

# Scripting Reference

---

**Motion** Version 2.2

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

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

Copyright © 2015 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instructions, statements, computer programs, and/or related material (collectively the “Data”) in these files contain unpublished information proprietary to Motion Workshop, which is protected by US federal copyright law and by international treaties.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	Interactive Console . . . . .	2
2.3	Script File . . . . .	3
2.4	Basic Commands . . . . .	4
<b>3</b>	<b>Module Reference</b>	<b>6</b>
3.1	node . . . . .	6
	close . . . . .	8
	configuration . . . . .	8
	connect . . . . .	8
	connected . . . . .	8
	define_identity_pose . . . . .	8
	erase . . . . .	9
	export . . . . .	9
	export_stream . . . . .	9
	get_node . . . . .	9
	get_node_by_id . . . . .	10
	get_node_by_key . . . . .	10
	get_preview . . . . .	10
	get_raw . . . . .	10
	get_sensor . . . . .	10
	have_take . . . . .	10
	insert . . . . .	11
	is_configured . . . . .	11
	is_connected . . . . .	11
	is_id . . . . .	11
	is_reading . . . . .	11
	is_taking . . . . .	11
	is_tracking . . . . .	12
	load_configuration . . . . .	12
	load_location . . . . .	12
	load_ractor . . . . .	12
	load_script . . . . .	13
	load_take . . . . .	13
	load_take_source . . . . .	13
	open . . . . .	14
	reading . . . . .	14
	read_user_data . . . . .	14
	replay_take . . . . .	14
	save_configuration . . . . .	14
	scan . . . . .	15

---

set_configuration . . . . .	15
set_gain . . . . .	15
set_gain_sensor . . . . .	16
set_gselect . . . . .	16
set_name . . . . .	16
set_pipeline_select . . . . .	16
set_pose . . . . .	16
set_pose_marker . . . . .	17
set_source . . . . .	17
set_track_gyroscope_bias . . . . .	17
start . . . . .	17
start_take . . . . .	18
stop . . . . .	18
stop_take . . . . .	18
take . . . . .	18
track_take . . . . .	18
write_configuration . . . . .	18
write_user_data . . . . .	19
3.2 node.usb . . . . .	19
restore_configuration . . . . .	19
3.3 node.system . . . . .	19
calibrate_from_take . . . . .	21
configuration_matches_take . . . . .	21
console_client . . . . .	21
export_type . . . . .	22
file_canonical . . . . .	22
file_exists . . . . .	22
geocode_address . . . . .	22
get_default . . . . .	22
get_history . . . . .	22
get_local_mode . . . . .	22
get_location_list . . . . .	23
get_preference . . . . .	23
get_ractor_list . . . . .	23
get_service . . . . .	23
get_system_info . . . . .	23
get_wifi_list . . . . .	23
get_zeroconf_list . . . . .	23
initialize . . . . .	24
is_filename . . . . .	24
is_initialized . . . . .	24
load_plugin . . . . .	24
local_mode . . . . .	24
location . . . . .	24
log . . . . .	25
magnetic_from_take . . . . .	25

---

open_database . . . . .	25
print . . . . .	25
quit . . . . .	25
ractor . . . . .	25
republish_services . . . . .	26
reset_wifi . . . . .	26
restart . . . . .	26
save_initialization . . . . .	26
save_location . . . . .	26
save_ractor . . . . .	27
set_date_time . . . . .	27
set_data_path . . . . .	27
set_default . . . . .	27
set_local_mode . . . . .	28
set_search_path . . . . .	28
service_exists . . . . .	28
set_wifi . . . . .	28
shutdown . . . . .	29
sleep . . . . .	29
start_services . . . . .	29
test_wifi . . . . .	29
unique_id . . . . .	29
unload_plugin . . . . .	29
3.4 Classes . . . . .	30
ConfigurationContainer . . . . .	30
ConfigurationNode . . . . .	30
LocationNode . . . . .	30
LocationContainer . . . . .	30
PreferenceNode . . . . .	31
PreviewContainer . . . . .	31
PreviewNode . . . . .	31
RawContainer . . . . .	31
RawNode . . . . .	32
SensorContainer . . . . .	32
SensorNode . . . . .	32
ServiceNode . . . . .	32
SystemInfoNode . . . . .	32
RactorNode . . . . .	33
RactorContainer . . . . .	33
TakeNode . . . . .	33
WifiContainer . . . . .	33
WifiNode . . . . .	33
ZeroconfContainer . . . . .	34
ZeroconfNode . . . . .	34
3.5 Lua . . . . .	34
boolean . . . . .	34

---

iterator . . . . .	34
nil . . . . .	34
number . . . . .	34
string . . . . .	35
table . . . . .	35

## 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 light-weight programming language designed for extending applications. Lua is also frequently used as a general-purpose, stand-alone language. Lua is free software.

