# Rinstrum Lua Commands (v5.1)

## Rinstrum Documents - available from www.rinstrum.com or USB memory stick (A10030)

- Rinstrum Application Package and API Reference (L001-503)
- Rinstrum Environment Setup (L001-506)
- Rinstrum Lua Reference Manual (L001-600)
- Rinstrum Lua Quick Start M4223 (L001-601)
- Rinstrum Linux Commands (L001-602)

### External References

- Online Lua tutorials http://lua-users.org/wiki/TutorialDirectory
- Online Lua reference manual http://www.lua.org/pil/contents.html

## This listing is an extract from http://thomaslauer.com/download/luarefv51single.pdf

## Reserved identifiers and comments

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| and | break | do | else | elseif | end | false | for | function | if | in |
| local | nil | not | or | repeat | return | then | true | until | while | |
| -- ... | comment to end of line | | | | --[=[ ]=] | multi line comment (zero or multiple '=' are valid) | | | | |
| _X is "reserved" (by convention) for constants (with X being any sequence of uppercase letters) | | | | | #! | usual Unix shebang; Lua ignores whole first line if this starts the line. | | | | |

## Types (the string values are the possible results of base library function type())

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| "nil" | "boolean" | "number" | "string" | "table" | "function" | "thread" | "userdata" |

Note: for type boolean, nil and false count as false; everything else is true (including 0 and "").

## Strings and escape sequences

| | | | | | | |
|---|---|---|---|---|---|---|
| '...' and "..." | string delimiters; interpret escapes. | | [=[...]=] | multi line string; escape sequences are ignored. | | |
| \a bell | \b backspace | \f form feed | \n newline | \r return | \t horiz. tab | \v vert. tab |
| \\ backslash | \" d. quote | \' quote | \[ sq. bracket | \] sq. bracket | \ddd decimal (up to 3 digits) | |

## Operators, decreasing precedence

| | | | | | |
|---|---|---|---|---|---|
| **^** (exponential) math library required | | | | | |
| **not** | # (length of strings and tables) | | | - (unary) | |
| (multiplication) | **/** (division) | | | **%** (modulo) | |
| **+** (addition) | (subtraction) | | | | |
| **..** (string concatenation, right associative) | | | | | |
| **<** (less than) | > (greater than) | **<=**(less than or equal) | **>=**(greater than or equal) | **~=** (not equal) | == (equality) |
| **and** (stops on **false** or **nil**, returns last evaluated value) | | | | | |
| **or** (stops on **true** (not **false** or **nil**), returns last evaluated value) | | | | | |

## Assignment and coercion

| | |
|---|---|
| a = 5  b= "hi" c = 0x64 local a = a | simple assignment; variables are not typed and can hold different types. 0x for hexadecimal constants<br>Local variables are lexically scoped; their scope begins after the full declaration (so that local a = 5). |
| a, b, c = 1, 2, 3 | multiple assignments are supported |
| a, b = b, a | swap values: right hand side is evaluated before assignment takes place |
| a, b = 4, 5, "6" | excess values on right hand side ("6") are evaluated but discarded |
| a, b = "there" | for missing values on right hand side nil is assumed |
| a = nil | destroys a; its contents are eligible for garbage collection if unreferenced. |
| a = z | if z is not defined it is nil, so nil is assigned to a (destroying it) |
| a = "3" + "2" | numbers expected, strings are converted to numbers (a = 5) |
| a = 3 .. 2 | strings expected, numbers are converted to strings (a = "32") |

## Control structures

| | |
|---|---|
| **do** *block* **end** | block; introduces local scope. |
| **if** *exp* **then** *block* {**elseif** *exp* **then** *block*} [**else** *block*] **end** | conditional execution |
| **while** *exp* **do** *block* **end** | loop as long as *exp* is true |
| **repeat** *block* **until** *exp* | exits when *exp* becomes true; *exp* is in loop scope. |
| **for** *var* = *start*, *end* [, *step*] **do** *block* **end** | numerical for loop; *var* is local to loop. |
| **for** *vars* **in** *iterator* **do** *block* **end** | iterator based for loop; *vars* are local to loop. |
| **break** | exits loop; must be last statement in block. |

