|  |
| --- |
|  |
|  |
| **Lua Reference Manual Milestone 312 Ace** |

Release 3

Januar 2017

Table of contents

1. Audio Library 4
	1. audio.play() 4
	2. audio.stop() 4
	3. audio.pause() 4
	4. audio.resume() 4
	5. audio.status() 5
	6. audio.tts() 5
	7. audio.record() 5
	8. tbd: audio.external\_rec\_setting() 6
	9. audio.back\_play() 7
	10. audio.back\_stop() 7
2. Event Library 7
	1. event.pop() 7
	2. event.clear() 8
	3. event.reset() 8
	4. event.push() 8
	5. event.count() 9
3. Tuner Library 9
	1. tuner.status() 9
	2. tuner.open() 9
	3. tuner.play() 10
	4. tuner.close() 10
	5. tuner.seek() 10
	6. tuner.tune() 11
	7. tuner.tune\_status() 11
	8. tuner.tx\_tune() 12
4. MEMS Library 12
5. Miscellanous Libraries 13
	1. led.on(), led.off(), led.toggle() 13
	2. motor.on(), motor.off(), motor.toggle() 13
	3. power.charging() 13
	4. power.batvoltage(), power.batlevel() 14
	5. power.5vpresent() 14
	6. power.mode() 14
	7. power.otg() 14
	8. rtc.time() 15
	9. timer.read() 15
	10. timer.delay() 15
6. Lists 16
	1. Event list 16
	2. Prompt audio file list 17
7. Audio Library
	1. audio.play()

This function starts playing an audio file.

Arguments

String with name of file to play

Return values

nil if OK or error message otherwise

* 1. audio.stop()

This function stops playing / recording started by Lua. This function is called automatically when the Lua interpreter is exited, i.e., your Lua program comes to an end.

The function automatically detects if it has to stop playback or recording (or nothing).

Arguments

None

Return values

None

* 1. audio.pause()

This function pauses playback or recording.

Arguments

None

Return values

nil if OK or error message otherwise.

* 1. audio.resume()

This function resumes playing or recording of a previously paused playback or recording action.

Arguments

None

Return values

nil if OK or error message otherwise.

* 1. audio.status()

This function returns the status of the audio player / recorder.

Arguments

None

Return values

Status table if OK, nil and error message otherwise. The status table contains the elements:

field **status**: play, record, pause, stop or error

field **elapsed**: time played (in seconds, integer)

(if playing) field **remaining**: time remaining (in seconds, integer)

* 1. audio.tts()

This function talks the input string to the output (Text To Speech).

Arguments

String with text to play.

Return values

nil if OK or error message otherwise.

* 1. audio.record()

This function starts recording.

Arguments

String with file name to store recording. If this name is "M\_TEMP.MP3", Milestone automatically selects a numbered name like M\_0007.MP3.

Optional table with options:

* source = mic or tuner, default = mic.
* headphones = <boolean> true or false, default = false. This option defines if the recorded sound shall be monitored to the headphones during recording.
* sound = <string> (default = ""). This defines the name of the start sound that is played prior to recording, e.g., the click sound for normal voice records. Any predefined prompt file can be given here, like "TAB.WAV". See the prompt list at the end of this document for the whole list of available prompts.
* external = <boolean> true or false, default = false. This defines is recording from an external source is allowed, like an external microphone or line-in recording. If "external" is true AND an external source is connected, recording will take place from there. Otherwise recording takes place from the internal microphone.
* power = <boolean> true or false, default = false. This defines if 2 volts voltage is being applied to the external recording jack. This voltage is used by external microphones, but not line-in recording.
* raw = true or false, default = false. If set to true, raw audio data are recorded (16 bit stereo, 44100 Hz) instead of MP3. Pay attention not to chose a file name "M\_TEMP.MP3" together with the raw option set as true, because then a file with MP3-ending results, where in fact it is a raw data file (like WAV, just without the header). This function works only in combination with option "external = true", since the internal microphone always records in MP3.

Please note the syntax of the optional table as given in the example.

Return values

nil if OK or error message otherwise.

Example

Record from external microphone, with sound being monitored on headphones, start with "click" prompt at recording start:

audio.record("record1.raw", {source="mic", headphones="true", sound="TAB.WAV", power="true", raw="false"})

* 1. tbd: audio.external\_rec\_setting()

To do (define parameters for external recording, like power, bitrate, format, filter)

* 1. audio.back\_play()

This function starts playing a WAV background sound. To change a running background sound, this function can be called again, without prior audio.back\_stop().

