Try this

- Change the `@23:00` rule to a short interval `@@00:00:30` and observe the logs, then revert.
- Add a multi-time trigger: `@{09:00,12:00,18:00} => log('Check-in')`.

Working with Variables

Variables let you store values and share data between rules. You can access global Lua functions, but it's better to define your own variables in `er.variables`:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Define variables
  var.x = 8
  var.y = 9
  var.homeMode = "normal"

  -- Use variables in rules
  rule("@08:00 => log('x + y = %d', x + y)")  -- Outputs: x + y = 17
  rule("@sunset => homeMode = 'evening'; log('Switched to %s mode',
end
```

Types of Rules

Time-based Rules

Time-based rules run at specific times of the day:

```
-- Single time
rule("@08:00 => log('Time for breakfast')")

-- Multiple times
rule("@{07:00,12:00,18:00} => log('Meal time!')")

-- Sunset/sunrise
rule("@sunset => outdoorLights:on")
rule("@sunrise => outdoorLights:off")

-- Time ranges (guards)
rule("motion:breached & 22:00..06:00 => nightLight:on")
```

Time rules only specify times during the day, 00:00-24:00. To restrict the action to specigfic days or months use a time guard.

Try this

- Add a weekday guard: `@07:30 & wday('mon-fri') => log('Weekday wake-up')`.
- Schedule two times in one rule: `@{07:00,19:00} => log('Twice a day')`.

When you add rules and restart EventRunner, you may have rules that should trigger in ex. the morning

```
rule("@08:00 => morningLight:on")
```

If you add the above rule at 10:00 in the morning and restart EventRunner, the rule will not run until the next day at 08:00. To run rules whos times have passed when EventRunner starts, we can add the keyword 'catch' to the time list

```
rule("@{08:00,catch} => morningLight:on")
```

If ER restarts after 08:00 this rule will run immediatly - a catch up - so we get our morning lights on.

Interval Rules

Interval rules run repeatedly at fixed intervals:

```
-- Every 5 minutes
rule("@@00:05 => log('5 minute check')")

-- Every hour (aligned to clock)
rule("@@-01:00 => log('Hourly report at %s', HM(now))")

-- Every 30 seconds
rule("@@00:00:30 => temperatureCheck()")
```

Try this

- Switch to aligned hourly: `@@-01:00 => log('Top of the hour')` and notice it fires at HH:00.
- Use a short test interval: `@@00:00:10 => log('10s test')` and remove after verifying.

Device-triggered Rules

These rules respond to changes in your smart devices:

```
-- Motion sensor triggers light
rule("motionSensor:breached => hallwayLight:on")

-- Door sensor triggers alert
rule("frontDoor:isOpen => log('Front door opened!')")

-- Temperature sensor triggers fan
rule("tempSensor:value > 25 => ceilingFan:on")

-- Multiple devices
rule("{door1,door2,window1}:breached => securityAlert()")
```

Try this

- Combine with a time guard: `motionSensor:breached & 22:00..06:00 => nightLight:on`.
- Trigger on a numeric threshold: `tempSensor:value >= 26 => fan:on`.

Using Lua Functions

You can access all global Lua functions within rules, including `fibaro.*` functions. You can also define your own functions:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Define custom functions
  function var.myFun(x, y)
    return x + y
  end

  function var.securityCheck()
    print('Running security check...')
  if frontDoorOpen() then
      print('WARNING: Front door is open!')
    end
  end

  -- Use functions in rules
  rule("@sunset => log('MyFun returns %d', myFun(8, 9))")
```

```
  rule("@{22:00,02:00,06:00} => securityCheck()")
end
```

Structuring Rules with Events

Use custom events to structure complex automations like subroutines. Events allow you to break down complex logic into manageable pieces and create sophisticated timing sequences.

Basic Event Posting

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Main trigger posts custom event
  rule("@sunset => post(#eveningRoutine)")
  rule("motion:breached & 22:00..06:00 => post(#nightMode)")