## Table constructors

| | |
|---|---|
| t = {} | creates an empty table and assigns it to t |
| t = {"yes", "no", "?"} | simple array; elements are t[1], t[2], t[3]. |
| t = { [1] = "yes", [2] = "no", [3] = "?" } | same as above, but with explicit fields |
| t = {[-900] = 3, [900] = 4} | sparse array with just two elements (no space wasted) |
| t = {x=5, y=10} | hash table, fields are t["x"], t["y"] (or t.x, t.y) |
| t = {x=5, y=10; "yes", "no"} | mixed, fields/elements are t.x, t.y, t[1], t[2] |
| t = {msg = "choice", {"yes", "no", "?"}} | tables can contain others tables as fields |

## Function definition

| | |
|---|---|
| function *name* ( *args* ) *body* [return *values*] end | defines function and assigns to global variable name |
| local function *name* ( *args* ) *body* [return *values*] end | defines function as local to chunk |
| f = function ( *args* ) *body* [return *values*] end | anonymous function assigned to variable f |
| function ( [*args*, ] ... ) *body* [return *values*] end | variable argument list, in *body* accessed as ... |
| function *t.name* ( *args* ) *body* [return *values*] end | shortcut for *t.name* = function ... |
| function *obj*:*name* ( *args* ) *body* [return *values*] end | object function, gets *obj* as additional first argument self |

## Function call

| | |
|---|---|
| f (x) | simple call, possibly returning one or more values |
| t.f (x) | calling a function assigned to field f of table t |
| x:move (2, -3) | object call: shortcut for x.move(x, 2, -3) |

## Environment and global variables

| | |
|---|---|
| getfenv ([f]) | if f is a function, returns its environment; if f is a number, returns the environment of function at level f (1 = current [default], 0 = global); if the environment has a field __fenv, returns that instead. |
| setfenv (f, t) | sets environment for function f (or function at level f, 0 = current thread); if the original environment has a field __fenv, raises an error. Returns function f if f ~= 0. |
| _G | global variable whose value is the global environment (that is, _G._G == _G) |
| _VERSION | global variable containing the interpreter's version (e.g. "Lua 5.1") |

## Loading and executing

| | |
|---|---|
| **require** (pkgname) | loads a package, raises error if it can't be loaded |
| **dofile** ([filename]) | loads and executes the contents of **filename** [default: standard input]; returns its returned values. |
| **load** (func [, chunkname]) | loads a chunk (with chunk name set to **name**) using function **func** to get its pieces; returns compiled chunk as function (or **nil** and error message). |
| **loadfile** (filename) | loads file **filename**; return values like **load()**. |
| **loadstring** (s [, name]) | loads string **s** (with chunk name set to **name**); return values like **load()**. |

## Simple output and error feedback

| | |
|---|---|
| **print** (*args*) | prints each of the passed *args* to stdout using **tostring()** (see below) |
| **error** (msg [, n]) | terminates the program or the last protected call (e.g. **pcall()**) with error message **msg** quoting level **n** [default: 1, current function] |
| **assert** (v [, msg]) | calls **error(msg)** if **v** is **nil** or **false** [default **msg**: **"assertion failed!"**] |

## Information and conversion

| | |
|---|---|
| **select** (index, ...) | returns the arguments after argument number **index** or (if index is "#") the total number of arguments it received after **index** |
| **type** (x) | returns the type of **x** as a string (e.g. **"nil"**, **"string"**); see *Types* above. |
| **tostring** (x) | converts **x** to a string, using **t**'s metatable's **__tostring** if available |
| **tonumber** (x [, b]) | converts string **x** representing a number in base **b** [2..36, default: 10] to a number, or **nil** if invalid; for base 10 accepts full format (e.g. "1.5e6"). |
| **unpack** (t) | returns **t[1]**..**t[n]** (n = **#t**) as separate values |

