lsnes Lua functions reference

December 14, 2013

1 Table of contents

Contents

2 Special tokens

These tokens are special, and are expanded while the script is being loaded

2.1 @@LUA_SCRIPT_FILENAME@@

Expanded to string token containing path and filename of this Lua script. Handy for referencing other lua scripts or resources that are relative to this Lua script.

In practicular, this is suitable to be passed as base argument of various functions like loadfile, dofile, resolve_filename, gui.bitmap_load, gui.bitmap_load_png and gui.bitmap_load_pal.

3 Global

3.1 print: Print values to console

- Syntax: none print(value... values)

Prints specified values to console. Can print any Lua type at least enough to identify the type and instance.

3.2 tostringx: Format a value to string

- Syntax: string tostringx(value val)

Formats value <val> like print would, and returns the result as a string.

3.3 exec: Execute lsnes commands

- Syntax: none exec(string cmd)

Execute lsnes command <cmd>.

3.4 utime: Get current time

- Syntax: (number,number) utime()

Returns two numbers. First is time since some epoch in seconds, the second is microseconds mod 10^6 since that epoch.

3.5 emulator_ready: Check if emulator has been fully initialized

- Syntax: boolean emulator_ready()

Returns true if emulator has finished booting, false if not (on_startup() will be issued later).

3.6 set_idle_timeout: Run function after timeout when emulator is idle

- Syntax: none set_idle_timeout(number timeout)

Set number of microseconds to block idle for. After this timeout has expired, on_idle() will be called once.

3.7 set_timer_timeout: Run function after timeout.

- Syntax: none set_timer_timeout(number timeout)

Set number of microseconds to block timer for. After this timeout has expired, on_timer() will be called once.

3.8 bus_address: Look up address in system bus.

- Syntax: none bus_address(number bus_addr)

Returns virtual address corresponding to specified address on system bus.

3.9 loopwrapper: Convert loop into callable function

- Syntax: function loopwrapper(function fun, ...)

Calls function <fun> with function and specified arguments. The function passed suspends execution until the function returned is called. Handy for linear flow control among multiple invocations of a hook. Example code:

```
on_paint = loopwrapper(function(wait)
    while true do
        gui.text(0, 0, “Test!”);
        wait();
    end
end);
```

3.10 list_bindings: List keybindings

- Syntax: table list_bindings([string cmd])

Get table of all keybindings, indexed by keyspec (modifiers|mask/key). If <cmd> is specified, the table is limited to that command. Also searches for controller keys.

3.11 get_alias: Get expansion of alias

- Syntax: string get_alias(string aname)

Get expansion of given alias <aname>.

3.12 set_alias: Set expansion of alias

- Syntax: none set_alias(string aname, string value)

Set expansion of given alias.

3.13 create_ibind: Create invese binding

- Syntax: INVERSEBIND create_ibind(string name, string cmd)

Return object representing inverse binding with specified name <name> and specified command <cmd>.

- Note: To create press/release commands, use aliases +foo and -foo .
- Note: Keep the returned object around.

3.14 create_command: Create a command

- Syntax: COMMANDBIND create_commmand(string name, function a)
- Syntax: COMMANDBIND create_commmand(string name, function a, function b)

Return object representing a command (pair).

- If only one function is specied, the command is level-sensitive, <a> is callback.
- If <b> is function, the function is edge-sensitive, <a> is positive edge callback and <b> is negative edge callback.
- All callbacks get single argument: The parameters passed.
- Keep the returned object around.

3.15 loadfile: Load Lua script

- Syntax: function loadfile(string filename[, string base])

Load lua script from <filename>, resolved relative to <base> (if empty, current directory).

3.16 dofile: Execute Lua script

- Syntax: function dofile(string filename[, string base])

Execute lua script from <filename>, resolved relative to <base> (if empty, current directory) and return all return values.

3.17 open_file: Open a stream

- Syntax: FILEREADER open_file(string filename[, string base])

Open file <filename>, resolved relative to <base> (if empty, current directory) and return a handle.

3.18 FILEREADER(): Read line/bytes from stream

- Syntax: string/nil FILEREADER()
- Syntax: string/nil FILEREADER(number bytes)

Reads next line or <bytes> bytes from specified file handle. On EOF, nil is returned.

- Note: The line-oriented variant reads in text mode, so CR at end of line is stripped.

3.19 FILEREADER:lines: Iterator to read all lines

- Syntax: for line in <foo>:lines() do ... end

Iterator for reading all lines of <foo> in a loop.

3.20 resolve_filename: Resolve name of file relative to another

- Syntax: string resolve_file(string filename, string base)

Resolve name of file <filename> relative to <base> and return the result.

4 Table bit:

Bitwise logical functions and related.

4.1 bit.none/bit.bnot: Bitwise none or NOT function

- Syntax: number bit.none(number...)
- Syntax: number bit.bnot(number...)

48-bit bitwise NOT / NONE function (set bits that are set in none of the arguments).

4.2 bit.any/bit.bor: Bitwise any or OR function

- Syntax: number bit.any(number...)
- Syntax: number bit.bor(number...)

48-bit bitwise OR / ANY function (set bits that are set in any of the arguments).

4.3 bit.all/bit.band: Bitwise all or AND function

- Syntax: number bit.all(number...)
- Syntax: number bit.band(number...)

48-bit bitwise AND / ALL function (set bits that are set in all of the arguments).

4.4 bit.parity/bit.bxor: Bitwise parity or XOR function

- Syntax: number bit.parity(number...)
- Syntax: number bit.bxor(number...)

48-bit bitwise XOR / PARITY function (set bits that are set in odd number of the arguments).

4.5 bit.lrotate: Rotate a number left

- Syntax: number bit.lrotate(number base[, number amount[, number bits]])

Rotate <bits>-bit (max 48, default 48) number <base> left by <amount> (default 1) places.

4.6 bit.rrotate: Rotate a number right

- Syntax: number bit.rrotate(number base[, number amount[, number bits]])

Rotate <bits>-bit (max 48, default 48) number <base> right by <amount> (default 1) places.

4.7 bit.lshift: Shift a number left

- Syntax: number bit.lshift(number base[, number amount[, number bits]])

Shift <bits>-bit (max 48, default 48) number <base> left by <amount> (default 1) places. The new bits are filled with zeroes.

4.8 bit.lrshift: Shift a number right (logical)

- Syntax: number bit.lrshift(number base[, number amount[, number bits]])

Shift <bits>-bit (max 48, default 48) number <base> logically right by <amount> (default 1) places. The new bits are filled with zeroes.

4.9 bit.arshift: Shift a number right (arithmetic)

- Syntax: number bit.arshift(number base[, number amount[, number bits]])

Shift <bits>-bit (max 48, default 48) number <base> logically right by <amount> (default 1) places. The new bits are shifted in with copy of the high bit.

4.10 bit.extract: Extract/shuffle bits from number

- Syntax: number bit.extract(number base[, number bit0[, number bit1,...]])

Returns number that has bit0-th bit as bit 0, bit1-th bit as 1 and so on.

- Note: Bit numbers up to 51 should work reliably (then things start falling apart due to double precision issues).
- Note: There are two special bit positions, true and false, standing for always set bit and always clear bit.

4.11 bit.value: Construct number with specified bits set

- Syntax: number bit.value([number bit1[, number bit2,...]])

Returns bitwise OR of 1 left shifted by <bit1> places, 1 left shifted by <bit2> places and so on. As special value, nil argument is no-op.

4.12 bit.test_any: Test if any bit is set

- Syntax: boolean bit.test_any(number a, number b)

Returns true if bitwise and of <a> and <b> is nonzero, otherwise false.

4.13 bit.test_all: Test if all bits are set

- Syntax: boolean bit.test_all(number a, number b)

Returns true if bitwise and of <a> and <b> equals <b>, otherwise false.