  -- Event handlers act like subroutines
  rule([[#eveningRoutine =>
    outdoorLights:on;
    securitySystem:arm;
    log('Evening routine activated')
  ]])

  rule([[#nightMode =>
    nightLight:on;
    wait(5);  -- Wait 5 seconds
    nightLight:off
  ]])
end
```

Event Posting with Delays and Times

Events can be posted immediately or scheduled for future execution using various time formats:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Immediate posting
  rule("@sunset => post(#eveningRoutine)")

  -- Post with relative delay (+ prefix for relative time)
  rule("@sunset => post(#lightsOff, +/01:30)")  -- 1 hour 30 minutes
  rule("motion:breached => post(#autoOff, +/00:05)")  -- 5 minutes l
  rule("door:open => post(#securityCheck, +/00:00:30)")  -- 30 secon

  -- Post at specific time (absolute time)
  rule("@sunset => post(#bedtimeRoutine, t/23:00)")  -- At 23:00 tod
  rule("@08:00 => post(#weekendCleanup, n/10:00)")  -- At 10:00 toda
```

```
  -- Post with date and time
  rule("alarm:armed => post(#vacationMode, 2024/12/24/18:00')")  --
  rule("alarm:armed => post(#vacationMode, /12/24/18:00')")  -- Chri

  -- Event handlers
  rule([[#lightsOff =>
    allLights:off;
    log('Auto lights off after sunset')
  ]])
  rule([[#autoOff =>
    if !motion:breached then
      lights:off
    end
  ]])
  rule([[#securityCheck =>
    if door:isOpen then
      log('ALERT: Door still open!')
    end
  ]])
  rule([[#bedtimeRoutine =>
    bedLights:on;
    wait(10);
    allLights:off
  ]])
end
```

Event Pattern Matching and Parameters

Events can carry parameters that can be matched or used in the handling rules:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Post events with parameters
  rule("temp:value > 25 => post(#temperatureAlert{level='high', temp
  rule("temp:value < 15 => post(#temperatureAlert{level='low', temp=
  rule("motion:breached => post(#motionDetected{room='kitchen', time

  -- Pattern matching on event parameters
  rule([[#temperatureAlert{level='high'} =>
    fan:on;
    log('High temperature alert: %d°C', temp)
  ]])

  rule([[#temperatureAlert{level='low'} =>
    heater:on;
    log('Low temperature alert: %d°C', temp)
  ]])
```

```
  -- Match events with any parameters (catch-all)
  rule("#temperatureAlert => log('Temperature event received')")

  -- Match specific room
  rule("#motionDetected{room='kitchen'} => kitchenLight:on")
  rule("#motionDetected{room='bedroom'} => bedroomLight:on")

  -- Match any motion event regardless of room
  rule("#motionDetected => log('Motion at %s in %s', time, room)")
end
```

Complex Event Sequences

Create sophisticated automation sequences using event chains:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Multi-step security sequence
  rule("@23:00 => post(#securitySequence{step='start'})")

  rule([[#securitySequence{step='start'} =>
    log('Starting security sequence...');
    post(#securitySequence{step='checkDoors'}, +/00:01)
  ]])

  rule([[#securitySequence{step='checkDoors'} =>
    if doors:isAllClosed then
      log('All doors secure');
      post(#securitySequence{step='checkWindows'}, +/00:01)
    else
      log('WARNING: Some doors are open!');
      post(#securitySequence{step='abort'})
    end
  ]])

  rule([[#securitySequence{step='checkWindows'} =>
    if windows:isAllClosed then
      log('All windows secure');
      post(#securitySequence{step='arm'}, +/00:01)
    else
      log('WARNING: Some windows are open!');
      post(#securitySequence{step='abort'})
    end
  ]])

  rule([[#securitySequence{step='arm'} =>
    securitySystem:arm;
    log('Security system armed - Good night!')
```

```
  ]])

  rule([[#securitySequence{step='abort'} =>
    log('Security sequence aborted - please check doors and windows'
  ]])
end
```

Cancelling Scheduled Events

You can cancel scheduled events using the reference returned by `post()`:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Store event reference for later cancellation
  rule([[motion:breached =>
    lightTimer = post(#autoLightsOff, +/00:10); -- Auto-off in 10 mi
    hallLight:on
  ]])

  -- Cancel the timer if motion detected again
  rule([[motion:breached & lightTimer =>
    cancel(lightTimer); -- Cancel previous timer
    lightTimer = post(#autoLightsOff, +/00:10) -- Start new timer
  ]])

  -- Handle the auto-off event
  rule([[#autoLightsOff =>
    if !motion:breached then
      hallLight:off;
      log('Auto-turned off hall light')
    else
      log('Motion still detected, keeping light on')
    end
  ]])
end
```

Advanced Event Patterns

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Event with multiple parameters and conditions
  rule("door:open => post(#doorEvent{door=door:name, time=now, weath

  -- Match events with specific combinations
  rule([[#doorEvent{door='front', weather='rain'} =>
    log('Front door opened in rain - activating entrance light');
    entranceLight:on
  ]])
```

```
  -- Use event parameters in calculations
  rule([[#doorEvent =>
    if door == 'front' & time > sunset then
      securityLight:on;
      post(#securityLightOff, +/00:05)
    end
  ]])

  -- Event broadcasting to multiple handlers
  rule([[alarm:breached =>
    post(#emergency{type='breach', location=alarm:location});
    post(#notification{message='Security breach detected'});
    post(#lightSequence{mode='emergency'})
  ]])

  -- Different handlers for the same event
  rule("#emergency{type='breach'} => securitySystem:alert")
  rule("#notification => log('ALERT: %s', message)")
  rule("#lightSequence{mode='emergency'} => allLights:on; strobeLigh
end
```

