

# Scripting Reference

---

Shadow Version 4.0

[www.motionshadow.com](http://www.motionshadow.com)

Copyright © 2021 Motion Workshop. All rights reserved.

The coded instructions, statements, computer programs, and/or related material (collectively the “Data”) in these files contain unpublished information proprietary to Motion Workshop, which is protected by US federal copyright law and by international treaties.

The Data may not be disclosed or distributed to third parties, in whole or in part, without the prior written consent of Motion Workshop.

The Data is provided “as is” without express or implied warranty, and with no claim as to its suitability for any purpose.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Getting Started</b>	<b>2</b>
2.1	What is Lua? . . . . .	2
2.2	Console . . . . .	2
2.3	Script File . . . . .	3
2.4	JSON-RPC Interface . . . . .	4
2.5	Basic Commands . . . . .	4
<b>3</b>	<b>Module Reference</b>	<b>7</b>
3.1	node . . . . .	7
	auto_connect . . . . .	9
	close . . . . .	9
	configuration . . . . .	10
	connect . . . . .	10
	connected . . . . .	10
	enumerate . . . . .	10
	erase . . . . .	10
	export . . . . .	11
	export_stream . . . . .	11
	get_data_path . . . . .	11
	get_node . . . . .	11
	get_node_ancestor . . . . .	12
	get_node_by_id . . . . .	12
	get_node_by_key . . . . .	12
	get_node_default . . . . .	12
	get_version . . . . .	12
	get_take . . . . .	12
	get_take_path . . . . .	12
	have_take . . . . .	13
	insert . . . . .	13
	insert_node . . . . .	13
	is_configured . . . . .	13
	is_connected . . . . .	13
	is_id . . . . .	14
	is_reading . . . . .	14
	is_taking . . . . .	14
	is_tracking . . . . .	14
	load_configuration . . . . .	14
	load_license . . . . .	15
	load_location . . . . .	15
	load_ractor . . . . .	15
	load_script . . . . .	15
	load_take . . . . .	16

---

load_take_source . . . . .	16
merge_configuration . . . . .	16
num_active . . . . .	17
num_configured . . . . .	17
num_connected . . . . .	17
num_reading . . . . .	17
open . . . . .	17
parent . . . . .	18
reading . . . . .	18
save_configuration . . . . .	18
scan . . . . .	18
set_configuration . . . . .	18
set_gain . . . . .	19
set_gain_sensor . . . . .	19
set_gselect . . . . .	19
set_hierarchy . . . . .	20
set_marker . . . . .	20
set_name . . . . .	20
set_pipeline_select . . . . .	20
set_pose . . . . .	21
set_pose_marker . . . . .	21
set_rotate . . . . .	21
set_source . . . . .	21
set_time_step . . . . .	22
set_track_gyroscope_bias . . . . .	22
set_translate . . . . .	22
start . . . . .	22
start_take . . . . .	22
stop . . . . .	23
stop_take . . . . .	23
track_take . . . . .	23
write_configuration . . . . .	23
3.2 node.system . . . . .	23
authorize_device . . . . .	25
calibrate_from_take . . . . .	25
configuration_matches_take . . . . .	26
export_default_type . . . . .	26
export_type . . . . .	26
export_stream_type . . . . .	27
file_canonical . . . . .	27
file_exists . . . . .	27
geocode_address . . . . .	27
get_default . . . . .	27
get_history . . . . .	27
get_local_mode . . . . .	28
get_location_list . . . . .	28

---

get_preference . . . . .	28
get_ractor_list . . . . .	28
get_service . . . . .	28
get_settings . . . . .	28
get_system_conf_list . . . . .	28
get_system_db_list . . . . .	29
get_system_info . . . . .	29
get_wifi_list . . . . .	29
get_zeroconf_list . . . . .	29
initialize . . . . .	29
initialize_session . . . . .	29
is_authorized . . . . .	29
is_filename . . . . .	30
is_initialized . . . . .	30
load_plugin . . . . .	30
local_mode . . . . .	30
location . . . . .	30
log . . . . .	30
open_database . . . . .	31
print . . . . .	31
quit . . . . .	31
ractor . . . . .	31
republish_services . . . . .	31
reset_ractor . . . . .	32
reset_wifi . . . . .	32
restart . . . . .	32
save_initialization . . . . .	32
save_location . . . . .	32
save_ractor . . . . .	32
set_date_time . . . . .	32
set_data_path . . . . .	33
set_default . . . . .	33
set_local_mode . . . . .	33
set_search_path . . . . .	33
set_timecode . . . . .	34
set_wifi . . . . .	34
shutdown . . . . .	34
sleep . . . . .	34
start_services . . . . .	34
trigger_state_change . . . . .	34
unique_id . . . . .	35
unload_plugin . . . . .	35
3.3 Classes . . . . .	35
configuration_node . . . . .	35
configuration_node_list . . . . .	36
export_type . . . . .	36

---

export_type_list . . . . .	37
location . . . . .	37
location_list . . . . .	37
preference . . . . .	37
ractor . . . . .	37
ractor_list . . . . .	37
service . . . . .	38
system_db . . . . .	38
system_db_list . . . . .	38
system_info . . . . .	38
take_channel . . . . .	38
take . . . . .	38
version . . . . .	39
wifi . . . . .	39
wifi_list . . . . .	40
zeroconf . . . . .	40
zeroconf_list . . . . .	40
3.4 Lua . . . . .	40
array . . . . .	40
boolean . . . . .	40
iterator . . . . .	40
nil . . . . .	40
number . . . . .	40
string . . . . .	41
table . . . . .	41

## 1 Introduction

The Motion Service provides a scripting interface based on the Lua programming language. All user interaction with the Motion Service is implemented through a custom set of Lua commands. This set of commands is defined in a Lua module named `node`.

This document provides an overview of the `node` commands, a detailed description of the embedded Lua interpreter, and a per function reference of the `node` module.

## 2 Getting Started

### 2.1 What is Lua?

From the *About* page on the Lua web site:

Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua is a scripting language. The Motion Service extends the Lua language with C++ functions and manages a persistent global Lua state.