4.14 bit.popcount: Population count

- Syntax: number bit.popcount(number a)

Returns number of set bits in <a>.

4.15 bit.clshift: Chained left shift

- Syntax: (number, number) bit.clshift(number a, number b, [number amount,[number bits]])

Does chained left shift on <a>, <b> by <amount> positions (default 1), assuming numbers to be of specified number of bits <bits> (default 48).

4.16 bit.crshift: Chained right shift

- Syntax: (number, number) bit.crshift(number a, number b, [number amount,[number bits]])

Does chained right shift on <a>, <b> by <amount> positions (default 1), assuming numbers to be of specified number of bits <bits> (default 48).

4.17 bit.flagdecode: Decode bitfield into flags

- Syntax: string bit.flagdecode(number a, number bits, [string on, [string off]])

Return string of length bits where ith character is ith character of on if bit i is on, otherwise ith character of off. Out of range reads give last character.

- Note: <on> defaults to '*' if empty.
- Note: <off> defaults to '-' if empty.

4.18 bit.rflagdecode: Decode bitfield into flags

- Syntax: string bit.rflagdecode(number a, number bits, [string on, [string off]])

Like bit.flagdecode, but outputs the string in the opposite order (most significant bit first).

5 Table gui:

- Functions that draw to screen can only only be called in on_paint and on_video callbacks or if non-default render queue has been set.
- Colors are 32-bit. Bits 0-7 are the blue component, bits 8-15 are the green component, bits 16-23 are the red component, bits 24-31 are alpha component (0 is fully opaque, 255 is almost transparent). -1 is the fully transparent color.
- Alpha values greater than 127 do work properly.
- Origin of coordinates is at top left corner of game display area. Left and top gaps correspond to negative coordinates.

5.1 gui.resolution: Get current resolution

- Syntax: (number, number) gui.resolution()

Returns 2-tuple (hresolution, vresolution).

5.2 gui.left_gap/gui.right_gap/gui.top_gap/gui.bottom_gap: Set edge gaps

- Syntax: number gui.left_gap(number gap)
- Syntax: number gui.right_gap(number gap)
- Syntax: number gui.top_gap(number gap)
- Syntax: number gui.bottom_gap(number gap)

Set the specified edge gap to specified value <gap> (max gap is 8191). If successful, old gap is returned.

5.3 gui.delta_left_gap/gui.delta_right_gap/gui.delta_top_gap/gui.delta_bottom_gap: Adjust edge gaps

- Syntax: number gui.delta_left_gap(number dgap)
- Syntax: number gui.delta_right_gap(number dgap)
- Syntax: number gui.delta_top_gap(number dgap)
- Syntax: number gui.delta_bottom_gap(number dgap)

Increase the specified edge gap by specified value <dgap> (max gap is 8191) and return the old gap (returns nothing on error).

5.4 gui.text/gui.textH/gui.textV,gui.textHV: Draw text

- Syntax: none gui.text(number x, number y, string text[, number fgc[, number bgc]])
- Syntax: none gui.textH(number x, number y, string text[, number fgc[, number bgc]])
- Syntax: none gui.textV(number x, number y, string text[, number fgc[, number bgc]])
- Syntax: none gui.textHV(number x, number y, string text[, number fgc[, number bgc]])

Draw specified text on the GUI (each character cell is 8 or 16 wide and 16 high). Parameters:

- x: X-coordinate to start the drawing from (and x-coordinate at begining of the lines).
- y: Y-coordinate to start the drawing from.
- text: The text to draw.
- fgc: Text color (default is 0xFFFFFF (white))
- bgc: Background color (default is -1 (transparent))

Note: The H variants draw at double width and V variants draw at double height.

5.5 gui.rectangle: Draw a rectangle

- Syntax: none gui.rectangle(number x, number y, number width, number height[, number thickness[, number outline[, number fill]]])

Draw rectangle on the GUI. Parameters:

- x: X-coordinate of left edge.
- y: Y-coordinate of upper edge.
- width: Width of rectangle.
- height: Height of rectangle.
- thickness: Thickness of outline (default is 1).
- outline: Color of outline (default is 0xFFFFFF (white))
- fill: Color of fill (default is -1 (transparent))

5.6 gui.box: Draw a 3D-effect box

- Syntax: none gui.box(number x, number y, number width, number height[, number thickness[, number outline1[,number outline2[, number fill]]]])

Draw rectangle with 3D effect on the GUI. Parameters:

- x: X-coordinate of left edge.
- y: Y-coordinate of upper edge.
- width: Width of rectangle.
- height: Height of rectangle.
- thickness: Thickness of outline (default is 1).
- outline1: First color of outline (default is 0xFFFFFF (white))
- outline2: First color of outline (default is 0x808080 (dark gray))
- fill: Color of fill (default is 0xC0C0C0 (light grayy))

5.7 gui.pixel: Draw a single pixel

- Syntax: none gui.pixel(number x, number y[, number color])

Draw one pixel on the GUI. Parameters:

- x: X-coordinate of the pixel
- y: Y-coordinate of the pixel
- color: Color of the pixel (default is 0xFFFFFF (white))

5.8 gui.crosshair: Draw a crosshair

- Syntax: none gui.crosshair(number x, number y[, number length[, number color]])

Draw a crosshair. Parameters:

- x: X-coordinate of the crosshair
- y: Y-coordinate of the crosshair
- length: Length of the crosshair lines (default 10).
- color: Color of the crosshair (default is 0xFFFFFF (white))

5.9 gui.line: Draw a line

- Syntax: none gui.line(number x1, number y1, number x2, number y2[, number color])

Draw a thin line. Parameters:

- x1: X-coordinate of one end.
- y1: Y-coordinate of one end.
- x2: X-coordinate of the other end.
- y2: Y-coordinate of the other end.
- color: Color of the line (default is 0xFFFFFF (white)).

5.10 gui.circle: Draw a (filled) circle

- Syntax: none gui.circle(number x, number y, number r[, number thick[, number border[, number fil]]])

Draw a circle. Parameters.

- x: X-coordinate of the center
- y: Y-coordinate of the center
- r: The radius of the circle
- thick: Border thickness
- border: Border color (default is 0xFFFFFF (white))
- fill: Fill color (default is -1 (transparent)).

5.11 gui.bitmap_draw: Draw a bitmap

- Syntax: none gui.bitmap_draw(number x, number y, BITMAP bitmap, PALETTE palette)
- Syntax: none gui.bitmap_draw(number x, number y, DBITMAP bitmap)

Draw a bitmap <bitmap> on screen with specified palette <palette> (if bitmap is paletted) . Parameters:

- x: X-coordinate of left edge.
- y: Y-coordinate of top edge.
- bitmap: The bitmap to draw
- palette: The palette to draw the bitmap using.

5.12 gui.palette_new: Create a new palette

- Syntax: PALETTE gui.palette_new()

Returns a new palette (initially all transparent).

5.13 gui.bitmap_new: Create a new bitmap

- Syntax: BITMAP/DBITMAP gui.bitmap_new(number w, number h, boolean direct[, bool icolor])

Returns a new bitmap/dbitmap.
Parameters:

- w: The width of new bitmap
- h: The height of new bitmap
- direct: If true, the returned bitmap is dbitmap, otherwise bitmap.
- icolor: Initital fill color (defaults to 0 on BITMAP, -1 on DBITMAP)

5.14 gui.bitmap_load/gui.bitmap_load_str: Load a bitmap from file or string

- Syntax: DBITMAP/(BITMAP, PALETTE) gui.bitmap_load(string file[, string base])
- Syntax: DBITMAP/(BITMAP, PALETTE) gui.bitmap_load_str(string content)

Reads file <file> (resolved relative to <base>) or string <content> and returns loaded bitmap/dbitmap (if bitmap, the second return value is palette for bitmap).