Arguments

File name of background sound. The sound must have the format WAV, 16bit, 44.1 kiloHertz sampling rate, stereo. It is recommended to use low-volume sounds only, because mixing is done by adding. This could lead to clipping in case of high volume sounds. A volume level of 20% maximum level is recommended.

Return values

nil if OK or error message otherwise.

* 1. audio.back\_stop()

This function stops a playing background sound.

Arguments

None.

Return values

None.

1. Event Library

Within Milestone, many things work with events. This means, messages are being sent for keys pressed / released, or USB has been connected, and many other things. It is vital to work with events to handle real life situations correctly and in an efficient manner.

A list of event messages follows at the end of this document.

* 1. event.pop()

This function fetches the next event from the event buffer.

Arguments

None.

Return values

The according event or nil, if none available.

Example

ev = event.pop()

if ev == event.KEY\_PLAY then

 print("Play has been pressed");

end

* 1. event.clear()

This function clears non keyboard events in the event buffer.

Arguments

None.

Return values

None.

* 1. event.reset()

This function clears all pending events in the event buffer. Use this function mostly at beginning and end of a Lua programm, to avoid false reactions due to leftover events in the Lua FIFO buffer.

Arguments

None.

Return values

None.

* 1. event.push()

This function pushes an event into the event buffer, instead of getting it back.

Arguments

The according event.

Return values

None.

* 1. event.count()

This function counts how many events are waiting in the event buffer.

Arguments

None.

Return values

Number of events waiting. The total size of the event buffer is 50 events.

1. Tuner Library

The tuner library function take control of low level tuner functions, both receiving and transmitting.

* 1. tuner.status()

Give the status of the tuner.

Arguments

None.

Return values

A table with the tuner status fields:

field mode:

field play:

field txaudio:

field output\_sample\_rate:

field input\_sample\_rate:

field tune\_timeout:

* 1. tuner.open()

Power up and configure the tuner, setting its properties for the specified region and operational mode Arguments.

Arguments

Country configuration code (CFG\_USA, CFG\_JAPAN or CFG\_EUROPE).

Mode: MODE\_RX or MODE\_TX for receiving / transmitting.

Return values

None.

* 1. tuner.play()

Start playback of tuner.

Arguments

Play control: true for ON, false for OFF.

Return values

 Output: Result code 0 if OK or negative error number otherwise.

* 1. tuner.close()

Shutdown the tuner.

Arguments

None.

Return values

None.

* 1. tuner.seek()

Seek next receive station.

Arguments

SEEK\_UP or SEEK\_DOWN for searching upwards or downwards,

WRAP (boolean) = true or false, to determine if search frequency should wrap upon reaching bandwidth end (108.0 MHz or 87.5MHz) or end.

Return values

Result code 0 if OK or negative error number otherwise,

Table with tune status.

* 1. tuner.tune()

Tune to specified frequency.

Arguments

Frequency to tune in 10Khz units (from 6'400 t0 10'800). 6'400 is for Japanese band. For Europe and USA, minimum frequency is 8'750.

***Return values***

Result code 0 if OK or negative error number otherwise,

Table with tune status.

* 1. tuner.tune\_status()

Give back a table with information about the reception status.

Arguments

None.

***Return values***

A table with entries:

field valid: true or false

field afcrl:

field bltf:

field freq:

field rssi:

field snr:

field mult:

field antcap:

* 1. tuner.tx\_tune()

Transmit: Tune to specified frequency.

Arguments

Frequency to tune in 10Khz units (from 6'400 t0 10'800). 6'400 is for Japanese band. For Europe and USA, minimum frequency is 8'750.

***Return values***

Result code 0 if OK or negative error number otherwise,

Table with tune status.

1. MEMS Library

The MEMS chip (Microelectromechanical systems chip) measures the orientation of Milestone towards earth gravity. With continuous tracking of the MEMS values, changes of orientation can be monitored, leading to applications like step counter, snooze alarm clock, acceleration metering and much more.