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 please refer to the excellent “Programming in Lua” book. A copy is available online at <http://www.lua.org/pil/>.

### 2.2 Interactive 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. Note that the user input is highlighted.

To start a console, open **Programs > Motion > Tools > Run Console (Command Prompt)**. The interactive console is most useful to run advanced commands that are not available through the user interface. 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 exports the results to a 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.fbx") 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 **File** > **Open** command in the Motion User Interface. The HTTP server embedded in the Motion Service will also handle GET requests for Lua script files. For example, the main Motion User Interface is simply the file `"/index.lua"`. The HTTP server executes the script file and returns the printed results as the file contents.

## 2.4 Basic Commands

The most basic feature of the Motion Service is capturing motion data. The `start_take` command will start logging data streams from all Nodes in the current configuration.

```
node.start_take()
```

Example 3: 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.xml".
-- Replaces current configuration.
node.open("configuration.xml")

-- 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 4: Commands that add Nodes to the configuration.

The Motion 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 5 for part of the system initialization script. It is also an example of how scripting is used in the Motion 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 Motion Service error log.

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



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

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

Example 5: Part of system initialization, managing the configuration.

```
-- 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 6: 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.

## 3 Module Reference

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

### 3.1 `node`

```
boolean, string close([string id])
ConfigurationContainer configuration()
boolean, string connect([string id])
ConfigurationContainer connected()
string define_identity_pose()
boolean, string erase([string id])
boolean, string export(string filename,
    [string option],
    [string type])
boolean, string export_stream(string filename, [string option])
ConfigurationNode get_node(string key,
    type value,
    [type compare])
ConfigurationNode get_node_by_id(string id)
ConfigurationNode get_node_by_key(number key)
PreviewContainer get_preview()
RawContainer get_raw()
SensorContainer get_sensor()
boolean have_take()
boolean, string insert(string id, [string bus], [string parent_id])
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_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 open(string filename)
ConfigurationContainer reading()
table read_user_data(string id)
boolean, string replay_take([string filename])
boolean, string save_configuration(string filename)
boolean, string scan([string filter])
boolean, string set_configuration(string key,
    type value,
    [string id])
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_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_source(string value, [string id])
boolean, string set_track_gyroscope_bias(number value, [string id])
boolean, string start([string id])
boolean, string start_take()
boolean, string stop([string id])
boolean, string stop_take()
TakeNode take()
boolean, string track_take(string filename)
boolean, string write_configuration(string id,
    [boolean write_factory])
boolean write_user_data(string id, table 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**

ConfigurationContainer configuration()

**Precondition** is\_configured() == true  
at least one Node exists in the configuration state

**Postcondition** configuration():list() is iterable

**Return** an iterable container of all configured Nodes.

**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**

ConfigurationContainer connected()

**Precondition** at least one Node is currently connected

**Postcondition** connected():list() is iterable

**Return** an iterable container of all connected Nodes.

**define\_identity\_pose**

string define\_identity\_pose()

**Summary** Capture the current orientation of all Nodes and use it as a starting reference for all subsequent frames. Affects the local orientation outputs.

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

**Effect** Define the identity orientation of all configured Nodes. The local orientation output is defined relative to the identity orientation.