5.15 gui.bitmap_load_png/gui.bitmap_load_png_str: Load a bitmap from PNG

- Syntax: DBITMAP/(BITMAP, PALETTE) gui.bitmap_load_png(string file[, string base])
- Syntax: DBITMAP/(BITMAP, PALETTE) gui.bitmap_load_png_str(string content)

Load a bitmap from PNG file <file> (resolved relative to <base>) or BASE64 encoded content <content>.

- If the PNG is of color type 3 (PALETTE), returns two value. First is BITMAP containing the image data from the PNG and second is PALETTE containg the palette data from the PNG.
- For color types 0 (GRAY), 2 (RGB), 4 (GRAY_ALPHA) and 6 (RGBA), returns one DBITMAP containg the image data loaded from the PNG.

5.16 gui.bitmap_load_pal/gui.bitmap_load_pal_str: Load a palette

- Syntax: PALETTE gui.bitmap_load_pal(string file[, string base])
- Syntax: PALETTE gui.bitmap_load_pal_str(string content)

Load a palette from file <file>(resolved relative to <base>) or string <content>.

- The kinds of lines supported:
  - Blank or just whitespace: Ignored
  - First non-whitespace is '#': Ignored
  - <r> <g> <b>: Fully opaque color with specified RGB values (0-255)
  - <r> <g> <b> <a>: Color with specified RGB values (0-255) and specified alpha (0-256, 0 being fully transparent and 256 fully opaque).
  - transparent: Fully transparent color

5.17 gui.palette_set: Set palette entry

- Syntax: none gui.palette_set(PALETTE palette, number index, number color)

Sets color in palette. Parameters:

- palette: The palette to manipulate
- index: Index of color (0-65535).
- color: The color value.

5.18 gui.bitmap_pset: Set pixel in bitmap

- Syntax: none gui.bitmap_pset(BITMAP/DBITMAP bitmap, number x, number y, number color)

Sets specified pixel in bitmap. Parameters:

- bitmap: The bitmap to manipulate
- x: The x-coordinate of the pixel.
- y: The y-coordinate of the pixel.
- color: If bitmap is a bitmap, color index (0-65535). Otherwise color value.

5.19 gui.bitmap_pget: Get pixel in bitmap

- Syntax: number gui.bitmap_pget(BITMAP/DBITMAP bitmap, number x, number y)

Gets specified pixel in bitmap. Parameters:

- bitmap: The bitmap to query
- x: The x-coordinate of the pixel.
- y: The y-coordinate of the pixel.

The bitmap color (color index if paletted, otherwise color value).

5.20 gui.bitmap_size: Get size of bitmap

- Syntax: (number, number) gui.bitmap_size(BITMAP/DBITMAP bitmap)

Get size of bitmap <bitmap>. The first return is the width, the second is the height.

- Note: Can be used anywhere.

5.21 gui.bitmap_blit: Blit a bitmap into another

- Syntax: none gui.bitmap_blit(BITMAP dest, number dx, number dy, BITMAP src, number sx, number sy, number w, number h[, number ck])
- Syntax: none gui.bitmap_blit(DBITMAP dest, number dx, number dy, DBITMAP src, number sx, number sy, number w, number h[, number ck])
- Syntax: none gui.bitmap_blit(DBITMAP dest, number dx, number dy, BITMAP src, PALETTE pal, number sx, number sy, number w, number h[, number ck])

Blit a part of bitmap to another. Parameters:

- dest: Destination to blit to.
- dx: left edge of target
- dy: Top edge of target
- src: The source to blit from.
- pal: The palette to use in blit.
- sx: left edge of source
- sy: Top edge of source
- w: Width of region
- h: Height of region.
- ck: Color key. Pixels of this color are not blitted.
  - If soruce bitmap is bitmap, this is color index of colorkey. Values outside range 0-65535 cause no key to be used as colorkey.
  - If source bitmap is dbitmap, this is the color value of colorkey.
  - May be absent or nil for no colorkey blit.

5.22 gui.repaint: Arrange a repaint

- Syntax: none gui.repaint()

Request on_repaint() to happen as soon as possible.

5.23 gui.synchronous_repaint: Paint screen now

- Syntax: none gui.synchronous_repaint(RENDERQUEUE queue)

Paints specified render queue on screen right there and then.

5.24 gui.subframe_update: Enable/Disable subframe updates

- Syntax: none gui.subframe_update(boolean on)

Request subframe updates (calling on_paint() on subframes) to happen (<on>=true) or not happen (<on>=false).

5.25 gui.screenshot: Write a screenshot

- Syntax: none gui.screenshot(string filename)

Write PNG screenshot of the current frame (no drawings) to specified file <filename>.

5.26 gui.color: Compose a color.

- Syntax: number gui.color(number r, number g, number b[, number a])

Returns color (in notation Lua scripts use) corresponding to color (<r>,<g>,<b>), each component in scale 0-255. If <a> is specified, that is alpha (0 is fully transparent, 256(sic) is fully opaque). The default alpha is 256.

5.27 gui.status: Set status variable

- Syntax: none gui.status(string name, string value)

Set status field "L[<name>]" to <value> in status area.

5.28 gui.rainbow: Rainbow color calculation

- Syntax: number gui.rainbow(number step, number steps[, number color])

Perform hue rotation of color <color> (default bright red), by <step> steps. The number of steps per full rotation is given by absolute value of <steps>.

If <steps> is negative, the rotation will be counterclockwise.

5.29 gui.renderq_new: Create a render queue

- Syntax: RENDERQUEUE gui.renderq_new(number width, number height)

Create render queue with specified reported size and return it.

5.30 gui.renderq_clear: Clear a render queue

- Syntax: none gui.renderq_clear(RENDERQUEUE queue)

Clear specified render queue.

5.31 gui.renderq_set: Change active render queue

- Syntax: none gui.renderq_set(RENDERQUEUE queue)

Switch to specified render queue. Use nil as queue to switch to default queue.

- Note: When switched to another queue, all drawing functions work and draw there, even outside on_video/on_paint.

5.32 gui.renderq_run: Run render queue

- Syntax: none gui.renderq_run(RENDERQUEUE queue)

Run specified render queue, copying the objects to current render queue.

- Warning: Don't try to run the current render queue.

5.33 gui.loadfont: Load a font file

- Syntax: CUSTOMFONT gui.loadfont([string filename])

Loads font from specified file (CUSTOMFONT object). If filename is not given, loads the system default font.

5.34 CUSTOMFONT(): Render text to screen

- Syntax: none CUSTOMFONT(number x, number y, string text[, number fgc[, number bgc[, number hlc]]])

Draw string with custom font to screen. The parameters are the same as in gui.text, except <hlc> is the halo color (default is no halo).

5.35 gui.adjust_transparency: Adjust transparency of DBITMAP or PALETTE

- Syntax: none gui.adjust_transparency(DBITMAP obj, number adj)
- Syntax: none gui.adjust_transparency(PALETTE obj, number adj)

Multiply alpha channel of <obj> by <adj>/256. Useful for making "ghosts" out of solid bitmaps.

5.36 gui.kill_frame: Kill video frame and associated sound

- Syntax: none gui.kill_frame()

Kills the currently dumped video frame + the associated sound. Only valid in on_video callback.

5.37 gui.arrow: Draw an arrow

- Syntax: none gui.arrow(number x, number y, number length, number hwidth, number direction[, bool fill[, number color[, number twidth[, number hthick]]]])

Draws an arrow using color <color>. The tip of arrow is at (<x>, <y>). Other parameters:

1. <length>: The length of arrow tail.
2. <hwidth>: The width of arrow head. Should be odd.
3. <direction>: Direction of arrow. 0 is to right, +1 rotates 45 degrees counterclockwise.
4. <fill>: If true, fill the arrow head. Default false.
5. <twidth>: Tail width. Should be odd. Default 1.
6. <hthick>: Head thickness (only used if <fill> is false). Default is <twidth>.