## Iterators

| | |
|---|---|
| ipairs (t) | returns an iterator getting index, value pairs of array t in numerical order |
| pairs (t) | returns an iterator getting key, value pairs of table t in an unspecified order |
| next (t [, inx]) | if inx is nil [default] returns first index, value pair of table t; if inx is the previous index returns next index, value pair or nil when finished. |

## Modules and the package library [package]

| | |
|---|---|
| package.path, package.cpath | contains the paths used by require() to search for a Lua or C loader, respectively |
| package.loaded | a table used by require to control which modules are already loaded (see module) |

## The coroutine library [coroutine]

| | |
|---|---|
| coroutine.create (f) | creates a new coroutine with Lua function f() as body and returns it |
| coroutine.resume (co, *args*) | starts or continues running coroutine co, passing *args* to it; returns true (and possibly values) if co calls coroutine.yield() or terminates or false and an error message. |
| coroutine.yield (*args*) | suspends execution of the calling coroutine (not from within C functions, metamethods or iterators); any *args* become extra return values of coroutine.resume(). |
| coroutine.status (co) | returns the status of coroutine co: either "running", "suspended" or "dead" |
| coroutine.running () | returns the running coroutine or nil when called by the main thread |
| coroutine.wrap (f) | creates a new coroutine with Lua function f as body and returns a function; this function will act as coroutine.resume() without the first argument and the first return value, propagating any errors. |

## The table library [table]

| | |
|---|---|
| table.insert (t, [i,] v) | inserts v at numerical index i [default: after the end] in table t |
| table.remove (t [, i]) | removes element at numerical index i [default: last element] from table t; returns the removed element or nil on empty table. |
| table.maxn (t) | returns the largest positive numerical index of table t or zero if t has no positive indices |
| table.sort (t [, cf]) | sorts (in place) elements from t[1] to #t, using compare function cf(e1, e2) [default: '<'] |
| table.concat (t [, s [, i [, j]]]) | returns a single string made by concatenating table elements t[i] to t[j] [default: i =1, j = #t] separated by string s; returns empty string if no elements exist or i > j. |

## The mathematical library [math]

### Basic operations

| | |
|---|---|
| math.abs (x) | returns the absolute value of x |
| math.mod (x, y) | returns the remainder of x / y as a rounded-down integer, for y ~= 0 |
| math.floor (x) | returns x rounded down to the nearest integer |
| math.ceil (x) | returns x rounded up to the nearest integer |
| math.min (*args*) | returns the minimum value from the *args* received |
| math.max (*args*) | returns the maximum value from the *args* received |

### Splitting on powers of 2

| | |
|---|---|
| math.frexp (x) | splits x into normalized fraction and exponent of 2 and returns both |
| math.ldexp (x, y) | returns x * (2 ^ y) with x = normalized fraction, y = exponent of 2 |

### Pseudo-random numbers

| | |
|---|---|
| math.random ([n [, m]) | returns a pseudo-random number in range [0, 1] if no arguments given; in range [1, n] if n is given, in range [n, m] if both n and m are passed. |
| math.randomseed (n) | sets a seed n for random sequence (same seed = same sequence) |

### Trigonometrical,

| | |
|---|---|
| math.sqrt (x) | returns the square root of x, for x >= 0 |
| math.pow (x, y) | returns x raised to the power of y, i.e. x^y; if x < 0, y must be integer. |
| __pow (x, y) | global function added by the math library to make operator '^' work |
| math.exp (x) | returns e (base of natural logs) raised to the power of x, i.e. e^x |
| math.log (x) | returns the natural logarithm of x, for x >= 0 |
| math.log10 (x) | returns the base-10 logarithm of x, for x >= 0 |