There are multiple input methods for Lua commands, an interactive console, an HTTP remote procedure call interface, an HTTP GET file interpreter, and a regular file interpreter. The Motion Service accepts input from any source on a first in, first out basis.

This reference only describes Lua with respect to the Motion Service and the `node` module. For general Lua usage and syntax help refer to the excellent “Programming in Lua” book by Roberto Ierusalimsky.

### 2.2 Console

The Motion Service implements an interactive Lua console. The console is modeled after the stand alone Lua interpreter. An interactive console session might look something like Example 1, where the user input is highlighted.

The console is a command line application. To run it, open the Program Files folder and run `motion-cli --console` command. The interactive console is most useful to run advanced commands that are not available through the desktop application. For repeated tasks you may want to use script files instead.

```
console> print("Hello World")
Hello World
console> for i=1,5 do
    > print(i)
    > end
1
2
3
4
5
console> EOF (Ctrl-Z on Windows)
```

Example 1: An interactive console session.

## 2.3 Script File

A script file is simply a text file that contains a sequence of Lua commands. For example, here is a script file that records 10 seconds of motion data and then exports the results to a spreadsheet file.

The entire script file is parsed into one chunk and executed at once. This has the effect that all of the printed output is displayed after the script has completed.

```
-- Start a take. Sleep for 10 seconds, and then
-- export the data.
if node.start_take() then
    node.system.sleep(10)
    node.close()
    if node.export("take_10sec.csv") then
        print("Exported take")
    end
end

print("Done")
```

Example 2: An example script file.

Script files can be loaded with the `load_script` command, or using the **Open** command in the desktop application. The HTTP server embedded in the Motion Service will also handle GET requests for Lua script files.

## 2.4 JSON-RPC Interface

The Motion Service implements the JSON-RPC 2.0 protocol in its HTTP server. Simply POST a command request and receive a response. All commands available using this interface are in the `node` module.

```
POST /node.jsonrpc HTTP/1.1
Host: 127.0.0.1:32080
```

```
{
  "jsonrpc": "2.0",
  "method": "scan",
  "params": ["usb"],
  "id": "1"
}
```

```
HTTP/1.1 200 OK
```

```
{
  "jsonrpc": "2.0",
  "result": [false, "no devices found"],
  "id": "1"
}
```

Example 3: JSON-RPC example request and response over HTTP

The JSON-RPC interface is used for compatibility with JavaScript based client applications. Refer the the JSON-RPC 2.0 Specification for more details.

The Motion Service requires that the request `id` property is string valued and that `param` property is an array of positional function parameters. The JavaScript parameter types `Boolean`, `Null`, `Number`, and `String` are converted to their Lua equivalent.

## 2.5 Basic Commands

A core feature of the Motion Service is capturing motion data. The `start_take` command will start recording data streams from all devices in the current configuration.

```
node.start_take()
```

Example 4: Start a take.



Before running the `start_take` command there must be at least one Node in the configuration. There are many ways to add Nodes to the configuration. Use the `open` command to load a configuration file and replace the current configuration. Use the `scan` command to enumerate available devices and add them to the configuration. Use the `insert` command to manually add a single Node to the configuration. Note that two hyphens (`--`) comment out the rest of the line.

```
-- Open the configuration file "configuration.mNode".
-- Replaces current configuration.
node.open("configuration.mNode")

-- Scan for any Nodes that are plugged in. Add them to the current configuration.
node.scan()

-- Manually insert a Node entry, adding it to the current configuration.
node.insert("MyNode")
```

Example 5: Commands that add Nodes to the configuration.

```
-- Load default configuration file.
if node.system.file_exists("default/configuration.mNode") then
  node.load_configuration("default/configuration.mNode")
end

-- Scan for available devices if the configuration is empty.
if not node.is_configured() then
  node.scan()
end
```

Example 6: Part of system initialization, managing the configuration.

The Shadow Service attempts to load a default configuration at start time. First, the default configuration file is loaded if it exists. Second, if the configuration is still empty, the command `scan` is run. See Example 6 for part of the system initialization script. It is also an example of how scripting is used in the Shadow Service.

Most of the `node` commands return a `boolean`, `string` pair. In this case, the `boolean` indicates success or failure and the `string` will contain some description of the result. The result description `string` will be fairly generic. For more detailed error messages refer to the Shadow Service log.

Many `node` commands take an optional `id` parameter, denoted by a set of

```
-- Repeat the first example, check the result of the
-- command this time.
result, message = node.start_take()
if not result then
  -- Print out the error message.
  print(message)
  -- Or, interrupt execution and return an error.
  --error(message)
end
```

Example 7: An example with error checking.

square brackets ([id]). Each configured Node has a unique string identifier, called the `id` field. By specifying an `id` the command operates on a single configured Node. Otherwise, the command operates over all configured Nodes.

The Motion Service binds some custom classes to Lua. For example, the `configuration_node` class is used to reference a single device in the configuration list. You can read and write properties a `configuration_node` object to control the settings and data processing for a device.

```
-- A list is custom usertype. It can be iterated over using pairs.
list = node.configuration()
if list then
  -- Each item in the integer indexed list is a usertype as well.
  for _,item in pairs(list) do
    item.gselect = 8
  end

  -- Size operator.
  num_item = #list

  -- Access by index, must use items property. First item is at 1.
  item = list.items[1]

  -- Make a copy of the item stored in the list. You must do this
  -- to keep the item the around after list goes out of scope.
  item = copy_usertype(item)
end
```

Example 8: Working with a Lua usertype class.

## 3 Module Reference

This section provides a comprehensive list of all functions in the `node` module.