**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 option],  
 [string type])

**Precondition** system.is\_filename(filename) == true  
 have\_take() == true  
 is\_configured() == true  
 system.configuration\_matches\_take() == true  
 type == nil or system.export\_type()[type] ~=  
 nil

**Effect** Export the current take to an external file format.

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

**export\_stream**

boolean, string export\_stream(string filename, [string option])

**Precondition** system.is\_filename(filename) == true  
 have\_take() == true  
 is\_configured() == true  
 system.configuration\_matches\_take() == true

**Effect** Export data streams from the current take to an external file format.

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

**get\_node**

ConfigurationNode get\_node(string key,  
*type* value,  
 [*type* compare])

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

`ConfigurationNode get_node_by_id(string id)`

**Effect** `get_node('id', id)`

#### `get_node_by_key`

`ConfigurationNode get_node_by_key(number key)`

**Effect** `get_node('key', key)`

#### `get_preview`

`PreviewContainer get_preview()`

**Precondition** `is_reading() == true`

**Postcondition** `get_preview():list()` is iterable

**Return** an iterable container of data for all configured Nodes.

#### `get_raw`

`RawContainer get_raw()`

**Precondition** `is_reading() == true`

**Postcondition** `get_raw():list()` is iterable

**Return** an iterable container of data for all configured Nodes.

#### `get_sensor`

`SensorContainer get_sensor()`

**Precondition** `is_reading() == true`

**Postcondition** `get_sensor():list()` is iterable

**Return** an iterable container of data for all configured Nodes.

#### `have_take`

`boolean have_take()`

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

**Return** true iff a take is currently loaded.

**insert**

boolean, string insert(string id, [string bus], [string parent\_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 and the new unique identifier iff a new Node is inserted into the configuration state.

**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\_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 defines 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.

**open**

boolean, string open(string filename)

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

**Effect** Load a configuration, location, Lua script, or take file.  
Detect the file type and call the appropriate file handler.

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

**reading**

ConfigurationContainer reading()

**Postcondition** reading():list() is iterable

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

**read\_user\_data**

table read\_user\_data(string id)

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

**Return** a table of number entries from the user portion of the  
device memory

**replay\_take**

boolean, string replay\_take([string filename])

**Precondition** load\_take\_source(filename) == true

**Effect** Replay a take. Rebuild and run the filtering pipeline for  
the current take. Optionally pre-load a take to replay.

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

**Return** true iff the the take data sources are successfully copied  
and a take is in progress, reading from the take data  
files.

**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 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'.

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

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

**set\_configuration**

boolean, string set\_configuration(string key,  
type value,  
[string id])

**Precondition** id == nil or 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 `ConfigurationNode` objects in the current configuration state.

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

**set\_gain**

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

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

**Effect** `set_configuration('gain', value, id)` . Set the gain parameter of one or all configured Nodes, which 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 sensor gain parameter of one or all configured Nodes, which 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 parameter of one or all configured Nodes, select the output range of the accelerometer. Any value less than 6 maps to 2 g, otherwise 6 g.

**set\_name**

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

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

**Effect** set\_configuration('name', value, id) . Set the name parameter 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])

**Effect** define\_identity\_pose()

**set\_pose\_marker**

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

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

**Precondition** `is_reading(id) == true`  
`is_taking() == false`

**set\_source**

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

**Summary** Set the source parameter of one or all configured Nodes. The source parameter defines the physical input source for a Node. For example, a Node may read from a USB device or from a file on disk.

**Precondition** `is_connected(id) == false`  
**Effect** `set_configuration('source', 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)`.

**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()

- 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.

**stop**

boolean, string stop([string id])

- Precondition** is\_reading([id]) == true  
is\_taking() == false or stop\_take() == true
- Effect** Stop reading data from one or more Nodes. Stop the current take.
- Postcondition** is\_reading([id]) == false
- Return** true iff all requested Nodes stopped streaming data.

**stop\_take**

boolean, string stop\_take()

- Precondition** is\_taking() == true
- Effect** Stop the current take. Save the take definition to a file.
- Postcondition** is\_taking() == false
- Return** true iff the current take is successfully stopped and all output files are saved.

**take**

TakeNode take()

- Precondition** have\_take() == true
- Return** a description of the currently loaded take.

**track\_take**

boolean, string track\_take(string filename)

- Precondition** system.file\_exists(filename) == true
- Effect** Modify the user section of a take. Run offline based tracking systems.
- Return** true iff the take file is successfully read, the tracking systems run, and the data save back into the take.

**write\_configuration**

boolean, string write\_configuration(string id,  
[boolean write\_factory])

**Precondition** `is_configured(id) == true`  
`is_connected(id) == false`  
**Effect** Write the configuration values for this Node to the on-board non-volatile/flash memory. Not all data sources support onboard memory writes.