## Exponential and logarithmic

| | |
|---|---|
| math.deg (a) | converts angle a from radians to degrees |
| math.rad (a) | converts angle a from degrees to radians |
| math.pi | constant containing the value of pi |
| math.sin (a) | returns the sine of angle a (measured in radians) |
| math.cos (a) | returns the cosine of angle a (measured in radians) |
| math.tan (a) | returns the tangent of angle a (measured in radians) |
| math.asin (x) | returns the arc sine of x in radians, for x in [-1, 1] |
| math.acos (x) | returns the arc cosine of x in radians, for x in [-1, 1] |
| math.atan (x) | returns the arc tangent of x in radians |
| math.atan2 (y, x) | similar to math.atan(y / x) but with quadrant and allowing x = 0 |

## The string library [string]

Note: string indexes extend from 1 to #string, or from end of string if negative (index -1 refers to the last character).

Note: the string library sets a metatable for strings where the __index field points to the string table. String functions can be used in object-oriented style, e.g. string.len(s) can be written s:len(); literals have to be enclosed in parentheses, e.g. ("xyz"):len().

## Basic operations

| | |
|---|---|
| string.len (s) | returns the length of string s, including embedded zeros (see also # operator) |
| string.sub (s, i [, j]) | returns the substring of s from position i to j [default: -1] inclusive |
| string.rep (s, n) | returns a string made of n concatenated copies of string s |
| string.upper (s) | returns a copy of s converted to uppercase according to locale |
| string.lower (s) | returns a copy of s converted to lowercase according to locale |

## Character codes

| | |
|---|---|
| string.byte (s [, i [, j]]) | returns the platform-dependent numerical code (e.g. ASCII) of characters s[i], s[i+1], ···, s[j]. The default value for i is 1; the default value for j is i. |
| string.char (*args*) | returns a string made of the characters whose platform-dependent numerical codes are passed as *args* |

## Function storage

| | |
|---|---|
| string.dump (f) | returns a binary representation of function f(), for later use with loadstring() (f() must be a Lua function with no upvalues) |

*Formatting*

| | |
|---|---|
| string.format (s [, *args*]) | returns a copy of s where formatting directives beginning with '%' are replaced by the value of arguments *args*, in the given order (see *Formatting directives* below) |

## Formatting directives for string.format

% [*flags*] [*field_width*] [.*precision*] *type*

## Formatting field types

| | |
|---|---|
| %d | decimal integer |
| %o | octal integer |
| %x | hexadecimal integer, uppercase if %X |

| | |
|---|---|
| %f | floating-point in the form [-]nnnn.nnnn |
| %e | floating-point in exp. Form [-]n.nnnn e [+\|-]nnn, uppercase if %E |
| %g | floating-point as %e if exp. < -4 or >= precision, else as %f; uppercase if %G. |
| %c | character having the (system-dependent) code passed as integer |
| %s | string with no embedded zeros |
| %q | string between double quotes, with all special characters escaped |
| %% | '%' character |

## Formatting flags

| | |
|---|---|
| - | left-justifies within field_width [default: right-justify] |
| + | prepends sign (only applies to numbers) |
| (space) | prepends sign if negative, else blank space |
| # | adds "0x" before %x, force decimal point for %e, %f, leaves trailing zeros for %g |

## Formatting field width and precision

| | |
|---|---|
| n | puts at least n (<100) characters, pad with blanks |
| 0n | puts at least n (<100) characters, left-pad with zeros |
| .n | puts at least n (<100) digits for integers; rounds to n decimals for floating-point; puts no more than n (<100) characters for strings. |

## Formatting examples

| | |
|---|---|
| string.format("results: %d, %d", 13, 27) | results: 13, 27 |
| string.format("<%5d>", 13) | <  13> |
| string.format("<%-5d>", 13) | <13  > |
| string.format("<%05d>", 13) | <00013> |
| string.format("<%06.3d>", 13) | <  013> |
| string.format("<%f>", math.pi) | <3.141593> |
| string.format("<%e>", math.pi) | <3.141593e+00> |
| string.format("<%.4f>", math.pi) | <3.1416> |
| string.format("<%9.4f>", math.pi) | <  3.1416> |
| string.format("<%c>", 64) | <@> |
| string.format("<%.4s>", "goodbye") | <good> |
| string.format("%q", [[she said "hi"]]) | "she said \"hi\"" |

