Scripting Reference

Motion Version 2.6

www.motionnode.com
www.motionshadow.com

Contents

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 take_filename,
      [string option],
      [string type],
      [number mode])
boolean, string export_stream(string filename, string take_filename,
[string option], [string type])
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([string name], [string description], [string
timecode])
boolean, string stop([string id])
boolean, string stop_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 take_filename,
    [string option],
    [string type],
    [number mode])
```

Precondition	`system.is_filename(filename) == true`
	`system.file_exists(take_filename) == true`
	`type == nil or system.export_type(mode, 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. 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 type])`

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

Effect	Export data streams from the current take to an external file format. Calls `export`(filename, take_filename, option, type, 2) to do the work.
Return	`true` iff the current take is successfully exported to the requested file. Returns canonical filename of the exported file on success and error message on failure.

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 hiearchical 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([string name], [string description], [string timecode])`

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.

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 sucessfully saved to the user portion of the device memory.

3.2 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([number mode], [string type])
table export_stream_type([string 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()
TakeNode get_take()
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 set_timecode(string value)
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 trigger_state_change(number code)
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 succefully 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([number mode], [string type])`

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

export_stream_type

`table export_stream_type([string type])`

Effect `export_type(2, 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)`

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_take

`TakeNode get_take()`

Precondition `have_take() == true`

Return a description of the currently loaded take.

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 contants.

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.

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 succefully 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

set_timecode

`boolean set_timecode(string value)`

Summary Sets a system wide timecode value.

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.

trigger_state_change

`boolean trigger_state_change(number code)`

Precondition `0 < 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

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
list	`iterator`(`ConfigurationNode`)	

ConfigurationNode

Name	Type	Mutable
key	`number`	
id	`string`	
name	`string`	true
source	`string`	true
gselect	`number`	true
time_step	`number`	true
gain	`number`	true
gain_sensor	`number`	true
track_gyroscope_bias	`number`	true
pipeline_select	`number`	true
node_version	`number`	
active	`boolean`	
bus	`boolean`	
parent	`string`	true
mass	`number`	
uuid	`string`	

ExportTypeNode

Simple container for information about an file format support by either a built exporter or an active plugin.

Name	Type	Mutable
id	`string`	
description	`string`	
option	`string`	
mode	`number`	

ExportTypeContainer

Name	Type	Mutable
list	`iterator(ExportTypeNode)`	

LocationNode

Simple container for geographic location data.

Name	Type	Mutable
key	`number`	
name	`string`	
canonical_address	`string`	
latitude	`number`	
longitude	`number`	
elevation	`number`	
active	`number`	

LocationContainer

Name	Type	Mutable
list	`iterator(LocationNode)`	

PreferenceNode

Name	Type	Mutable
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`	

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`	
essid	`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`	

3.4 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).