### 3.1 `node`

```
boolean, string auto_connect([boolean rescan])
boolean, string close([string id])
configuration_node_list configuration()
boolean, string connect([string id])
configuration_node_list connected()
configuration_node_list enumerate([string name])
boolean, string erase([string id])
boolean, string export(string filename,
    [string take_filename],
    [string option],
    [string file_type],
    [number mode])
boolean, string export_stream(string filename,
    [string take_filename],
    [string option],
    [string file_type])
boolean, string get_data_path()
configuration_node get_node(string key,
    type value)
array get_node_ancestor(string id)
configuration_node get_node_by_id(string id)
configuration_node get_node_by_key(number key)
configuration_node get_node_default()
version get_version()
take get_take()
boolean, string get_take_path()
boolean have_take()
boolean, string insert(string id)
configuration_node insert_node(configuration_node obj)
boolean is_configured([string id])
boolean is_connected([string id])
boolean is_id(string id)
boolean is_reading([string id])
```

---

```
boolean is_taking()
boolean is_tracking()
boolean, string load_configuration(string filename)
boolean, string load_license(string filename)
boolean, string load_location(string filename)
boolean, string load_ractor(string filename)
boolean, string load_script(string filename)
boolean, string load_take(string filename)
boolean, string load_take_source([string filename])
boolean, string merge_configuration(string source, string tree_doc)
number num_active(configuration_node_list container, boolean active)
number num_configured(boolean active)
number num_connected(boolean active)
number num_reading(boolean active)
boolean, string open(string filename)
boolean, string parent(string id, string parent_id)
configuration_node_list reading()
boolean, string save_configuration(string filename)
boolean, string scan([string filter])
boolean, string set_configuration(string key,
    type value,
    [string id],
    [boolean ignore_children])
boolean, string set_gain(number value, [string id])
boolean, string set_gain_sensor(number value, [string id])
boolean, string set_gselect(number value, [string id])
boolean, string set_hierarchy([string id],
    string fn,
    number rx,
    number ry,
    number rz,
    [number tx],
    [number ty],
    [number tz])
boolean, string set_marker([string id], number rx, number ry, number
rz, [number tx], [number ty], [number tz])
boolean, string set_name(string value, [string id])
boolean, string set_pipeline_select(number value, [string id])
```

---

```

boolean, string set_pose([string id])
boolean, string set_pose_marker([string id])
boolean, string set_rotate([string id], number x, number y, number
z)
boolean, string set_source(string value, [string id])
boolean, string set_time_step(number value, [string id])
boolean, string set_track_gyroscope_bias(number value, [string id])
boolean, string set_translate([string id], number x, number y, number
z)
boolean, string start([string id])
boolean, string start_take([string name], [string description])
boolean, string stop([string id])
boolean, string stop_take()
boolean, string track_take(string filename)
boolean, string write_configuration(string id)

```

**auto\_connect**

```

boolean, string auto_connect([boolean rescan])
  Effect Close, scan, and reconnect. Attempt to discover all
  Nodes and start reading from them. If the rescan pa-
  rameter is true then clear the configuration before the
  scan.
  Postcondition is_reading() == true
  Return true iff at least one Node is streaming data

```

**close**

```

boolean, string close([string id])
  Precondition id == nil or is_connected(id) == true
  id ~= nil or connected() ~= nil
  Effect Close existing connection to one or more configured
  Nodes.
  Postcondition is_connected([id]) == false
  Return true iff at least one Node connection was closed.

```

**configuration**`configuration_node_list configuration()`

- Precondition** `is_configured() == true`  
at least one Node exists in the configuration state
- Postcondition** `configuration()` is iterable
- Return** an iterable container of all configured Nodes. Returns nil if the configuration is empty

**connect**`boolean, string connect([string id])`

- Precondition** `is_configured([id]) == true`  
`is_connected([id]) == false`
- Effect** Connect to one or more configured Nodes.
- Postcondition** `is_connected([id]) == true`
- Return** true iff all requested Node connections are open.

**connected**`configuration_node_list connected()`

- Precondition** at least one Node is currently connected
- Postcondition** `connected()` is iterable
- Return** an iterable container of all connected Nodes.

**enumerate**`configuration_node_list enumerate([string name])`

- Precondition** `is_connected() == false`  
`connected() == nil`
- Effect** Enumerate all available Nodes and return a list. Optionally limit the devices by class name, for example "usb" or "bus".
- Parameter** name Limit the device enumerate to a single source type.
- Return** an iterable container of all discovered Nodes.

**erase**`boolean, string erase([string id])`

- Precondition** `is_configured([id]) == true`  
`is_connected() == false`  
`connected() == nil`
- Effect** Remove one or more configured Nodes from the configuration state.
- Postcondition** `is_configured([id]) == false`
- Return** true iff all requested Nodes are removed from the configuration state.

**export**

```
boolean, string export(string filename,  
    [string take_filename],  
    [string option],  
    [string file_type],  
    [number mode])
```

**Precondition** `system.is_filename(filename) == true`  
`system.file_exists(take_filename) == true`  
`file_type == nil or system.export_type(mode,  
file_type) ~= nil`

**Effect** Export a take to an external file format.

**Return** `true` iff the take is successfully exported to the requested output file.

Returns canonical filename of the exported file on success and error message on failure.

**export\_stream**

```
boolean, string export_stream(string filename,  
    [string take_filename],  
    [string option],  
    [string file_type])
```

**Effect** `export(filename, take_filename, option, file_type, 2)`

**get\_data\_path**

```
boolean, string get_data_path()
```

**Return** `true` iff the user data folder is set and available in the `system.get_preference().data_path` property  
the absolute path to the user data folder

**get\_node**

```
configuration_node get_node(string key,  
    type value)
```

**Precondition** `configuration() ~= nil`

**Effect** Get the configured Node with named property `key == value` .

**Return** the first Node in the current configuration state with named property `key == value` .

**get\_node\_ancestor**

array get\_node\_ancestor(string id)

**Precondition** is\_id(id) == true  
is\_configured(id) == true

**Return** array of id strings that are the ancestors of a configured Node.

**get\_node\_by\_id**

configuration\_node get\_node\_by\_id(string id)

**Effect** get\_node('id', id)

**get\_node\_by\_key**

configuration\_node get\_node\_by\_key(number key)

**Effect** get\_node('key', key)

**get\_node\_default**

configuration\_node get\_node\_default()

**Effect** Get the Node that defines the system wide defaults for the configuration.

**get\_version**

version get\_version()

**Return** a description of the software version currently running

**get\_take**

take get\_take()

**Precondition** have\_take() == true

**Return** a description of the currently loaded take.

**get\_take\_path**

boolean, string get\_take\_path()

**Precondition** have\_take() == true

**Return** true iff there is a take loaded and we created an absolute path to the take definition file.  
the absolute path to the mTake format file



**have\_take**

boolean have\_take()

**Precondition** is\_taking() == false

**Return** true iff a take is currently loaded and not actively recording.

**insert**

boolean, string insert(string id)

**Precondition** is\_id(id) == true  
 system.unique\_id(id) == true  
 is\_configured(system.unique\_id(id)) == false  
 is\_connected() == false  
 connected() == nil

**Effect** Insert a new Node into the configuration state. If id already exists in the current configuration, generate a unique identifier.