## Finding, replacing, iterating (for the Patterns see below)

| | |
|---|---|
| string.find (s, p [, i [, d]]) | returns first and last position of pattern p in string s, or nil if not found, starting search at position i [default: 1]; returns captures as extra results. If d is true, treat pattern as plain string. |
| string.gmatch (s, p) | returns an iterator getting next occurrence of pattern p (or its captures) in string s as substring(s) matching the pattern. |
| string.gsub (s, p, r [, n]) | returns a copy of s with up to n [default: all] occurrences of pattern p (or its captures) replaced by r if r is a string (r can include references to captures in the form *%n*). If r is a function r() is called for each match and receives captured substrings; it should return the replacement string. If r is a table, the captures are used as fields into the table. The function returns the number of substitutions made as second result. |
| string.match (s, p [, i]) | returns captures of pattern p in string s (or the whole match if p specifies no captures) or nil if p does not match s; starts search at position i [default: 1]. |

## Patterns and pattern items

| General pattern format: *pattern_item* [ *pattern_items* ] | |
|---|---|
| *cc* | matches a single character in the class *cc* (see *Pattern character classes* below) |
| *cc** | matches zero or more characters in the class *cc*; matchest longest sequence (greedy). |
| *cc-* | matches zero or more characters in the class *cc*; matchest shortest sequence (non-greedy). |
| *cc+* | matches one or more characters in the class *cc*; matchest longest sequence (greedy). |
| *cc?* | matches zero or one character in the class *cc* |
| *%n* | matches the *n*-th captured string (*n* = 1..9, see *Pattern captures*) |
| %b*xy* | matches the balanced string from character *x* to character *y* (e.g. %b() for nested parentheses) |
| ^ | anchors pattern to start of string, must be the first item in the pattern |
| $ | anchors pattern to end of string, must be the last item in the pattern |

## Captures

| | |
|---|---|
| (*pattern*) | stores substring matching *pattern* as capture %1..%9, in order of opening parentheses |
| () | stores current string position as capture |

## Pattern character classes

| | | | |
|---|---|---|---|
| . | any character | | |
| %a | any letter | %A | any non-letter |
| %c | any control character | %C | any non-control character |
| %d | any digit | %D | any non-digit |
| %l | any lowercase letter | %L | any non-(lowercase letter) |
| %p | any punctuation character | %P | any non-punctuation character |
| %s | any whitespace character | %S | any non-whitespace character |
| %u | any uppercase letter | %U | any non-(uppercase letter) |
| %w | any alphanumeric character | %W | any non-alphanumeric character |
| %x | any hexadecimal digit | %X | any non-(hexadecimal digit) |
| %z | the byte value zero | %Z | any non-zero character |
| %*x* | if *x* is a symbol the symbol itself | *x* | if *x* not in ^$()%.[]*+-? the character itself |
| [ *set* ] | any character in any of the given classes; can also be a range [*c1-c2*], e.g. [a-z]. | [ ^*set* ] | any character not in *set* |

## Pattern examples

| | | |
|---|---|---|
| string.find("Lua is great!", "is") | 5 | 6 |
| string.find("Lua is great!", "%s") | 4 | 4 |
| string.gsub("Lua is great!", "%s", "-") | Lua-is-great! | 2 |
| string.gsub("Lua is great!", "[%s%l]", "*") | L**********! | 11 |
| string.gsub("Lua is great!", "%a+", "*") | * * *! | 3 |
| string.gsub("Lua is great!", "(.)", "%1%1") | LLuuaa iiss ggrreeaatt!! | 13 |
| string.gsub("Lua is great!", "%but", "") | L! | 1 |
| string.gsub("Lua is great!", "^.-a", "LUA") | LUA is great! | 1 |
| string.gsub("Lua is great!", "^.-a", function(s) return string.upper(s) end) | LUA is great! | 1 |