#### **write\_user\_data**

`boolean write_user_data(string id, table data)`  
**Precondition** `is_configured(id) == true`  
`is_connected(id) == false`  
`#data > 0 and #data <= 32`  
**Return** true iff the the input data is successfully saved to the user portion of the device memory.

### **3.2 node.usb**

`boolean, string restore_configuration(string id)`

#### **restore\_configuration**

`boolean, string restore_configuration(string id)`  
**Precondition** `is_configured(id) == true`  
`is_connected(id) == false`  
**Effect** Restore the Node configuration stored in onboard non-volatile memory to the factory setting. The factory setting is simply a backup, and is writable by the command `write_configuration`.

### **3.3 node.system**

`boolean, string calibrate_from_take(string id,`  
`[boolean calibrate_accelerometer],`  
`[boolean calibrate_gyroscope_only])`  
`boolean, string configuration_matches_take()`  
`boolean, string console_client(string address, number port)`  
`table export_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()`

---

```
LocationContainer get_location_list()
PreferenceNode get_preference()
RactorContainer get_ractor_list()
table get_service()
SystemInfoNode get_system_info()
WifiContainer get_wifi_list()
ZeroconfContainer get_zeroconf_list()
boolean, string initialize()
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 magnetic_from_take(string id)
boolean, string open_database(string filename)
void print(type ...)
boolean quit()
boolean, string ractor(string name,
                       number height,
                       number arm,
                       number leg)
boolean republish_services()
void 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 service_exists(string name)
boolean set_wifi(string hwid, string essid, string password)
boolean shutdown(string id)
```



```

boolean sleep(number second)
boolean, string start_services(table service)
boolean test_wifi(string hwid, string essid, string password)
boolean, string unique_id(string id)
boolean, string unload_plugin([string name])

```

#### calibrate\_from\_take

```

boolean, string calibrate_from_take(string id,
    [boolean calibrate_accelerometer],
    [boolean calibrate_gyroscope_only])

```

**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\_accelerometer** 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\_gyroscope\_only** 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.

#### console\_client

```

boolean, string console_client(string address, number port)

```

**Effect** Start an interactive console in the current thread.

**export\_type**

table export\_type()

**Return** an associative array of valid export types.

**file\_canonical**

string file\_canonical(string filename)

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

**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 interface 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**

LocationContainer get\_location\_list()

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

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

**get\_preference**

PreferenceNode get\_preference()

**Return** the current shared system preferences.

**get\_ractor\_list**

RactorContainer get\_ractor\_list()

**Postcondition** get\_ractor\_list():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\_system\_info**

SystemInfoNode get\_system\_info()

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

**get\_wifi\_list**

WifiContainer get\_wifi\_list()

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

**get\_zeroconf\_list**

ZeroconfContainer 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.

**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**  $-90 < \text{latitude} < 90$   
 $-180 < \text{longitude} < 180$   
 $-1000 < \text{elevation} < 600000$

**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.

**magnetic\_from\_take**

boolean, string magnetic\_from\_take(string id)

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

is\_connected(id) == true

calibrate\_from\_take(id) == true

**Effect** Use the current take to automatically generate magnetic calibration offset values for all Nodes. Calibrate one node in the magnetic environment to generate a mapping for all other nodes.

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

**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 `republish_services()`

**Effect** Republish the Zeroconf/Bonjour services on the local network. This can be useful if the propagation of the services is taking too much time.

**Return** Returns true iff services were running and successfully republished.

### reset\_wifi

void `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.

**Return** Returns true.

### 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.

**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

**service\_exists**

boolean service\_exists(string name)

**Return** Returns true iff the named service is implemented in this version of the software.

**set\_wifi**

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

**Return** Returns true iff the requested network is saved as the preferred network. Use empty parameters to clear the preferred network.



### 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.

### test\_wifi

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

**Return** Switch to the selected Wi-Fi network immediately. Do not save the test network as the preferred network. The preferred network will load again when the system is restarted.

### 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.

### 3.4 Classes

#### ConfigurationContainer

Simple container for a list of `ConfigurationNode` objects. Use the `list` method as a Lua `iterator` to access each child node object.