**Postcondition** is\_configured(system.unique\_id(id)) == true

**Return** true iff a new Node is inserted into the configuration state.  
 the new unique identifier that can be used to access the new Node.

**insert\_node**

configuration\_node insert\_node(configuration\_node obj)

**Precondition** is\_id(obj.id) == true  
 is\_configured(obj.id) == false  
 is\_connected() == false  
 connected() == nil

**Postcondition** is\_configured(obj.id) == true

**Return** the new Node that was inserted into the configuration list. Returns nil on any error condition.

**is\_configured**

boolean is\_configured([string id])

**Precondition** id == nil or is\_id(id) == true

**Return** true iff all requested Nodes exist in the configuration state.

**is\_connected**

boolean is\_connected([string id])

**Precondition** is\_configured([id]) == true

**Return** true iff all requested Nodes are currently connected.

**is\_id**

boolean is\_id(string id)

**Precondition** #id > 0

id consists of alphanumeric, underscore (\_), and hyphen (-) characters

**Return** true iff the input string is a valid Node identifier

**is\_reading**

boolean is\_reading([string id])

**Precondition** is\_connected([id]) == true

**Return** true iff all requested Nodes are currently reading data.

**is\_taking**

boolean is\_taking()

**Precondition** is\_reading() == true

**Return** true iff a take is currently in progress.

**is\_tracking**

boolean is\_tracking()

**Precondition** is\_reading() == true

**Return** true iff a skeleton position and angle tracking system is active.

**load\_configuration**

boolean, string load\_configuration(string filename)

**Precondition** system.file\_exists(filename) == true

is\_connected() == false

connected() == nil

**Effect** Load a configuration file. Replace the current configuration state. Clear the current take state.

**Postcondition** is\_configured() == true

have\_take() == false

**Return** true iff the configuration file is successfully read, parsed, and at least one Node is loaded into the configuration state.

**load\_license**

boolean, string load\_license(string filename)

**Precondition** system.file\_exists(filename) == true

**Effect** Load a license file. Add one or more device licenses to the local database.

**Return** true iff the license file is successfully loaded and parsed.

**load\_location**

boolean, string load\_location(string filename)

**Precondition** system.file\_exists(filename) == true

is\_connected() == false

connected() == nil

**Effect** Load a location file. Replace the current location state of the system. This is used to define the geomagnetic field reference.

**Return** true iff the location file is successfully loaded and parsed.

**load\_ractor**

boolean, string load\_ractor(string filename)

**Precondition** system.file\_exists(filename) == true

is\_configured() == false

configuration() == nil

**Effect** Load a ractor definition file. Replace the current ractor state of the system. This defines the scaling of the configuration hierarchy.

**Return** true iff the ractor file is successfully loaded and parsed.

**load\_script**

boolean, string load\_script(string filename)

**Precondition** system.file\_exists(filename) == true

**Effect** Load and execute a Lua script file.

**Return** true iff the script file is successfully read, parsed, and executed.

**load\_take**

```
boolean, string load_take(string filename)
  Precondition system.file_exists(filename) == true
                 is_connected() == false
                 connected() == nil
  Effect Load a take file. Replace the current take state. Replace
           the current configuration state.
  Postcondition is_configured() == true
                 have_take() == true
  Return true iff the take file is successfully read, parsed, and the
           associated configuration file is successfully loaded.
```

**load\_take\_source**

```
boolean, string load_take_source([string filename])
  Precondition filename == nil or load_take(filename) == true
                 have_take() == true
                 is_configured() == true
                 is_connected() == false
                 connected() == nil
                 system.configuration_matches_take() == true
  Effect Copy the data source list from the current take into
           the current configuration source list. Clear the current
           take state. Optionally pre-load a take to copy the data
           sources from.
  Postcondition have_take() == false
  Return true iff the take data files are loaded as the current
           configuration sources for all configured Nodes.
```

**merge\_configuration**

```
boolean, string merge_configuration(string source, string tree_doc)
  Precondition get_node('source', source) ~= nil
                 is_connected() == false
                 connected() == nil
  Effect Create a hierarchy based on the current list of devices
           which are from a scan merged with an mNode format
           configuration file. The structure and rest pose comes
           from the document. The actual hardware devices come
           from an enumeration.
  Postcondition is_configured() == true
  Return true iff the configuration file contains a matching set of
           devices and they are successfully merged into the con-
           figuration state.
```

**num\_active**

number num\_active(configuration\_node\_list container, boolean active)

**Effect** Return the number to devices that are in a particular state, configured, connected, or reading. Filter by whether the device has sensor hardware attached or not.

**Return** return the number of Nodes in the input container. If the `active` parameter is true, filter by devices with hardware attached.

**num\_configured**

number num\_configured(boolean active)

**Effect** num\_active(configuration(), active)

**num\_connected**

number num\_connected(boolean active)

**Effect** num\_active(connected(), active)

**num\_reading**

number num\_reading(boolean active)

**Effect** num\_active(reading(), active)

**open**

boolean, string open(string filename)

**Precondition** system.file\_exists(filename) == true  
is\_connected() == false or close()  
is\_connected() == false  
connected() == nil

**Effect** Load a configuration, license, location, Lua script, or take file. Detect the file type and call the appropriate file handler. If there is a connected device, close it before loading the file and then reconnect.