5.38 gui.tilemap: Create a tilemap

- Syntax: TILEMAP gui.tilemap(number w, number h, number bw, number bh)

Create a new tilemap of size <w>*<h>, with each cell being <bw>*<bh>.

5.39 TILEMAP:getsize: Query tilemap size

- Syntax: number, number TILEMAP:getsize()

Return size of tilemap (width first).

5.40 TILEMAP:getcsize: Query tilemap cell size

- Syntax: number, number TILEMAP:getcsize()

Return size of tilemap cell (width first).

5.41 TILEMAP:get: Query tilemap cell

- Syntax: none TILEMAP:get(number x, number y)
- Syntax: dbitmap TILEMAP:get(number x, number y)
- Syntax: bitmap,palette TILEMAP:get(number x, number y)

Return contents of cell at <x>,<y>.

5.42 TILEMAP:set: Set tilemap cell

- Syntax: none TILEMAP:set(number x, number y)
- Syntax: none TILEMAP:set(number x, number y, dbitmap b)
- Syntax: none TILEMAP:set(number x, number y, bitmap b, palette p)

Set contents of cell at <x>,<y>. If no bitmap/dbitmap is given, cell is cleared. Otherwise specified (d)bitmap is used (with specified palette if bitmap).

5.43 TILEMAP:scroll: Scroll tilemap

- Syntax: none TILEMAP:scroll(number ox, number oy)
- Syntax: none TILEMAP:scroll(number ox, number oy, number x, number y, number w, number h)
- Syntax: none TILEMAP:scroll(number ox, number oy, number x, number y, number w, number h, boolean circx, boolean circy)

Scrolls the tilemap tiles by <ox>,<oy>. If <x>,<y>,<w>,<h> is specified, the scrolling is limited to <w>*<h> window starting at <x>,<y> (in tiles).

If <circx> is true, the window is circular in horizontal direction. Similarly with <circy> and vertical direciton.

5.44 TILEMAP:draw: Draw tilemap

- Syntax: none TILEMAP:draw(number x, number y)
- Syntax: none TILEMAP:draw(number x, number y, number x0, number y0)
- Syntax: none TILEMAP:draw(number x, number y, number x0, number y0, number w, number h)

Draw tilemap at <x>,<y>. If <x0>,<y0> is given, that is tilemap coordinate (in pixels) of upper left edge. If <w>,<h> is given, that is the size of window to draw (in pixels)

5.45 gui.bitmap_save_png: Save a bitmap to PNG

- Syntax: none gui.bitmap_save_png(string filename[, string base], BITMAP bmp, PALETTE pal)
- Syntax: none gui.bitmap_save_png(string filename[, string base], DBITMAP bmp)
- Syntax: string gui.bitmap_save_png(BITMAP bmp, PALETTE pal)
- Syntax: string gui.bitmap_save_png(DBITMAP bmp)

Save specified bitmap <bmp>, with palette <pal> (only if paletted) into PNG file <filename> (relative to <base>) or return BASE64 encoding as return value.

5.46 gui.bitmap_hash: Hash a bitmap

- Syntax: string gui.bitmap_hash(BITMAP bmp)
- Syntax: string gui.bitmap_hash(DBITMAP bmp)

Hashes bitmap <bmp> and returns 64-hex digit crypto-strong hash. Identical bitmaps result in indentical hashes (but color order in indexed bitmaps is significant).

5.47 gui.palette_hash: Hash a palette

- Syntax: string gui.palette_hash(PALETTE pal)

Hashes palette <pal> and retruns 64-hex digit crypto-strong hash. Identical palettes result in identical hashes (fully transparent colors at end of palette don't affect the hash).

6 table input

Input handling. Functions manipulating input are only available in on_input callback.

6.1 input.get: Read controller button/axis (deprecated)

- Syntax: number input.get(number controller, number index)

Read the specified index <index> (zero-based) from specified controller <controller> (zero-based).

6.2 input.set: Write controller button/axis (deprecated)

- Syntax: none input.set(number controller, number index, number value)

Write the specified index <index> (zero-based) from specified controller <controller> (zero-based), storing value <value>.

6.3 input.get2: Read controller button/axis

- Syntax: number input.get2(number port, number controller, number index)

Read the specified input tuple. Port 0 is system port.

6.4 input.set2: Write controller button/axis

- Syntax: input.set2(number port, number controller, number index, number value)

Write the specified input tuple. Port 0 is system port.

6.5 input.lcid_to_pcid2: Look up logical controller

- Syntax: (number, number) input.lcid_to_pcid2(number lcid)

Look up physical pcid pair (port, controller) corresponding to specified logical controller (1-based). Returns nothing if controller does not exist.

6.6 input.port_type: Look up port type

- Syntax: string input.port_type(number port)

Return type of specified port.

6.7 input.controller_info: Get information about controller

- Syntax: table input.controller_info(number port, number controller)

Get controller info for specified controller. If controller does not exist, returns nil. Otherwise returns a table with following fields:

- type (string): Type of the controller.
- class (string): Class of the controller.
- classnum (number): Number of the controller within its class (1-based)
- lcid (number): Logical controller number of the controller.
- button_count (number): Number of buttons on controller
- buttons (array): Array of following info about each button:
  - type (string): Type of button. Currently one of “null”, “button”, “axis”, “raxis”.
  - name (string): Name of button.
  - symbol (string): Symbol of button. Only present for type “button”.
  - hidden (boolean): True if hidden button.

6.8 input.veto_button: Veto a button press

- Syntax: none input.veto_button()

Signals that the button event should be vetoed. Only valid in on_button callback.

6.9 input.geta: Get all buttons for controller (deprecated)

- Syntax: (number, number...) input.geta(number controller)

Get input state for entiere controller. Returns n return values.

- 1st return value: Bitmask: bit i is set if i:th index is nonzero
- 2nd- return value: value of i:th index.

6.10 input.seta: Set all buttons for controller (deprecated)

- Syntax: none input.seta(number controller, number bitmask, number args...)

Set state for entiere controller. args is up to N values for indices (overriding values in bitmask if specified).

6.11 input.controllertype: Get controller type (deprecated)

- syntax: string input.controllertype(number controller)

Get the type of controller as string.

6.12 input.reset: Execute (delayed) reset

- Syntax: none input.reset([number cycles])

Execute reset. If <cycles> is greater than zero, do delayed reset. 0 (or no value) causes immediate reset.

- Note: Only available with subframe flag false.

6.13 input.raw: Return raw input data

- Syntax: table input.raw()

Returns table of tables of all available keys and axes. The first table is indexed by key name (platform-dependent!), and the inner table has the following fields:

- value: Last reported value for control
  - For keys: 1 for pressed, 0 for released.
  - For axes: -32767...32767.
  - For presure-sensitive buttons: 0...32767.
  - For hats: Bitmask: 1=>Up, 2=>Right, 4=>Down, 8=>Left.
  - For mouse: Coordinates relative to game area.
- ktype: Type of key (disabled, key, mouse, axis, hat, pressure).

6.14 input.keyhook: Hook a key

- Syntax: none input.keyhook(string key, boolean state)

Requests that keyhook events to be sent for key <key> (<state>=true) or not sent (<state>=false).

6.15 input.joyget: Get controls for controller

- Syntax: table input.joyget(number logical)

Returns table for current controls for specified logical controller <logical>. The names of fields vary by controller type.

- The buttons have the same name as those are referred to in other contexts in the emulator
- The analog axes are usually "xaxis" and "yaxis".
- Each field is numeric or boolean depending on axis/button.

6.16 input.joyset: Set controls for controller

- Syntax: none input.joyset(number controller, table controls)

Set the the state of specified controller to values specified in specified table.