Event Debugging and Logging

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Log specific event types
  rule("#debug => log('Debug event: %s', message)")
  rule("#error => log('ERROR: %s', error)")

  -- Post debug events from other rules
  rule([[temp:value > 30 =>
    fan:on;
    post(#debug{message='Fan activated due to high temperature'})
  ]])
end
```

Try this

- Chain an event with a delay: in the action, do `post(#eveningFollowUp, +/00:15)` and handle `#eveningFollowUp` in another rule.
- Post a custom event from a device trigger and handle it separately.
- Create a multi-step sequence with parameters: `post(#sequence{step='start', room='kitchen'})`.
- Use event cancellation: store a timer reference and cancel it when conditions change.

Trigger Variables

Trigger variables are special variables that can trigger rules when their values change:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Define trigger variable
  triggerVar.homeOccupied = false

  -- Rule that triggers when variable changes
  rule("homeOccupied == true => "..
    "homeOccupied = false; "..
    "log('Someone is home!')")

  -- Set trigger variable from other rules
  rule("@sunset => homeOccupied = true")
  rule("motionSensor:breached => homeOccupied = true")
end
```

Try this

- Add another trigger variable (e.g., `triggerVar.night = false`) and a rule that reacts when it becomes true.
- Flip `homeOccupied` from a time-based rule to test the interaction with the device-triggered rule.

Setting up a Home Table

A Home Table (HT) is a structured way to organize your devices. This makes your rules much more readable and maintainable:

```
function QuickApp:main(er)
  local rule, var, triggerVar = er.rule, er.variables, er.triggerVar

  -- Define your home structure
  local HT = {
    kitchen = {
      sensor = {
        motion = 77,
        door = 99,
        temp = 101
      },
      light = {
        ceiling = 54,
        under_cabinet = 78,
        window = 82
      },
      appliances = {
        dishwasher = 203,
        coffee_maker = 204
```

```
      }
    },
    livingroom = {
      sensor = {
        motion = 88,
        lux = 89
      },
      light = {
        ceiling = 91,
        floor_lamp = 92,
        tv_backlight = 93
      },
      entertainment = {
        tv = 301,
        sound_system = 302
      }
    },
    bedroom = {
      sensor = { motion = 65, temp = 66 },
      light = { ceiling = 67, bedside = 68 }
    }
  }

  -- Make HT available to all rules
  var.HT = HT

  -- Now your rules are much more readable!
  rule("HT.kitchen.sensor.motion:breached => HT.kitchen.light.ceilin
  rule("HT.livingroom.sensor.lux:value < 100 => HT.livingroom.light:
  rule("@23:00 => HT.bedroom.light.bedside:on; wait(10); HT.bedroom.
end
```

Try this

- Add another room or device to the `HT` structure and reference it in a new rule.
- Create a list of lights (e.g., `var.allLights = {HT.livingroom.light.ceiling, HT.kitchen.light.ceiling}`) and turn them off together.

Common Home Automation Patterns

Here are some practical automation patterns for your home:

Morning Routine

```
rule("@07:00 & wday('mon-fri') => "..
  "HT.kitchen.light:on; "..
  "HT.kitchen.appliances.coffee_maker:on; "..
  "log('Good morning! Coffee is brewing')")
```

Security System

```
-- Arm security when leaving
rule("@{08:30,17:30} & wday('mon-fri') => "..
  "securitySystem:arm; "..
  "log('Security system armed')")

-- Motion during night hours
rule("HT.livingroom.sensor.motion:breached & 23:00..06:00 => "..
  "if !securitySystem:isArmed then "..
    "HT.livingroom.light:on; "..
    "post(#lightsOff, '+00:02') "..
  "else "..
    "log('SECURITY ALERT: Motion detected!') "..
  "end")

rule("#lightsOff => HT.livingroom.light:off")
```

Energy Saving

```
-- Turn off devices when no motion detected for 30 minutes
rule("trueFor(00:30, !HT.livingroom.sensor.motion:breached) => "..
  "HT.livingroom.entertainment:off; "..
  "log('Entertainment system turned off - no activity')")

-- Temperature-based fan control
rule("HT.livingroom.sensor.temp:value > 25 => HT.livingroom.fan:on")
rule("HT.livingroom.sensor.temp:value < 22 => HT.livingroom.fan:off"
```