**Return** true iff the file is of a valid type and the file handler successfully loaded the file.

**parent**

boolean, string parent(string id, string parent\_id)

**Precondition** is\_connected() == false  
is\_configured(id) == true  
is\_configured(parent\_id) == true

**Return** true iff the file is of a valid type and the file handler successfully loaded the file.

**reading**

configuration\_node\_list reading()

**Postcondition** reading() is iterable

**Return** an iterable container of all connected Nodes that are currently streaming data.

**save\_configuration**

boolean, string save\_configuration(string filename)

**Precondition** system.is\_filename(filename) == true  
is\_configured() == true

**Effect** Write the current configuration state to a mConf format file.

**Postcondition** system.file\_exists(filename) == true

**Return** true iff the configuration state is successfully written to the requested file.

**scan**

boolean, string scan([string filter])

**Precondition** is\_connected() == false  
connected() == nil

**Effect** Enumerate all available Nodes not currently configured and insert them into the current configuration state. Optionally limit the devices by class name, for example 'usb' or 'bus'. Use `enumerate`, `insert_node`, and `merge_configuration` to do the work.

**Postcondition** is\_configured() == true

**Return** true iff at least one new Node is inserted into the configuration state.

**set\_configuration**

boolean, string set\_configuration(string key,

*type* value,  
[string id],

[boolean ignore\_children])

**Precondition** `is_configured([id]) == true`

**Effect** Set a named member property of one or more configured Nodes. Class member properties in Lua are table entries, indexed by name. For example, `object.property = 1` is equivalent to `object["property"] = 1`. This is a convenience function to set a named property of one or all `configuration_node` objects in the current configuration state. If `id` is set, then set a property on that Node and any other Node that matches `parent == id`, or devices in the same bus. If `id` is not set, operate over all Nodes in the configuration.

**Return** `true` iff the named property key was set to value for at least one Node.

### `set_gain`

boolean, string `set_gain(number value, [string id])`

**Precondition** `is_taking() == false`

**Effect** `set_configuration("gain", value, id)`. Set the `gain` property of one or all configured Nodes. This adjusts the orientation output for specific applications. A value of 0 denotes smooth output, while a value of 1 denotes responsive output.

### `set_gain_sensor`

boolean, string `set_gain_sensor(number value, [string id])`

**Precondition** `is_taking() == false`

**Effect** `set_configuration("gain_sensor", value, id)`. Set the `gain_sensor` property of one or all configured Nodes. This adjusts post-filtering of the sensor data. A value of 0 denotes smoothest output, while a value of 1 denotes no filtering. Note that this does not effect the orientation output since the filter is applied after the orientation is computed.

### `set_gselect`

boolean, string `set_gselect(number value, [string id])`

**Precondition** `is_connected(id) == false`

**Effect** `set_configuration("gselect", value, id)`. Set the `gselect` property of one or all configured Nodes. Select the output range of the accelerometer. If a range is not available in hardware, the next smaller range is selected.

**set\_hierarchy**

```
boolean, string set_hierarchy([string id],
    string fn,
    number rx,
    number ry,
    number rz,
    [number tx],
    [number ty],
    [number tz])
```

**Precondition** `is_taking() == false`  
`is_configured([id]) == true`

**Effect** Set properties in the rest pose for one or more configured Nodes. Calls the `set_node_rotate`, `set_node_translate`, or `set_node_marker` function.

**set\_marker**

```
boolean, string set_marker([string id], number rx, number ry, number
    rz, [number tx], [number ty], [number tz])
```

**Effect** `set_hierarchy(id, "set_node_marker", x, y, z, tx, ty, tz)`

**set\_name**

```
boolean, string set_name(string value, [string id])
```

**Precondition** `is_connected(id) == false`

**Effect** `set_configuration("name", value, id)`. Set the name property of one or all configured Nodes.

**set\_pipeline\_select**

```
boolean, string set_pipeline_select(number value, [string id])
```

**Summary** Choose sensors that are active in the orientation tracking system. Set to 0 for all available, -1 for no tracking. Otherwise, set to the sum of 1 for Accelerometer, 2 for Magnetometer, and 3 for Gyroscope.

**Precondition** `is_taking() == false`

**Effect** `set_configuration("pipeline_select", value, id)`. Select the active orientation tracker of one or all configured Nodes.



**set\_pose**

boolean, string set\_pose([string id])

**Summary** Capture the current orientation of all Nodes and use it as a starting reference for all subsequent frames. Affects the local orientation outputs. If the Shadow plugin is active, then the local rotations are generated by the plugin and this command is a positional reset.

**Precondition** is\_reading() == true  
is\_taking() == false

**Effect** Define the identity orientation of all configured Nodes based on the current frame of data. The local orientation output is defined relative to the identity orientation. Optionally limit by id to a subtree of the configuration.

**set\_pose\_marker**

boolean, string set\_pose\_marker([string id])

**Summary** The performer should match the pose defined in the configuration file as closely as possible when sending this command. Updates the marker property in the configuration.

**Precondition** is\_reading(id) == true  
is\_taking() == false

**Effect** Create a hierarchical marker set for the current configuration. Capture the current pose as the initial offset of each Nodes relative to the skeleton definition. Optionally limit by id to a subtree of the configuration.

**set\_rotate**

boolean, string set\_rotate([string id], number x, number y, number z)

**Effect** set\_hierarchy(id, "set\_node\_rotate", x, y, z)

**set\_source**

boolean, string set\_source(string value, [string id])

**Summary** Set the source property of one or all configured Nodes. The source property defines the physical input source for a Node. For example, a Node may read from a USB device or over a Wi-Fi network.

**Precondition** is\_connected([id]) == false

**Effect** set\_configuration("source", value, id) .

**set\_time\_step**

boolean, string set\_time\_step(number value, [string id])

**Summary** Change the output sample rate for a device in seconds.

**Precondition** is\_connected([id]) == false

**Effect** set\_configuration("time\_step", value, id) .

**set\_track\_gyroscope\_bias**

boolean, string set\_track\_gyroscope\_bias(number value, [string id])

**Summary** Adjust or disable gyroscope bias tracking. User tunable parameter where 1 is maximum tracking and 0 disables tracking.