- Each field can be boolean or number.
- Also, buttons allow strings, which cause value to be inverted.

6.17 input.lcid_to_pcid: Look up logical controller (deprecated)

- Syntax: (number, number, number) input.lcid_to_pcid(number lcid)

Returns the legacy pcid for controller (or false if there isn't one), followed by pcid pair. Returns nothing if controller does not exist.

7 Table keyboard

Various keybinding-related functions

7.1 keyboard.bind: Bind a key

- Syntax: none keyboard.bind(string mod, string mask, string key, string cmd)

Bind specified key with specified modifers to specified command.

7.2 keyboard.unbind: Unbind a key

- Syntax: none keyboard.unbind(string mod, string mask, string key)

Unbind specified key with specified modifers.

7.3 keyboard.alias: Set alias expansion

- Syntax: none keyboard.alias(string alias, string expansion)

Set expansion of given command.

8 Table subtitle

Subtitle handling

8.1 subtitle.byindex: Look up start and length of subtitle by index

- Syntax: (number, number) subtitle.byindex(number i)

Read the frame and length of ith subtitle. Returns nothing if not present.

8.2 subtitle.set: Write a subtitle

- Syntax: none subtitle.set(number f, number l, string txt)

Set the text of subtitle.

8.3 subtitle.get: Read a subtitle

- Syntax: string subtitle.get(number f, number l)

Get the text of subtitle.

8.4 subtitle.delete: Delete a subtitle

- Syntax: nonesubtitle.delete(number f, number l)

Delete specified subtitle.

9 Table hostmemory

Host memory handling (extra memory saved to savestates). Host memory starts empty.

- Reads out of range return false.
- Writes out of range extend the memory.

9.1 hostmemory.read: Read byte from host memory

- Syntax: number hostmemory.read(number address)

Reads byte from hostmemory slot address <address>.

9.2 hostmemory.write: Write byte to host memory

- Syntax: none hostmemory.write(number address, number value)

Writes hostmemory slot with value <value> 0-255.

9.3 hostmemory.read{,s}{byte,{,h,d,q}word}: Read from host memory

- Syntax: number hostmemory.readbyte(number address)
- Syntax: number hostmemory.readsbyte(number address)
- Syntax: number hostmemory.readword(number address)
- Syntax: number hostmemory.readsword(number address)
- Syntax: number hostmemory.readhword(number address)
- Syntax: number hostmemory.readshword(number address)
- Syntax: number hostmemory.readdword(number address)
- Syntax: number hostmemory.readsdword(number address)
- Syntax: number hostmemory.readqword(number address)
- Syntax: number hostmemory.readsqword(number address)

Read elements (big-endian) from given address <address>.

- byte is 1 element
- word is 2 elements
- hword is 3 elements
- dword is 4 elements
- qword is 8 elements.
- The 's' variants do signed read.

9.4 hostmemory.read{float,double}: Read from host memory

- syntax: number hostmemory.readfloat(number address)
- Syntax: number hostmemory.readdouble(number address)

Read elements (big-endian) floating-pont from given address <address>.

9.5 hostmemory.write{,s}{byte,{,h,d,q}word}: Write to host memory

- Syntax: number hostmemory.writebyte(number address, number value)
- Syntax: number hostmemory.writesbyte(number address, number value)
- Syntax: number hostmemory.writeword(number address, number value)
- Syntax: number hostmemory.writesword(number address, number value)
- Syntax: number hostmemory.writehword(number address, number value)
- Syntax: number hostmemory.writeshword(number address, number value)
- Syntax: number hostmemory.writedword(number address, number value)
- Syntax: number hostmemory.writesdword(number address, number value)
- Syntax: number hostmemory.writeqword(number address, number value)
- Syntax: number hostmemory.writesqword(number address, number value)

Write value <value> to elements (little-endian) starting from given address <address>.

- byte is 1 element
- word is 2 elements
- hword is 3 elements
- dword is 4 elements
- qword is 8 elements.
- The 's' variants do signed write.

9.6 hostmemory.write{float,double}: Write to host memory

- syntax: none hostmemory.readfloat(number address, number value)
- Syntax: none hostmemory.readdouble(number address, number value)

Write elements (big-endian) floating-pont to given address <address>, storing <value>.

10 Table movie

Movie handling

10.1 movie.currentframe: Get current frame number

- Syntax: number movie.currentframe()

Return number of current frame.

10.2 movie.framecount: Get move frame count

- Syntax: number movie.framecount()

Return number of frames in movie.

10.3 movie.readonly: Is in readonly mode?

- Syntax: boolean movie.readonly()

Return true if in readonly mode, false if in readwrite.

10.4 movie.rerecords: Movie rerecord count

- Syntax: number movie.rerecords()

Returns the current value of rerecord count.

10.5 movie.set_readwrite: Set read-write mode.

- Syntax: none movie.set_readwrite()

Set readwrite mode (does not cause on_readwrite callback).

10.6 movie.frame_subframes: Count subframes in frame

- Syntax: number movie.frame_subframes(number frame)

Count number of subframes in specified frame <frame> (frame numbers are 1-based) and return that.

10.7 movie.read_subframes: Read subframe data (deprecated)

- Syntax: table movie.read_subframes(number frame, number subframe)

Read specifed subframe in specified frame and return data as array.

10.8 movie.read_rtc: Read current RTC time

- Syntax: (number, number) movie.read_rtc()

Returns the current value of the RTC as a pair (second, subsecond).

10.9 movie.unsafe_rewind: Fast movie rewind to saved state

- Syntax: none movie.unsafe_rewind([UNSAFEREWIND state])

Start setting point for unsafe rewind or jump to point of unsafe rewind.

- If called without argument, causes emulator to start process of setting unsafe rewind point. When this has finished, callback on_set_rewind occurs, passing the rewind state to lua script.
- If called with argument, causes emulator rewind to passed rewind point as soon as possible. Readwrite mode is implicitly activated.

The following warnings apply to unsafe rewinding:

- There are no safety checks against misuse (that's what "unsafe" comes from)!
- Only call rewind from timeline rewind point was set from.
- Only call rewind from after the rewind point was set.

10.10 movie.to_rewind: Load savestate as rewind point

- Syntax: UNSAFEREWIND movie.to_rewind(string filename)

Load specified savestate file <filename> as rewind point and return UNSAFEREWIND corresponding to it.

- Note: This operation does not take emulated time.

10.11 movie.copy_movie/INPUTMOVIE::copy_movie: Copy movie to movie object

- Syntax: INPUTMOVIE movie.copy_movie([INPUTMOVIE movie])
- Syntax: INPUTMOVIE INPUTMOVIE::copy_movie()

Copies specified movie <movie>/current object (if none or nil, the active movie) as new movie object.

10.12 movie.get_frame/INPUTMOVIE::get_frame: Read specified frame in movie.

- Syntax: INPUTFRAME movie.get_frame([INPUTMOVIE movie,] number frame)
- Syntax: INPUTFRAME INPUTMOVIE::get_frame(number frame);

Get INPUTFRAME object corresponding to specified frame in specified movie.

10.13 movie.set_frame/INPUTMOVIE::set_frame: Write speicifed frame in movie.

- Syntax: none movie.set_frame([INPUTMOVIE movie,] number frame, INPUTFRAME data)
- Syntax: none INPUTMOVIE::set_frame(number frame, INPUTFRAME data)

Set data in specified frame.

- Note: Past can't be edited in active movie.

10.14 movie.get_size/INPUTMOVIE::get_size: Get size of movie

- Syntax: integer movie.get_size([INPUTMOVIE movie])
- Syntax: integer INPUTMOVIE::get_size()

Return number of subframes in specified movie.