## The I/O library [io]

### Complete I/O

| | |
|---|---|
| io.open (fn [, m]) | opens file with name fn in mode m: "r" = read [default], "w" = write", "a" = append, "r+" = update-preserve, "w+" = update-erase, "a+" = update-append (add trailing "b" for binary mode on some systems); returns a file object (a userdata with a C handle). |
| file:close () | closes file |
| file:read (*formats*) | returns a value from file for each of the passed *formats*: "*n" = reads a number, "*a" = reads the whole file as a string from current position (returns "" at end of file), "*l" = reads a line (nil at end of file) [default], *n* = reads a string of up to *n* characters (nil at end of file) |
| file:lines () | returns an iterator function for reading file line by line; the iterator does not close the file when finished. |
| file:write (*values*) | writes each of the *values* (strings or numbers) to file, with no added separators. Numbers are written as text, strings can contain binary data (in this case, file may need to be opened in binary mode on some systems). |
| file:seek ([p] [, of]) | sets the current position in file relative to p ("set" = start of file [default], "cur" = current, "end" = end of file) adding offset of [default: zero]; returns new current position in file. |
| file:flush () | flushes any data still held in buffers to file |

### Simple I/O

| | |
|---|---|
| io.input ([file]) | sets file as default input file; file can be either an open file object or a file name; in the latter case the file is opened for reading in text mode. Returns a file object, the current one if no file given; raises error on failure. |
| io.output ([file]) | sets file as default output file (the current output file is not closed); file can be either an open file object or a file name; in the latter case the file is opened for writing in text mode. Returns a file object, the current one if no file given; raises error on failure. |
| io.close ([file]) | closes file (a file object) [default: closes the default output file] |
| io.read (*formats*) | reads from the default input file, usage as file:read() |
| io.lines ([fn]) | opens the file with name fn for reading and returns an iterator function to read line by line; the iterator closes the file when finished. If no fn is given, returns an iterator reading lines from the default input file. |
| io.write (*values*) | writes to the default output file, usage as file:write() |
| io.flush () | flushes any data still held in buffers to the default output file |

### Standard files and utility functions

| | |
|---|---|
| io.stdin, io.stdout, io.stderr | predefined file objects for stdin, stdout and stderr streams |
| io.popen ([prog [, mode]]) | starts program prog in a separate process and returns a file handle that you can use to read data from (if mode is "r", default) or to write data to (if mode is "w") |
| io.type (x) | returns the string "file" if x is an open file, "closed file" if x is a closed file or nil if x is not a file object |
| io.tmpfile () | returns a file object for a temporary file (deleted when program ends) |

Note: unless otherwise stated, the I/O functions return nil and an error message on failure; passing a closed file object raises an error instead.

## The operating system library [os]

## System interaction

| | |
|---|---|
| os.execute (cmd) | calls a system shell to execute the string cmd as a command; returns a system-dependent status code. |
| os.exit ([code]) | terminates the program returning code [default: success] |
| os.getenv (var) | returns a string with the value of the environment variable var or nil if no such variable exists |
| os.setlocale (s [, c]) | sets the locale described by string s for category c: "all", "collate", "ctype", "monetary", "numeric" or "time" [default: "all"]; returns the name of the locale or nil if it can't be set. |
| os.remove (fn) | deletes the file fn; in case of error returns nil and error description. |
| os.rename (of, nf) | renames file of to nf ; in case of error returns nil and error description. |
| os.tmpname () | returns a string usable as name for a temporary file; subject to name conflicts, use io.tmpfile() instead. |