**Precondition** is\_taking() == false

**Effect** set\_configuration("track\_gyroscope\_bias", value, id) .

**set\_translate**

boolean, string set\_translate([string id], number x, number y, number z)

**Effect** set\_hierarchy(id, "set\_node\_translate", x, y, z)

**start**

boolean, string start([string id])

**Precondition** is\_reading([id]) == false  
is\_connected([id]) == true or connect([id]) == true

**Effect** Start reading data from one or more configured Nodes.

**Postcondition** is\_reading([id]) == true

**Return** true iff all requested Nodes are currently streaming data.

**start\_take**

boolean, string start\_take([string name], [string description])

**Precondition** is\_taking() == false  
is\_reading() == true or start() == true

**Effect** Start a take. Start writing Node data streams to files.

**Postcondition** is\_taking() == true

**Return** true iff a take is currently in progress.



---

```
[boolean calibrate_gyro])
boolean, string configuration_matches_take()
string export_default_type()
export_type_list export_type([number mode], [string file_type])
export_type_list export_stream_type([string file_type])
string file_canonical(string filename)
boolean file_exists(string filename)
boolean, string geocode_address(string address)
string get_default(string name)
table get_history()
string get_local_mode()
location_list get_location_list()
preference get_preference()
ractor_list get_ractor_list()
table get_service()
boolean, string get_settings(string id)
system_db_list get_system_conf_list()
system_db_list get_system_db_list()
system_info get_system_info()
wifi_list get_wifi_list()
zeroconf_list get_zeroconf_list()
boolean, string initialize()
boolean, string initialize_session()
boolean is_authorized(string uuid)
boolean is_filename(string filename)
boolean is_initialized()
boolean, string load_plugin([string name])
table local_mode()
boolean, string location(number latitude,
    number longitude,
    number elevation)
boolean, string log(string filename)
boolean, string open_database(string filename)
void print(type ...)
boolean quit()
boolean, string ractor(string name,
    number height,
    number arm,
```

```
    number leg)
boolean, boolean republish_services()
boolean reset_ractor()
boolean reset_wifi()
boolean restart()
boolean, string save_initialization()
boolean, string save_location([string filename])
boolean, string save_ractor([string filename])
boolean set_date_time(string date)
boolean set_data_path(string path)
boolean set_default(string name, string value)
boolean set_local_mode(string value)
boolean set_search_path(string search_path)
boolean, string set_timecode(string value)
boolean, string set_wifi(string hwid, string essid, string password)
boolean shutdown(string id)
boolean sleep(number second)
boolean, string start_services(table service)
boolean trigger_state_change(number code)
boolean, string unique_id(string id)
boolean, string unload_plugin([string name])
```

### authorize\_device

```
boolean, string authorize_device(string uuid,
    [string token])
Effect Load a license token into the local database. If the token
is not supplied, attempt to retrieve from the internet
license server.
```

### calibrate\_from\_take

```
boolean, string calibrate_from_take(string id,
    [boolean calibrate_accel],
    [boolean calibrate_gyro])
```

**Precondition** `have_take() == true`  
`is_configured(id) == true`  
`configuration_matches_take() == true`

**Effect** Use the current take to automatically generate calibration values for a Node. This command does not write the calibration values to the device. Use the `write_configuration` command to write the values to the Node onboard non-volatile memory.

`calibrate_accel` Enable accelerometer calibration. Disabled by default since they are factory calibrated and do not require location or mounting based calibration. Note that the function will detect accelerometer only devices and always enable calibration.

`calibrate_gyro` Enable gyroscope bias calibration mode. Disables all other calibration procedures since they are rotation based. Bias calibration data should be recorded in a static orientation. Does nothing for devices without gyroscopes onboard.

**Return** `true` iff the calibration values were successfully generated and copied into the system configuration.

#### `configuration_matches_take`

`boolean, string configuration_matches_take()`

**Precondition** `have_take() == true`  
`is_configured() == true`

**Return** `true` iff there is exactly a one to one mapping from configuration entries to take channels.

#### `export_default_type`

`string export_default_type()`

**Return** the unique identifier and file extension of the default exporter type used in the software.

#### `export_type`

`export_type_list export_type([number mode], [string file_type])`

**Effect** Get a list of exporters currently available in this software version. Optionally request a single type or file extension or limit to animation or data stream formats. Set mode to 0 to get all formats, 1 for animation formats, 2 for data stream formats, or the sum (bitmask) of multiple modes to select multiple formats.

**Return** an array of valid export types, or if a type is specified the single object that matches the file format.

**export\_stream\_type**

export\_type\_list export\_stream\_type([string file\_type])

**Effect** export\_type(2, file\_type)

**Return** an array of valid export stream types, or if a type is specified the single object that matches the file format

**file\_canonical**

string file\_canonical(string filename)

**Effect** Normalize a filename and return a relative URL style path that can be used to open the file.

**Return** relative path iff this file exists in one of the search paths.

**file\_exists**

boolean file\_exists(string filename)

**Return** true iff this file exists in the search path

**geocode\_address**

boolean, string geocode\_address(string address)

**Effect** Set the current system location based on an online geocode database lookup.

**Parameter** address Street address of the current location.

**Return** true iff the input address was successfully mapped to a valid location.

**get\_default**

string get\_default(string name)

**Return** value of named parameter in the local database system table

**get\_history**

table get\_history()

**Effect** The system keeps track of some commands, for example open, such that the user applications can have some history.

**Return** an associative array of interactive command parameters.

**get\_local\_mode**

string get\_local\_mode()

**Summary** Read back the system wide preference for the local rotation coordinate frames.

**Effect** get\_default('local\_mode', value)

**get\_location\_list**

location\_list get\_location\_list()

**Postcondition** get\_location\_list() is iterable

**Return** an iterable container of all previously entered geographic locations

**get\_preference**

preference get\_preference()

**Return** the current shared system preferences.

**get\_ractor\_list**

ractor\_list get\_ractor\_list()

**Postcondition** get\_ractor\_list() is iterable

**Return** an iterable container of all previously entered ractor definitions

**get\_service**

table get\_service()

**Return** an iterable table of the services loaded at start up.

**get\_settings**

boolean, string get\_settings(string id)

**Precondition** is\_configured(id) == true  
is\_reading() == false

**Effect** Load all of the settings based on the type of device and the system wide default node settings into a configured Node.