To do: This library exists already and needs to be described. Functions are:

mems.init()

mems.active()

mems.standby()

mems.read() Read X, Y & Z raw data from local buffer. Input number of samples to read, output list with read samples or nil if error.

mems.readXYZ() ReadX, Y & Z raw axis data, return status, x, y, z

mems.readFIFO() Read FIFO X, Y & Z raw axis data, return status, x, y, z

mems.sensitivity()

mems.datasize()

mems.datarate()

mems.oversampling()

mems.hpf()

mems.fifo()

mems.fifo\_act()

mems.fifo\_deact()

mems.autosleep()

mems.orientation()

mems.ffmt()

mems.transient()

1. Miscellanous Libraries
	1. led.on(), led.off(), led.toggle()

Turn on red LED, or off, or toggle it (=switch other way around).

Arguments

None.

***Return values***

None.

* 1. motor.on(), motor.off(), motor.toggle()

Turn on vibration motor, or off, or toggle it.

Arguments

None.

***Return values***

None.

* 1. power.charging()

Give information of battery currently is charging.

Arguments

None.

***Return values***

true if charging, false if not.

* 1. power.batvoltage(), power.batlevel()

Give voltage of battery, or give level from 1 (full) to 4 (empty).

Arguments

None.

***Return values***

For power.batvoltage: Voltage in millivolts. This is 4'200 (max) down to 3'000.

For power.batlevel: 1 full, 2 charged, 3 please charge, 4 charge urgently.

* 1. power.5vpresent()

Give information if 5V supply is present on the USB connector.

Arguments

None.

***Return values***

true for present, false for not present.

* 1. power.mode()

Give information which supply feeds Milestone at this time.

Arguments

None.

***Return values***

true for battery, false for external 5 Volt source.power.5vpresent()

Give information if 5V supply is present on the USB connector.

* 1. power.otg()

Give information if a USB on-the-go device is connected. These are devices that have pin 5 of the USB connector shortened to ground, indicating that they need 5V to awake. Examples are USB sticks, Bones Add-On, Bones Woodscan.

Arguments

None.

***Return values***

true for OTG connected, false for not connected.

* 1. rtc.time()

Give back real time clock

***Arguments***

None.

***Return values***

Number of seconds since 1st January 1970, noon.

* 1. timer.read()

Give back content of a micro second timer. This is a hardware timer that continuously counts in micro seconds, with 32 bit width (making it several days for an overroll). This counter starts at zero upon turning on Milestone. Usually, this timer is used to calculate the time difference between two consecutive function calls. Usually, the absolute value of this timer is not relevant.

***Arguments***

None.

***Return values***

Value of timer that counts in micro seconds.

* 1. timer.delay()

A delay function in microseconds.

***Arguments***

Number of microseconds to sleep.

***Return values***

None.

1. Lists
	1. Event list

KEY\_NONE

KEY\_PLAY

KEY\_MODE

KEY\_ERASE

KEY\_REC

KEY\_REC\_LONG

KEY\_BLOCK

KEY\_ERASE\_REC

KEY\_FF

KEY\_FF\_PLAY)

KEY\_INC\_VOL

KEY\_ERASE\_FF

KEY\_REC\_FF

KEY\_REW

KEY\_REW\_PLAY

KEY\_DEC\_VOL

KEY\_ERASE\_REW

KEY\_REC\_REW

KEY\_ERASE\_FF\_REW

KEY\_PANIC)

KEY\_SEL

KEY\_SEL\_PLAY

KEY\_SEL\_MODE

KEY\_ERASE\_SEL)

KEY\_SEL\_REC

KEY\_SEL\_FF

KEY\_INC\_VOL\_SEL

KEY\_SEL\_REW

KEY\_DEC\_VOL\_SEL

KEY\_SEL\_FF\_REW)

FILE\_END

USB\_ON

USB\_OFF

RIP\_DONE

POWER\_UNPLUGGED

HID\_ON

HID\_OFF

ADD\_ON\_ON

ADD\_ON\_OFF

HOST\_MODE\_ON

HOST\_MODE\_OFF

MST\_ON

MST\_OFF

USB\_AUDIO\_ON

USB\_AUDIO\_OFF

* 1. Prompt audio file list

"1K.WAV"

"2K1K.WAV"

"1K2K.WAV"

"0K5.WAV"

"CONTINUE.WAV"

"PAUSE.WAV"

"TAB.WAV"

"poweroff.wav"

"WAKE.WAV"

"DELETE.WAV"

"4X0K5.WAV"

"NEXT.WAV"