10.15 movie.count_frames/INPUTMOVIE::count_frames: Count frames in movie

- Syntax: number movie.count_frames([INPUTMOVIE movie])
- Syntax: number INPUTMOVIE::count_frames()

Return number of frames in movie.

10.16 movie.find_frame/INPUTMOVIE::find_frame: Find subframe corresponding to frame

- Syntax: number movie.find_frame([INPUTMOVIE movie], number frame)
- Syntax: number INPUTMOVIE::find_frame(number frame)

Returns starting subframe of given frame (frame numbers are 1-based). Returns -1 if frame number is bad.

10.17 movie.blank_frame/INPUTMOVIE::blank_frame: Return a blank frame

- Syntax: INPUTFRAME movie.blank_frame([INPUTMOVIE movie])
- Syntax: INPUTFRAME INPUTMOVIE::blank_frame()

Return blank INPUTFRAME with frame type from specified movie.

10.18 movie.append_frames/INPUTMOVIE::append_frames: Append blank frames

- Syntax: none movie.append_frames([INPUTMOVIE movie,] number frames)
- Syntax: none INPUTMOVIE::append_frames(number frames)

Append specified number <frames> of frames.

10.19 movie.append_frame/INPUTMOVIE::append_frame: Append a frame

- Syntax: none movie.append_frame([INPUTMOVIE movie,] INPUTFRAME frame)
- Syntax: none INPUTMOVIE::append_frame(INPUTFRAME frame)

Append specified frame <frame>. Past of current movie can't be edited.

10.20 movie.truncate/INPUTMOVIE::truncate: Truncate a movie.

- Syntax: none movie.truncate([INPUTMOVIE movie,] number frames)
- Syntax: none INPUTMOVIE::truncate(number frames)

Truncate the specified movie to specified number of frames.

10.21 movie.edit/INPUTMOVIE::edit: Edit a movie

- Syntax: none movie.edit([INPUTMOVIE movie,] number frame, number port, number controller, number control, number/bool value)
- Syntax: none INPUTMOVIE::edit(number frame, number port, number controller, number control, number/bool value)

Change specified control in specified frame in specified movie. Past can't be edited in active movie.

10.22 movie.copy_frames2: Copy frames between movies

- Syntax: none movie.copy_frames2([INPUTMOVIE dstmov,] number dst, [INPUTMOVIE srcmov,] number src, number count)

Copy specified number of frames between two movies. The copy proceeeds in forward direction.

10.23 movie.copy_frames/INPUTMOVIE::copy_frames: Copy frames in movie

- Syntax: none movie.copy_frames([INPUTMOVIE mov,] number dst, number src, number count, bool backwards)
- Syntax: none INPUTMOVIE::copy_frames(number dst, number src, number count, bool backwards)

Copy specified number of frames from one point in movie to another. If backwards is true, the copy will be done backwards.

10.24 movie.serialize/INPUTMOVIE::serialize: Serialize movie

- Syntax: none movie.serialize([INPUTMOVIE movie,] string filename, bool binary)
- Syntax: none INPUTMOIVE::serialize(string filename, bool binary)

Serialize given movie into file. If binary is true, binary format (more compact and much faster) is used.

10.25 movie.unserialize: Unserialize movie

- Syntax: INPUTMOVIE movie.unserialize(INPUTFRAME template, string filename, bool binary)

Unserialize movie from file. The given frame is used as template to decide the frame type. If binary is true, binary format is decoded (much faster).

10.26 movie.current_first_subframe: Return first subframe in current frame

- Syntax: number movie.current_first_subframe()

Returns first subframe in current frame.

10.27 movie.pollcounter: Return poll counter for speified control

- Syntax: number movie.pollcounter(number port, number controller, number control)

Returns number of times the specified control has been polled this frame.

10.28 INPUTFRAME::get_button: Get button

- Syntax: boolean INPUTFRAME::get_button(number port, number controller, number control)

Returns state of given button as boolean.

10.29 INPUTFRAME::get_axis: Get axis

- Syntax: number INPUTFRAME::get_axis(number port, number controller, number control)

Returns state of given axis as number.

10.30 INPUTFRAME::set_button/INPUTFRAME::set_axis: Set button or axis

- Syntax: none INPUTFRAME::set_button(number port, number controller, number control, number/bool value)
- Syntax: none INPUTFRAME::set_axis(number port, number controller, number control)

Set the given button/axis to given value.

10.31 INPUTFRAME::serialize: Serialize a frame

- Syntax: string INPUTFRAME::serialize()

Return string representation of frame.

10.32 INPUTFRAME::unserialize: Unserialize a frame

- Syntax: none INPUTFRAME::unserialize(string data)

Set current frame from given data.

10.33 INPUTFRAME::get_stride: Get movie stride

- Syntax: number INPUTFRAME::get_stride()

Return number of bytes needed to store the input frame. Mainly useful for some debugging.

11 Table settings

Routines for settings manipulation

11.1 settings.get: Get value of setting

- Syntax: string settings.get(string name)

Get value of setting <name>. If setting value can't be obtained, returns (nil, error message).

11.2 settings.set: Set value of setting

- Syntax: none settings.set(string name, string value)

Set value <value> of setting <name>. If setting can't be set, returns (nil, error message).

12 Table memory

Contains various functions for managing memory

12.1 memory.vma_count: Count number of VMAs.

- Syntax: number memory.vma_count()

Returns the number of VMAs

12.2 memory.read_vma: Lookup VMA info by index

- Syntax: string memory.read_vma(number index)

Reads the specified VMA (indices start from zero). Trying to read invalid VMA gives nil. The read VMA is table with the following fields:

- region_name (string): The readable name of the VMA
- baseaddr (number): Base address of the VMA
- lastaddr (number): Last address in the VMA.
- size (number): The size of VMA in bytes.
- readonly (boolean): True of the VMA corresponds to ROM.
- iospace (boolean): True if the VMA is I/O space.
- native_endian (boolean): True if the VMA has native endian as opposed to little endian.

12.3 memory.find_vma: Find VMA info by address

- Syntax: table memory.find_vma(number address)

Finds the VMA containing specified address. Returns table in the same format as read_vma or nil if not found.

12.4 memory.read{,s}{byte,{,h,d,q}word}: Read memory

- Syntax: none memory.readbyte([string vma,]number address)
- Syntax: none memory.readword([string vma,]number address)
- Syntax: none memory.readhword([string vma,]number address)
- Syntax: none memory.readdword([string vma,]number address)
- Syntax: none memory.readqword([string vma,]number address)
- Syntax: none memory.readsbyte([string vma,]number address)
- Syntax: none memory.readsword([string vma,]number address)
- Syntax: none memory.readshword([string vma,]number address)
- Syntax: none memory.readsdword([string vma,]number address)
- Syntax: none memory.readsqword([string vma,]number address)

Reads the specified address <address> (if 's' variant is used, do undergo 2's complement).

12.5 memory.{,s}read_sg: Scatter/Gather read memory

- Syntax: none memory.read_sg(string/boolean/number...)
- Syntax: none memory.sread_sg(string/boolean/number...)

Perform (2s complement signed if using memory.sread_sg) scatter/gather read of memory. Each argument can be string, boolean or number:

- String: Set VMA addresses are relative to (e.g. 'WRAM').
- boolean: If true, increment relative address by 1, if false, decrement by 1. The new address is read as next higher byte.
- integer: Set the relative address to specified value and read the address as next higher byte.

12.6 memory.write_sg: Scatter/Gather write memory

- Syntax: none memory.write_sg(number value, string/boolean/number...)

Perform scatter/gather write of value <value> on memory. Each argument can be string, boolean or number:

- String: Set VMA addresses are relative to (e.g. 'WRAM').
- boolean: If true, increment relative address by 1, if false, decrement by 1. The new address is read as next higher byte.
- integer: Set the relative address to specified value and read the address as next higher byte.