**get\_system\_conf\_list**

system\_db\_list get\_system\_conf\_list()

**Effect** Load from the local database. If nothing found there, then search in the "conf" folder for know Shadow skeletons.

**Return** a list of shadow configuration paths.



**get\_system\_db\_list**

system\_db\_list get\_system\_db\_list()

**Return** a list of system settings and user preferences from the local database.

**get\_system\_info**

system\_info get\_system\_info()

**Return** the current system information, an uptime and odometer clock.

**get\_wifi\_list**

wifi\_list get\_wifi\_list()

**Return** an iterable list of wireless access points in range.

**get\_zeroconf\_list**

zeroconf\_list get\_zeroconf\_list()

**Return** an iterable list of active wireless bus devices.

**initialize**

boolean, string initialize()

**Effect** Load the initialization script file, which contains all site specific preferences and starts the services.

**Return** true iff the system is initialized after a call to this function.

**initialize\_session**

boolean, string initialize\_session()

**Effect** Read user preferences and settings that affect how the streaming and recording sessions will operate. Initialize the current session such that the preferences match system behavior.

**is\_authorized**

boolean is\_authorized(string uuid)

**Return** true iff the Node with the uuid is in the configuration state and its auth property is 0 or greater than 4 which indicates the device has an active license.

**is\_filename**

boolean is\_filename(string filename)

**Return** true iff this filename is a valid name, it is in the search path.

**is\_initialized**

boolean is\_initialized()

**Return** true iff the system has already been initialized.

**load\_plugin**

boolean, string load\_plugin([string name])

**Precondition** is\_connected() == false  
connected() == nil

**Effect** Load available plugins. Optionally limit by name.

**local\_mode**

table local\_mode()

**Return** an associative array of local rotation mode constants.

**location**

boolean, string location(number latitude,  
number longitude,  
number elevation)

**Precondition** -90 < latitude < 90  
-180 < longitude < 180  
-1000 < elevation < 600000

**Effect** Set the current geographic location. Required to estimate the geomagnetic field which varies based on location.

**Parameter** latitude In decimal degrees.  
longitude In decimal degrees.  
elevation In meters.

**log**

boolean, string log(string filename)

**Precondition** is\_filename(filename) == true

**Effect** Set the current system log filename.

**open\_database**

boolean, string open\_database(string filename)

**Precondition** file\_exists(filename) == true

**Effect** Open the system database file. This is a SQLite version 3 compatible file.

**print**

void print(*type* ...)

**Effect** Replacement for the Lua print function. Capture print output such that we can redirect to various output targets.

**quit**

boolean quit()

**Effect** Close all services and threads. Exit the program after everything has been shut down.

**Return** Returns true.

**ractor**

boolean, string ractor(string name,  
number height,  
number arm,  
number leg)

**Precondition** -1 denotes no value, 0 denotes default value  
-1 denotes no value, 0 denotes default value  
-1 denotes no value, 0 denotes default value

**Effect** Set the current ractor definition by name. Scale the configuration hierarchy at load time based on the active definition. Use an existing name with -1 measurements to simply load the definition.

**height** In centimeters. Performer height.

**arm** In centimeters. Performer wing span.

**leg** In centimeters. Performer leg length.

**republish\_services**

boolean, boolean republish\_services()

**Effect** Republish the Zeroconf/Bonjour services on the local network. This is useful if the propagation of the services is taking a long time.

Returns true iff services were running and successfully republished.

**reset\_ractor**

boolean reset\_ractor()

**Effect** Clear all ractor preferences. Remove the list of ractors from the local database.

**reset\_wifi**

boolean reset\_wifi()

**Effect** Clear all wireless network preferences. Reset the Wi-Fi network to the default factory settings.

**restart**

boolean restart()

**Effect** Close all services and threads. Re-initialize the system.

**save\_initialization**

boolean, string save\_initialization()

**Effect** Save the initialization file based on the current state of the system.

**save\_location**

boolean, string save\_location([string filename])

**Effect** Save a location file based on the current state of the system. If no filename is given, save to the default path.

**save\_ractor**

boolean, string save\_ractor([string filename])

**Effect** Save a ractor definition file based on the current state of the system.

**set\_date\_time**

boolean set\_date\_time(string date)

**Summary** Set the system clock to the date and time specified in the input parameter. The time is specified in UTC/GMT rather than the local time zone. Format is an ISO string, for example 20101031T103015 for October 31, 2010 at 10:30.15 AM. Also accepts 2010-10-31T10:10:15 for compatibility.

**Return** true iff the clock was successfully updated

**set\_data\_path**

boolean set\_data\_path(string path)

**Summary** Set the output path for take data as well as the highest precedence entry in the system search path. If the input path name does not exist it will be created.

**Precondition** is\_initialized() == false  
path is a valid path name

**Postcondition** path exists

**Parameter** path name for take data and other user output files

**Return** true iff the input path exists and is the current data path

**set\_default**

boolean set\_default(string name, string value)

**Effect** Set the name value pair in the local database system table.

**Return** true iff the system value was saved successfully

**set\_local\_mode**

boolean set\_local\_mode(string value)

**Summary** Sets a system wide preference for the local rotation coordinate frames. Only loaded into a configuration as it is updated. This should precede calls to load files or scan for new Nodes.

**Effect** set\_default('local\_mode', value)

**Parameter** value Select the local rotation mode by string name. The valid parameters are 'WORLD' , 'SENSOR' , and 'WORLD\_HEADING' .

**set\_search\_path**

boolean set\_search\_path(string search\_path)

**Summary** Path names that do not exist are not added to the search path. The data path always takes precedence over the search path.

