Lua

Edit New Page

Sergey Polyakov edited this page on Jul 13, 2015 · 3 revisions

Utility functions

```
util.dump(value)
```

Prints contents of the *value* to debugging console.

Date/time library

Date/time table structure:

```
date_time = {
    year = 2014, -- Year number
    month = 12, -- Month of the year (1 is January)
    day = 31, -- Day of the month (from 1 to 31)
    hour = 23, -- Hour of the day (from 0 to 23)
    min = 59, -- Minute of the hour (from 0 to 59)
    sec = 59, -- Second of the minute (from 0 to 59)
    msec = 999 -- Millisecond of the second (from 0 to 999)
}
```

> Note that all timestamps returned by the library functions are in UTC time zone.

```
date_time.now()
```

Pages 4

Home

eventsrv.conf

Lua

TODO

Add a custom sidebar

Returns current date/time.

```
date_time.day_of_week(date)
```

Returns number of the day within a week for specified *date*. 1 is Monday.

```
date_time.add(t1, t2) , date_time.sub(t1, t2)
```

Functions for adding and substracting date/time values. Note that these functions don't modify their arguments.

```
t1 = date_time.now() -- Current date/time
t2 = date_time.add(t1, {day = 14}) -- Adds 2 weeks to the original date/time
```

```
date_time.compare(t1, t2)
```

Compares two date/time values and returns negative number if *t1* precedes *t2*, positive number if *t2* precedes *t1*, or zero if *t1* and *t2* are equal.

```
t1 = date_time.now() -- Current date/time
t2 = date_time.add(t1, {hour = 1}) -- Adds 1 hour to the original date/time
if date_time.compare(t1, t2) < 0 then
    util.dump("t1 is less than t2") -- This message is always printed
end
```

Object library

Object table structure:

```
object = {
    id = "{0123456789abcdef}", -- System ID
    class = "some.object.Class", -- Object class name
    type = "device", -- Object type
    name = "Foo Bar", -- Object name
    location = "/My Objects", -- Location
```

```
    parent = "{0123456789abcdef}", -- Parent object ID
    children = {"{0123456789abcdef}", ...}, -- Children object IDs
    owners = {"{0123456789abcdef}" = true, ...}, -- Owner user IDs
    settings = {param_1 = "value_1", ...} -- Object settings
}
```

Type-specific object tables contain all above values and may contain their own values.

User object:

```
user = {
    ...
    type = "user", -- Fixed for user objects
    first_name = "John",
    last_name = "Doe",
    system_name = "jdoe",
    groups = {"{0123456789abcdef}" = true, ...} -- User's group IDs
}
```

Device object:

```
device = {
    ...
    type = "device",
    serial_number = "0123456789", -- Serial number
    sensors = {"{0123456789abcdef}" = true, ...}, -- Sensor IDs
    actuators = {"{0123456789abcdef}" = true, ...} -- Actuator IDs
}
```

Service object:

```
service = {
    ...
    type = "service"
}
```

Group object:

```
group = {
    ...
    type = "group",
    users = {"{0123456789abcdef}" = true, ...} -- Member user IDs
}
```

Script object:

```
script = {
    ...
    type = "script"
}
```

Policy object:

```
policy = {
    ...
    type = "policy",
    users = {"{0123456789abcdef}" = true, ...} -- Member user IDs
}
```

Location object:

```
location = {
    ...
    type = "location"
}
```

Sensor object:

```
sensor = {
    ...
    type = "sensor",
    serial_number = "0123456789"
}
```

Actuator object:

```
actuator = {
    ...
    type = "actuator",
    serial_number = "0123456789"
}
```

```
object.get(id)
```

Returns object identified by *id* argument. *id* can be system ID or object's location.

```
obj = object.get("{0123456789abcdef}") -- Get object by ID
user = object.get("/My Users/John Doe") -- Get object by location
sensor = object.get("Sensor 1") -- Get object by location (relative to the calling s
```

```
object.get_children(parent_id)
```

Returns array of children objects.

```
object.query(query)
```

Returns list of objects matching the query (Hibernate's HQL syntax is used).

```
objs = object.query("from Device where serialNumber = '0123456789'")
for _, obj in ipairs(objs) do
```

```
    util.dump(obj.name)
end
```

Event library

Event table structure:

```
event = {
    id = "{0123456789abcdef}", -- Event ID
    class = "some.event.Class", -- Event class name
    source = "{0123456789abcdef}", -- Source object ID
    dest = "{0123456789abcdef}", -- Destination object ID
    time = {...}, -- Event generation timestamp (date/time table)
    server_time = {...}, -- Server timestamp (date/time table)
    field_1 = ..., -- Event fields (depending on class name)
    field_N = ...
}
```

`event.get()`

Returns event which caused execution of the current script.

`event.send(event)` , `event.send(event, dest)`

Send event and return its ID.

```
e = {
    class = "some.event.Class",
    my_field = "My value"
}
event.send(e) -- Sends signal event (no destination specified)
event.send(e, object.get("Some device")) -- Sends control event
```

`event.get(class, from)` , `event.get(class, from, to)`

Returns list of events with specified *class* name, registered between *from* and *to* timestamps. If *to* is not specified, current date/time is used.

```
from = date_time.sub(date_time.now(), {hour = 3}) -- Getting events registered durir
events = event.get("some.event.category.*", from) -- Using wildcard matching
for _, e in ipairs(events) do
    util.dump(e)
end
```

Sensor library

`sensor.get()`

Returns sensor object which caused execution of the current script.

`sensor.get_value()`

Returns value of the sensor which caused execution of the current script.

`sensor.get_value(sensor)`

Returns sensor (or actuator) value. *sensor* can be object ID, location or object table.

`sensor.set_value(actuator)`

Sets actuator value. *actuator* can be object ID, location or object table.

`sensor.get_values(sensor, from, to)` , `sensor.get_values(sensor, from)`

Returns list of sensor values between *from* and *to* timestamps. If *to* is not specified, current date/time is used.

Each sensor value is accompanied with timestamp (date/time table):

```
sens = object.get("Some sensor")
t = date_time.sub(date_time.now(), {min = 10}) -- Get values for last 10 minutes
vals = sensor.get_values(sens, t)
for _, s in ipairs(vals) do
    util.dump(s.time)
    util.dump(s.value)
end
```

Script library

```
script.first_run()
```

Returns *true* if current script is being executed for the first time since server's startup.

```
if script.first_run() then
    -- Perform some script initialization
end
```

```
script.this()
```

Returns table which describes currently executed script (see "Object library" section for the details).

```
script.context()
```

Returns persistent script's context table. The table is stored to database after script is finished.

```
ctx = script.context()
if not ctx.initialized then
    ctx.initialized = true -- Custom table key
end
```

```
script.shared_context(name)
```

Returns shared context table identified by *name* argument. Shared context data is accessible from different scripts. The table is stored to database after script is finished.

```
ctx = script.shared_context("shared_data")
if not ctx.initialized then
    ctx.initialized = true
end
```

```
script.run(script)
```

Executes script. *script* can be script ID, location or object table.

```
script.run("{0123456789abcdef}") -- Run script by ID
script.run("/My Scripts/Test script") -- Run script by location
```

```
script.schedule(script, time)
```

Schedules delayed script execution (script is executed exactly once). *time* can be delay in milliseconds or date/time table.

```
script.schedule(script.this(), 5000) -- Run current script in 5 seconds
script.schedule(script.this(), {hour=12, min=0}) -- Run current script at specified
```