12.7 memory.read{float,double}: Read memory

- Syntax: none memory.readfloat([string vma,]number address)
- Syntax: none memory.readdouble([string vma,]number address)

Reads the specified address <address>

12.8 memory.write{byte,{,h,d,q}word,float,double}: Write memory

- Syntax: none memory.writebyte([string vma,]number address, number value)
- Syntax: none memory.writeword([string vma,]number address, number value)
- Syntax: none memory.writehword([string vma,]number address, number value)
- Syntax: none memory.writedword([string vma,]number address, number value)
- Syntax: none memory.writeqword([string vma,]number address, number value)
- Syntax: none memory.writefloat([string vma,]number address, number value)
- Syntax: none memory.writedouble([string vma,]number address, number value)

Writes the specified value <value> (negative integer values undergo 2's complement) to specified address <address>.

12.9 memory.map{{,s}{byte,{,h,d,q}word},float,double}: Map an array

- Syntax: userdata memory.map<type>([[string vma,]number base, number size])

Returns a table mapping specified memory aperture for read/write. If parameters are omitted, entiere map space is the aperture.

- Type may be one of: byte, sbyte, word, sword, hword, shword, dword, sdword, qword, sqword, float or double.

12.10 memory.hash_region: Hash region of memory

- Syntax: string memory.hash_region([string vma,]number base, number size)

Hash specified number of bytes starting from specified address and return the SHA-256.

12.11 memory.hash_state: Hash system state

- Syntax: string memory.hash_state()

Hash the current system state. Mainly useful for debugging savestates.

12.12 memory.readregion: Read region of memory

- Syntax: table memory.readregion([string vma,]number base, number size)

Read a region of memory.

- Warning: If the region crosses VMA boundary, the results are undefined.

12.13 memory.writeregion: Write region of memory

- Syntax: none memory.writeregion([string vma,]number base, number size, table data)

Write a region of memory.

- Warning: If the region crosses VMA boundary, the results are undefined.

12.14 memory.map_structure: Create mmap structure

- syntax: MMAP_STRUCT memory.map_structure()

Returns a new mapping structure (MMAP_STRUCT)

12.15 MMAP_STRUCT(): Bind key in mmap structure

- Syntax: none MMAP_STRUCT(string key, [string vma,]number address, string type)

Bind key <key> in mmap structure to specified address <address> with specified type <type>.

- Type may be one of: byte, sbyte, word, sword, hword, shword, dword, sdword, qword, sqword, float or double.

12.16 memory.read_expr: Evaluate memory watch expression

- Syntax: string memory.read_expr(string expr)

Evaluate specified watch expression and return result

12.17 memory.action: Run core action

- memory.action(string action, [<params>])

Run core action. The different models expect parameters as:

- string: String
- numeric: numeric
- enumeration: String
- boolean: String
- toggle: None.

12.18 memory.get_lag_flag: Get lag flag

- Syntax: boolean memory.get_lag_flag()

Get the value of core lag flag. True if this frame has been lag so far, false if poll has been detected.

12.19 memory.set_lag_flag: Set lag flag

- Syntax: none memory.set_lag_flag(boolean flag)

Set the value of core lag flag. This flag automatically gets cleared if poll is detected, but can be forcibly set or cleared if game so requires.

- Should only be used in on_frame_emulated callback.
- Setting or clearing this affects the emulator lag counter.

12.20 memory.{,un}register{read,write,exec}: (Un)Register read / write / execute callback

- Syntax: function memory.registerread([string vma,] number addr, function fn);
- Syntax: function memory.registerwrite([string vma,] number addr, function fn);
- Syntax: function memory.registerexec([string vma,] number addr, function fn);
- Syntax: none memory.unregisterread([string vma,] number addr, function fn);
- Syntax: none memory.unregisterwrite([string vma,] number addr, function fn);
- Syntax: none memory.unregisterexec([string vma,] number addr, function fn);

Add or remove callback on memory read, write or execute (depending on the function). If <vma> is specified, <addr> is relative to it, otherwise <addr> is global. <fn> is the callback. The register* functions return <fn> (which can then be passed to unregister* functions.

- Not all cores support this, and it may be unsupported for some VMAs.
- The functions are passed two parameters: Address and value.

12.21 memory.{,un}registertrace: Set/Clear trace hook

- Syntax: function memory.registertrace(number processor, function fn);
- Syntax: none memory.unregistertrace(number processor, function fn);

Add or remove trace callback. <processor> is system-dependent processor number (0 is usually main CPU). The function arguments work like in other (un)register* functions.

- The functions are passed two parameters: Trace CPU and Trace event string.

12.22 memory.cheat: Set cheat

- Syntax: none memory.cheat([string vma,] number addr, number value);
- Syntax: none memory.cheat([string vma,] number addr);

Set or clear cheat (value <value>) on address <addr>. If <vma> is specified, <addr> is relative to that. If <value> is not speicified, clear a cheat.

- Not all cores support this, and it may be unsupported for some VMAs.

12.23 memory.setxmask: Set global execute hook mask

- Syntax: none memory.setxmask(number mask)

Set the global execute hook mask to <mask>. The meaning of each bit is system-dependent, but bit 0 should be the main CPU.

13 Table memory2

Contains newer memory functions.

13.1 memory2(): Get all VMA names.

- Syntax: table memory2()

Returns array of all valid VMA names.

13.2 memory2.<vma>:info: Get VMA info

- Syntax: table memory2.<vma>:info()

Return table describing given VMA. Includes fields address, size, last, readonly, special and endian.

13.3 memory2.<vma>:<op>: Read/Write memory

- Syntax: none memory2.<vma>:<op>(number offset, number value)
- Syntax: number memory2.<vma>:<op>(number offset)

Read/Write value from/to given VMA <vma> at given offset <offset> (must be in-range). The value written is <value>. <Op> is of form: [i][s]<type>, where:

- <type> is one of 'byte', 'word', 'hword', 'dword', 'qword', 'float', 'double'.
- 'i' signifies that the value is treated as opposite-to-normal endianess,
- 's' signifies that value is treated as signed (not available for floating-point).

13.4 memory2.<vma>:read: Scatter-gather value read

- Syntax: number memory2.<vma>:read(number addr...)

Read value from given VMA <vma> at byte offsets <addr>..., given in order of increasing significance. Value of true and false are special. True increments address by 1, and false decrements address by 1.

13.5 memory2.<vma>:sread: Signed scatter-gather value read

- Syntax: number memory2.<vma>:sread(number addr...)

Like memory2.<vma>:read, but reads signed values.

13.6 memory2.<vma>:write: Scatter-gather value write

- Syntax: number memory2.<vma>:write(number val, number addr...)

Write value <val> to given VMA <vma> at byte offsets <addr>..., given in order of increasing significance. Value of true and false are special. True increments address by 1, and false decrements address by 1.

14 Table random

Contains random number generation methods. These functions do not return reproducable results.

14.1 random.boolean: Random boolean

- Syntax: boolean random.boolean()

Returns true or false at random (50-50 chance).

14.2 random.integer: Random integer

- Syntax: number random.integer(number highplusone)
- Syntax: number random.integer(number low, number high)

With one argument, return random integer [0,<highplusone>) (upper end exclusive). With two arguments, return random integer [<low>,<high>] (both ends inclusive).

The returned numbers are from uniform distribution.

14.3 random.float: Random float

- Syntax: number random.float()

Returns random decimal number [0,1).

14.4 random.among: Random parameter

- Syntax: value random.among(value values...)

Returns random parameter value, picked at uniform. Multiple equivalent values are returned with higher chance.

14.5 random.amongtable: Random from table

- Syntax: value random.amongtable(table tab)

Returns random value from table <tab>. As in random.among, no equality testing is done.

15 Table callback

Various callback-related functions.