**Precondition** is\_initialized() == false  
search\_path is a valid list of path names, all of which exist

**Postcondition** all path names in search\_path that exist are in the system search path

**Parameter** search\_path List of semicolon (;) separated path names ordered by precedence

**Return** Returns true iff all entries in search\_path are valid, existing path names

### set\_timecode

boolean, string set\_timecode(string value)

**Summary** Sets a system wide timecode value.

**Parameter** value String formatted timecode formatted as hours, minutes, seconds, and frames. The semicolon denotes drop frame "hh:mm:ss:ff" or "hh:mm:ss:ff".

### set\_wifi

boolean, string set\_wifi(string hwid, string essid, string password)

**Return** Returns true iff the requested network is saved as the preferred network. Use empty hwid parameter to disable all networks but do not remove them from the list.

### shutdown

boolean shutdown(string id)

**Effect** Connect to a wireless bus by id. Call quit and then shutdown the system if we can. Requires an exclusive lock on the wireless bus.

**Return** Returns true.

### sleep

boolean sleep(number second)

**Summary** Block the current thread of execution for second seconds. The blocked thread maintains an exclusive lock on the Lua command pipeline.

**Parameter** second Sleep for this many seconds

**Return** Always returns true

### start\_services

boolean, string start\_services(table service)

**Effect** Start named services on requested ports.

### trigger\_state\_change

boolean trigger\_state\_change(number code)

**Precondition**  $0 < \text{code} < 5$

**Effect** Manually notify any monitor state change subscribers that something important changed. For example, when we modify a configuration parameter and then want to notify all other user programs of that change.

**unique\_id**

```
boolean, string unique_id(string id)
```

**Effect** Generate a valid unique Node identifier based on the input string.

**unload\_plugin**

```
boolean, string unload_plugin([string name])
```

**Precondition** `is_connected() == false`  
`connected() == nil`

**Effect** Unload currently loaded plugins. Optionally limit by name.

**3.3 Classes****configuration\_node**

```
print("id = " .. node.id)
print("name = " .. node.name)
print("source = " .. node.source)
print("gain = " .. node.gain)
```

```
-- Set the name of this node in the configuration. Any of the writable
-- properties may be set this way. Changes will propagate the shared, persistent
-- configuration state.
imu.name = "RightLeg"
```

Name	Type	Writeable
key	number	
id	string	
name	string	yes
source	string	yes
gselect	number	yes
time_step	number	yes
gain	number	yes
gain_sensor	number	yes
track_gyroscope.bias	number	yes
pipeline_select	number	yes
parent	string	yes
parent_id	string	
node_version	number	
active	boolean	
bus	boolean	
mass	number	
uuid	string	
auth	number	
flags	number	
rotate	string	
translate	string	
marker	string	
tree_doc	string	

#### **configuration\_node\_list**

List of `configuration_node` objects sorted by the `id` property.

```
-- Print out the unique id of each node in the configuration.
list = node.configuration()
for _,item in pairs(list) do
    print("id = " .. item.id)
end
```

Name	Type
items	iterator(configuration_node)

#### **export\_type**

Information about a file format supported by a built in or plugin take exporter.

Name	Type
id	string
description	string
option	string
mode	number



**export\_type\_list**

Name	Type
items	iterator(export_type)

**location**

Simple container for geographic location data.

Name	Type
key	number
name	string
address	string
latitude	number
longitude	number
elevation	number
active	number

**location\_list**

Name	Type
items	iterator(location)

**preference**

Name	Type
location	location
ractor	ractor
data_path	string
log_path	string
database_path	string
valid_location	boolean

**ractor**

Simple container for current ractor definition information.

Name	Type
key	number
name	string
height	number
arm	number
leg	number
active	number

**ractor\_list**

Name	Type
items	iterator(ractor)

**service**

Simple container describes a service.

<b>Name</b>	<b>Type</b>
description	string
port	number

**system\_db**

Single entry in the table of system database entries.

<b>Name</b>	<b>Type</b>
name	string
value	string
description	string

**system\_db\_list**

<b>Name</b>	<b>Type</b>
items	iterator(system_db)

**system\_info**

Simple container for system information. Includes an uptime clock in seconds and an odometer for the total number of samples read.

<b>Name</b>	<b>Type</b>
uptime	number
remaining	number
connected	number
odometer	number
battery_level	number
battery_state	number
serial_number	string
uuid	string

**take\_channel**

<b>Name</b>	<b>Type</b>
key	number
id	string
name	string
source	string
channel_mask	number

**take**

Simple container for current take information.

---

<b>Name</b>	<b>Type</b>
version	number
uuid	string
name	string
description	string
start	string
end	string
start_time	number
end_time	number
location	array
geomagnetic	array
time_step	number
num_node	number
num_frame	number
frame_stride	number
channel_mask	number
flags	number
working	string
stream_path	string
configuration_path	string
info_path	string
items	array

**version**

Information about the software version running on this device.

<b>Name</b>	<b>Type</b>
product_name	string
platform_name	string
version	string
is_shadow	boolean
is_bus	boolean
is_pro	boolean

**wifi**

Simple container for wireless access point information.

<b>Name</b>	<b>Type</b>
key	number
hwid	string
ssid	string
password	string
signal	number
channel	number
type	number
active	number

**wifi\_list**

Name	Type
items	iterator(wifi)

**zeroconf**

Simple container for zeroconf data service information.

Name	Type
name	string
hostname	string
address	string
port	number

**zeroconf\_list**

Name	Type
items	iterator(zeroconf)

**3.4 Lua****array**

By convention, an array is a **table** with integer indices. An array is indexed starting at 1 since Lua standard libraries adhere to this convention. See chapter 11.1 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/11.1.html>).

**boolean**

Basic Lua type. See chapter 2.2 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/2.2.html>).

**iterator**

Lua iterator function usable in generic **for** loops. See chapter 4.3.5 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/4.3.5.html>).

**nil**

Basic Lua type. See chapter 2.1 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/2.1.html>).

**number**

Basic Lua type. See chapter 2.3 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/2.3.html>).

**string**

Basic Lua type. See chapter 2.4 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/2.4.html>).

**table**

Basic Lua type. See chapter 2.5 in the "Programming in Lua" book for more details (<https://www.lua.org/pil/2.5.html>).