## Date/time

| | |
|---|---|
| os.clock () | returns an approximation of the amount in seconds of CPU time used by the program |
| os.time ([tt]) | returns a system-dependent number representing date/time described by table tt [default: current]. tt must have fields year, month, day; can have fields hour, min, sec, isdst (daylight saving, boolean). On many systems the returned value is the number of seconds since a fixed point in time (the "epoch"). |
| os.date ([fmt [, t]]) | returns a table or a string describing date/time t (should be a value returned by os.time() [default: current date/time]), according to the format string fmt [default: date/time according to locale settings]; if fmt is "*t" or "!*t", returns a table with fields year (yyyy), month (1..12), day (1..31), hour (0..23), min (0..59), sec (0..61), wday (1..7, Sunday = 1), yday (1..366), isdst (true = daylight saving), else returns the fmt string with formatting directives beginning with '%' replaced according to *Time formatting directives* (see below). In either case a leading "!" requests UTC (Coordinated Universal Time). |
| os.difftime (t2, t1) | returns the difference between two values returned by os.time() |

## Time formatting directives (most used, portable features):

| | | | |
|---|---|---|---|
| %c | date/time (locale) | | |
| %x | date only (locale) | %X | time only (locale) |
| %y | year (nn) | %Y | year (yyyy) |
| %j | day of year (001..366) | | |
| %m | month (01..12) | | |
| %b | abbreviated month name (locale) | %B | full name of month (locale) |
| %d | day of month (01..31) | | |
| %U | week number (01..53), Sunday-based | %W | week number (01..53), Monday-based |
| %w | weekday (0..6), 0 is Sunday | | |
| %a | abbreviated weekday name (locale) | %A | full weekday name (locale) |
| %H | hour (00..23) | %I | hour (01..12) |
| %p | either AM or PM | | |
| %M | minute (00..59) | | |
| %S | second (00..61) | | |
| %Z | time zone name, if any | | |

## The stand-alone interpreter

## Command line syntax

lua [*options*] [*script* [*arguments*]]

## Options

| | |
|---|---|
| - | loads and executes script from standard input (no args allowed) |
| -e *stats* | executes the Lua statements in the literal string *stats*, can be used multiple times on the same line |
| -l *filename* | requires *filename* (loads and executes if not already done) |
| -i | enters interactive mode after loading and executing *script* |
| -v | prints version information |
| -- | stops parsing options |

## Recognized environment variables

| | |
|---|---|
| LUA_INIT | if this holds a string in the form *@filename* loads and executes *filename*, else executes the string itself |
| LUA_PATH | defines search path for Lua modules, with "?" replaced by the module name |
| LUA_CPATH | defines search path for dynamic libraries (e.g. .so or .dll files), with "?" replaced by the module name |
| _PROMPT[2] | set the prompts for interactive mode |

## Special Lua variables

| | |
|---|---|
| arg | nil if no arguments on the command line, else a table containing command line *arguments* starting from arg[1] while #arg is the number of *arguments*; arg[0] holds the script name as given on the command line; arg[-1] and lower indexes contain the fields of the command line preceding the script name. |
| _PROMPT[2] | contain the prompt for interactive mode; can be changed by assigning a new value. |

## The compiler

Command line syntax luac [options] [filenames]

Options

| | |
|---|---|
| - | compiles from standard input |
| -l | produces a listing of the compiled bytecode |
| -o *filename* | sends output to filename [default: luac.out] |
| -p | performs syntax and integrity checking only, does not output bytecode |
| -s | strips debug information; line numbers and local names are lost. |
| -v | prints version information |
| -- | stops parsing options |

Note: compiled chunks are portable between machines having the same word size.

*Lua is a language designed and implemented by Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes; for details see lua.org.*

*Drafts of this reference card (for Lua 5.0) were produced by Enrico Colombini <erix@erix.it> in 2004 and updated by Thomas Lauer <thomas.lauer@gmail.com> in 2007, 2008 and 2009. Comments, praise or blame please to the lua-l mailing list.*

*Modified by Rinstrum 2013 to for use with the M4223 Lua module*