Name	Type	Mutable
<code>list</code>	<code>iterator(ConfigurationNode)</code>	

#### ConfigurationNode

Name	Type	Mutable
<code>key</code>	<code>number</code>	
<code>id</code>	<code>string</code>	
<code>name</code>	<code>string</code>	<code>true</code>
<code>source</code>	<code>string</code>	<code>true</code>
<code>gselect</code>	<code>number</code>	<code>true</code>
<code>time_step</code>	<code>number</code>	<code>true</code>
<code>gain</code>	<code>number</code>	<code>true</code>
<code>gain_sensor</code>	<code>number</code>	<code>true</code>
<code>pipeline_select</code>	<code>number</code>	<code>true</code>
<code>node_version</code>	<code>number</code>	
<code>active</code>	<code>boolean</code>	
<code>bus</code>	<code>boolean</code>	
<code>parent</code>	<code>string</code>	<code>true</code>
<code>mass</code>	<code>number</code>	
<code>uuid</code>	<code>string</code>	

#### LocationNode

Simple container for geographic location data.

Name	Type	Mutable
<code>key</code>	<code>number</code>	
<code>name</code>	<code>string</code>	
<code>canonical_address</code>	<code>string</code>	
<code>latitude</code>	<code>number</code>	
<code>longitude</code>	<code>number</code>	
<code>elevation</code>	<code>number</code>	
<code>active</code>	<code>number</code>	

#### LocationContainer

Name	Type	Mutable
<code>list</code>	<code>iterator(LocationNode)</code>	

**PreferenceNode**

<b>Name</b>	<b>Type</b>	<b>Mutable</b>
address_input	string	
address_canonical	string	
valid_location	boolean	
latitude	number	
longitude	number	
elevation	number	
ractor_name	string	
height	number	
arm	number	
leg	number	
data_path	string	
log_path	string	
database_path	string	

**PreviewContainer**

<b>Name</b>	<b>Type</b>	<b>Mutable</b>
list	iterator(PreviewNode)	

**PreviewNode**

Simple container for preview data.

<b>Name</b>	<b>Type</b>	<b>Mutable</b>
id	number	
Gqw	number	
Gqx	number	
Gqy	number	
Gqz	number	
Lqw	number	
Lqx	number	
Lqy	number	
Lqz	number	
rx	number	
ry	number	
rz	number	
ax	number	
ay	number	
az	number	

**RawContainer**

<b>Name</b>	<b>Type</b>	<b>Mutable</b>
list	iterator(RawNode)	

### RawNode

Simple container for raw data.

Name	Type	Mutable
id	number	
ax	number	
ay	number	
az	number	
gx	number	
gy	number	
gz	number	
mx	number	
my	number	
mz	number	

### SensorContainer

Name	Type	Mutable
list	iterator(SensorNode)	

### SensorNode

Simple container for sensor data.

Name	Type	Mutable
id	number	
ax	number	
ay	number	
az	number	
gx	number	
gy	number	
gz	number	
mx	number	
my	number	
mz	number	

### ServiceNode

Simple container describes a service.

Name	Type	Mutable
description	string	
port	number	

### SystemInfoNode

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

---

Name	Type	Mutable
uptime	number	
remaining	number	
connected	number	
odometer	number	
battery_level	number	
battery_state	number	
serial_number	string	
uuid	string	

### RactorNode

Simple container for current ractor definition information.

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

### RactorContainer

Name	Type	Mutable
list	iterator(RactorNode)	

### TakeNode

Simple container for current take information.

Name	Type	Mutable
uuid	string	
name	string	
description	string	
start	number	
end	number	
location	string	
loaded_from	string	

### WifiContainer

Name	Type	Mutable
list	iterator(WifiNode)	

### WifiNode

Simple container for wireless access point information.

Name	Type	Mutable
hwid	string	
ssid	string	
password	string	
type	number	
signal	number	
active	number	

**ZeroconfContainer**

Name	Type	Mutable
list	iterator(ZeroconfNode)	

**ZeroconfNode**

Simple container for zeroconf data service information.

Name	Type	Mutable
name	string	
hostname	string	
address	string	
port	number	

**3.5 Lua****boolean**

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

**iterator**

Lua iterator function. See chapter 4.3.5 in the "Programming in Lua" book for more details (<http://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 (<http://www.lua.org/pil/2.1.html>).

**number**

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

**string**

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

**table**

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