15.1 callback.register: Register a callback

- Syntax: function callback.register(string cbname, function cbfun);

Instruct function <cbfun> to be added to list of callbacks to call on event <cbname> (See section 18). The callback name does not have the 'on_' prefix (e.g. "paint"). Returns <cbfun>.

15.2 callback.unregister: Unregister a callback

- Syntax: function callback.unregister(string cbname, function cbfun);

Instruct function <cbfun> to be removed from list of callbacks to call on event <cbname>.

15.3 callback.<cbname>:register: Register callback

- Syntax: function callback.<cbname>:register(function cbfun)

Synonym for callback.register (section 15.1), albeit with callback name specified differently.

15.4 callback.<cbname>:unregister: Register callback

- Syntax: function callback.<cbname>:unregister(function cbfun)

Synonym for callback.unregister (section 15.2), albeit with callback name specified differently.

16 table bsnes

Various bsnes-specific functions.

16.1 bsnes.dump_sprite: Dump a sprite

- Syntax: BITMAP bsnes.dump_sprite([string vma,] number addr, number width, number height[, number stride])

Dumps given sprite (in native format) from memory. VMA is usually "VRAM". <Width> and <height> are given in 8x8 blocks. <Stride> overrides row stride (default 512).

16.2 bsnes.dump_palette: Dump a palette

- Syntax: PALETTE bsnes.dump_palette([string vma,] number addr, bool full256, bool first_trans)

Dumps a palette from memory. VMA is usually "CGRAM". If <full256> is true, 256 colors are dumped (otherwise 16). If <first_trans> is true, first color is forced transparent.

17 Table _SYSTEM

Contains copy of global variables from time of Lua initialization. Non-writeable.

18 Callbacks

Various callbacks to Lua that can occur.

18.1 on_paint: Screen is being painted

- Callback: on_paint(bool not_synth)

Called when screen is being painted. Any gui.* calls requiring graphic context draw on the screen.

- not_synth is true if this hook is being called in response to received frame, false otherwise.

18.2 on_video: Dumped video frame is being painted

- Callback: on_video()

Called when video dump frame is being painted. Any gui.* calls requiring graphic context draw on the video.

18.3 on_frame_emulated: Frame emulation complete

- Callback: on_frame_emulated()

Called when emulating frame has completed and on_paint()/on_video() calls are about to be issued.

18.4 on_frame: Frame emulation starting.

- Callback: on_frame()

Called on each starting whole frame.

18.5 on_startup: Emulator startup complete

- Callback: on_startup()

Called when the emulator is starting (lsnes.rc and –run files has been run).

18.6 on_rewind: Movie rewound to beginning

- Callback: on_rewind()

Called when rewind movie to beginning has completed.

18.7 on_pre_load: Load operation is about to start

- Callback: on_pre_load(string name)

Called just before savestate/movie load occurs (note: loads are always delayed, so this occurs even when load was initiated by lua).

18.8 on_err_Load: Load failed

- Callback: on_err_load(string name)

Called if loadstate goes wrong.

18.9 on_post_load: Load completed

- Callback: on_post_load(string name, boolean was_savestate)

Called on successful loadstate. was_savestate gives if this was a savestate or a movie.

18.10 on_pre_save: Save operation is about to start

- Callback: on_pre_save(string name, boolean is_savestate)

Called just before savestate save occurs (note: movie saves are synchronous and won't trigger these callbacks if called from Lua).

18.11 on_err_save: Save failed

- Callback: on_err_save(string name)

Called if savestate goes wrong.

18.12 on_post_save: Save completed

- Callback: on_post_save(string name, boolean is_savestate)

Called on successful savaestate. is_savestate gives if this was a savestate or a movie.

18.13 on_quit: Emulator is shutting down

- Callback: on_quit()

Called when emulator is shutting down.

18.14 on_input: Polling for input

Called when emulator is just sending input to bsnes core. Warning: This is called even in readonly mode, but the results are ignored.

18.15 on_reset: System has been reset

- Callback: on_reset()

Called when system is reset.

18.16 on_readwrite: Entered readwrite mode

- Callback: on_readwrite()

Called when moving into readwrite mode as result of "set-rwmode" command (note: moving to rwmode by Lua won't trigger this, as per recursive entry protection).

18.17 on_snoop/on_snoop2: Snoop core controller reads

- Callback: on_snoop(number port, number controller, number index, number value)
- Callback: on_snoop2(number port, number controller, number index, number value)

Called each time bsnes asks for input. The value is the final value to be sent to bsnes core (readonly mode, autohold and autofire have been taken into account). Might be useful when translating movies to format suitable for console verification. Note: There is no way to modify the value to be sent.

- On_snoop2 is called instead of on_snoop if defined. Reserves port 0 for system, having first user port be port 1.

18.18 on_keyhook: Hooked key/axis has been moved

- Callback: on_keyhook(string keyname, table state)

Sent when key that has keyhook events requested changes state. Keyname is name of the key (group) and state is the state (same kind as table values in input.raw).

18.19 on_idle: Idle event

- Callback: on_idle()

Called when requested by set_idle_timeout(), the timeout has expired and emulator is waiting.

18.20 on_timer: Timer event

- Callback: on_timer()

Called when requested by set_idle_timeout() and the timeout has expired (regardless if emulator is waiting).

18.21 on_set_rewind: Rewind point has been set

- Callback: on_set_rewind(UNSAFEREWIND r)

Called when unsafe rewind object has been constructed.

18.22 on_pre_rewind: Rewind is about to occur

- Callback: on_pre_rewind()

Called just before unsafe rewind is about to occur.

18.23 on_post_rewind: Rewind has occured

- Callback: on_post_rewind()

Called just after unsafe rewind has occured.

18.24 on_button: Button has been pressed

- Callback: on_button(number port, number controller, number index, string type)

Called on controller button press, with following parameters:

- port: Port number (0 is system)
- controller: Controller within port
- index: Index of button.
- type: Type of event, one of:
  - "pressed": Button was pressed.
  - "released": Button was released.
  - "hold": Held.
  - "unhold": Released from hold.
  - "type": Typing input on button.
  - "untype": Typing input undone.
  - "autofire <duty> <cycle>": Autofire with specifie duty and cycle.
  - "autofire": Stop autofire.
  - "analog": Analog action on axis.

18.25 on_movie_lost: Movie data is about to be lost

- Callback: on_movie_lost(STRING kind)

Called just before something would happen that could lose movie data. Kind can be:

- readwrite: Switching to readwrite mode.
- reload: ROM is being reloaded in readwrite mode.
- load: New movie is being loaded.
- unsaferewind: Unsafe rewind is happening.

18.26 on_latch: Latch line is rising

- Callback: on_latch(<core-dependent-parameters>)

Called when latch line for controller is rising. Some cores may not support this.

19 System-dependent behaviour

19.1 bsnes core

- Registers are: pbpc, pb, pc, r0, r1, r2, r3, r4, r5, a, x, y, z, s, d, db, p, e, irq, wai, mdr, vector, aa, rd, sp, dp, p_n, p_v, p_m, p_x, p_d, p_i, p_z, p_c
- on_latch has no parameters
- CPU 0 is S-CPU, 1 is S-SMP.
- Cheats are supported for ROM, SRAM, WRAM, BSXFLASH, SLOT{A,B}_{RAM,ROM}.
- Read/Write/Execute hooks are supported for ROM, SRAM, WRAM, BSXFLASH, SLOT{A,B}_{RAM,ROM}.

19.2 gambatte core

- Registers are: wrambank, cyclecounter, pc, sp, hf1, hf2, zf, cf, a, b, c, d, e, f, h, l
- on_latch is not supported
- CPU 0 is main CPU.
- Cheats are supported for ROM, SRAM and WRAM.
- Read/Write/Execute hooks are supported for ROM (read/execute only), SRAM and WRAM.