See also: Reference for trueFor details and options in EventScript.md#truefor-function

Vacation Mode

```
-- Set vacation mode
triggerVar.vacationMode = false

rule("vacationMode == true => "..
  "log('Vacation mode activated'); "..
  "enable(vacationLights); "..
  "disable(normalRoutines)")

-- Random lights during vacation
rule("vacationMode & @{19:00,20:30,22:00} => "..
  "if rnd(1,10) > 5 then "..
    "HT.livingroom.light:on; "..
    "post(#vacationLightsOff, fmt('+00:%02d', rnd(30,90))) "..
  "end")
```

```
rule("#vacationLightsOff => HT.livingroom.light:off")
```

Weather-based Automation

```
-- Close blinds when sunny and hot
rule("weather:temp > 28 & weather:condition == 'sunny' => "..
  "HT.livingroom.blinds:close; "..
  "log('Closing blinds - hot and sunny')")

-- Turn on outdoor heater when cold
rule("weather:temp < 5 & @{17:00,18:00,19:00} => "..
  "HT.patio.heater:on; "..
  "post(#heaterOff, '+02:00')")  -- Auto-off after 2 hours

rule("#heaterOff => HT.patio.heater:off")
```

Best Practices

1. **Use meaningful names**: Name your devices and variables clearly

   ```
   -- Good
   var.HT = { kitchen = { light = { ceiling = 54 } } }

   -- Avoid
   var.devices = { k = { l = { c = 54 } } }
   ```

2. **Group related devices**: Use lists for similar devices

   ```
   var.allLights = {54, 67, 78, 91, 92}
   rule("@23:00 => allLights:off")
   ```

See also: List operations (sum, average, any/all) in EventScript.md#list-operations

3. **Use time guards**: Combine time ranges with other triggers

   ```
   rule("motion:breached & 22:00..06:00 => nightLight:on")
   ```

4. **Avoid false triggers**: Use `trueFor()` for conditions that might flicker

   ```
   rule("trueFor(00:05, !motion:breached) => lights:off")
   ```

See: EventScript.md#truefor-function

5. **Structure complex logic**: Use custom events for multi-step processes

   ```
   rule("@23:00 => post(#bedtimeRoutine)")
   rule("#bedtimeRoutine => lights:off; wait(10); security:arm")
   ```

Troubleshooting

Common Issues

1. **Rule not triggering**: Check your trigger syntax

```
 -- Wrong
 rule("motion:breach => lights:on")  -- Should be "breached"

-- Correct
rule("motion:breached => lights:on")
```

2. **Device not responding**: Verify device IDs

```
-- Check in Fibaro interface that device 54 exists
rule("motion:breached => 54:on")  -- Use device ID directly for
```

3. **Time rules not working**: Check time format

```
-- Wrong
rule("@8:00 => lights:on")  -- Should be "08:00"

-- Correct
rule("@08:00 => lights:on")
```

Debugging Tips

1. **Add logging**: Use `log()` to trace rule execution

```
rule("motion:breached => log('Motion detected!'); lights:on")
```

2. **Test with simple rules**: Start simple and build complexity

```
-- Test basic trigger first
rule("motion:breached => log('Motion works!')")

-- Then add action
rule("motion:breached => log('Motion works!'); lights:on")
```

3. **Use device IDs**: Test with numeric IDs before using Home Table

```
-- Test with ID first
rule("77:breached => 54:on")

-- Then use Home Table
rule("HT.kitchen.sensor.motion:breached => HT.kitchen.light:on")
```

Congratulations! You now have the foundation to create powerful home automation rules with EventScript. Start with simple rules and gradually build more complex automations as you become comfortable with the language. For detailed reference information, see the EventScript Language Documentation.

Glossary

- Trigger: The left side of a rule (`trigger => action`). A pure expression that, when true, causes the action to run. Examples: `@08:00`, `motion:breached`, `temp:value > 25`, `wday('mon-fri') & 22:00..06:00`.
- Action: The right side of a rule. One or more statements that perform side effects (device control, assignment, logging). Separate multiple statements with `;`.
- Guard: A condition that narrows when a trigger can fire, typically combined with `&` (AND). Examples: `wday('mon-fri')`, `22:00..06:00`, `lux:value < 100`.
- Event: A custom signal you can post and handle using `#name`. Post with `post(#name[, when])` and react with `rule("#name => ...")`.