Lua has been selected as the scripting language of choice because of its speed, compactness, ease of embedding and most of all its gentle learning curve. These characteristics allow the user to create simple scripts right through to advanced programming solutions that a computing science graduate would relish. In other words, Lua has all the depth and sophistication of a modern language, yet remains very accessible to the non-programmer.

Lua originated in Brazil and has a very active and helpful forum. While originally conceived as a scientific data description language, its greatest single application area has been computer gaming. The characteristics that make it a popular for gaming closely match those required for data acquisition - an efficient scripting machine controlling a fast acquisition engine written in "C".

Acknowledgments

Lua 5.1 Short Reference is a reformatted and updated version of Enrico Colombini’s “Lua 5.0 Short Reference (draft 2)” in which he acknowledged others “I am grateful to all people that contributed with notes and suggestions, including John Belmonte, Albert-Jan Brouwer, Tiago Dionizio, Marius Gheorghe, Asko Kauppi, Philippe Lhoste, Virgil Smith, Ando Sonenblick, Nick Trout and of course Roberto Ierusalimschy, whose ‘Lua 5.0 Reference Manual’ and ‘Programming in Lua’ have been my main sources of Lua lore”. This Lua 5.1 update further acknowledges and thanks Enrico Colombini’s for his Lua 5.0 Short Reference (draft 2), Roberto Ierusalimschy’s ‘Lua 5.1 Reference Manual’ and ‘Programming in Lua, 2nd Edition’ and more recently “Lua Programming Gems” edited by Luiz Henrique de Figueiredo et al.

This Short Reference update was initially done as a means of becoming familiar with Lua, so it has been edited, clarified and extended from the perspective of a new-comer to Lua. Thanks also to Matthias Seifert for some recent clarifying comments.

Graham Henstridge
Monday, 16 November 2009
Lua 5.1 Short Reference

Lua Core Language

Reserved words
and break do else elseif end false for function if in local nil not or repeat return then true until while
... A system variable, where A any uppercase letter.

Other reserved strings
+ - * / % ^ # == <= <=> < > = ( ) { } [ ] ; , . .. ...

Identifiers
Any string of letters, digits and underscores not starting with a digit and not a reserved word. Identifiers starting with underscore and uppercase letter are reserved.

Comments
-- Comment to end of line.
#! At start of first line for Linux executable.

Strings and escape sequences
's' or '' can be single-line, escape sequences delimiters, '[[ ... ]]' can be multi-line, escape sequences ignored. If '[s]' number of s's must balance.
\a - bell \b - backspace \f - form feed
\n - newline \r - return \t - tab
\v - vert. tab \- single quote \' - square bracket \[ - square bracket

Types
Type belongs to the value, NOT the variable:
boolean nil and false count as false, all other true including 0 and null string. Use type(x) to discover type of x.
number 64 bit IEEE floating point
dering string Can include zero, internally hashed.
table Index by numbers, strings
function Can return multiple values
thread A cooperative coroutine.
userdata C pointer to a C object. Can be assigned a metatable to allow use like a table or function
nil A special value meaning "nothing".

Operators in precedence order
( right-associative, math lib required)
not # (length) -- (unary negative)(unary positive illegal)
* / %
+ -
.. (string concatenation, right-associative)
< > <= >= ==
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 examples
a = 5 Simple assignment.
a = "hi" Variables are not typed, they can hold different types.
a, b, c = 1, 2, 3 Multiple assignment.
a, b = b, a Swap values, because right side values evaluated before assignment.
a, b = 4, 5, 6 Too many values, 6 is discarded.
a, b = "there" Too few values, nil is assigned to b.
a = nil a's prior value will be garbage collected if unreferenced elsewhere.
a = #b Size of b. If table, first index followed by nil.
a = z If z is not defined a = nil.
a = "3" + "2" Strings converted to numbers: a = 5.
a = 3 .. 2 Numbers converted to strings: a = "32".

Conditional expression results
False: false and nil values only
True: anything not false, including 0 and empty strings

Relational and boolean examples
"abc" < "abe" True: based first different character
"ab" < "abc" True: missing character is less than any

Scope, blocks and chunks
By default all variables have global scope from first use.
local Reduces scope from point of definition to end of block.
local var_name initialized to nil. Locals significantly faster to access
block Is the body of a control structure, body of a function or a chunk.
chunk A file or string of script.

Control structures
In following exp and var have local scope
if exp then block [elseif exp then block] [else block] end
if while do block end (simply a means of forcing local scope)
while exp do block end
repeat block until exp
for var = from_exp, to_exp, do block end
for var(s) in iterator do block end
break exit loop, must be last statement in block

Table constructors
New empty table assigned to t.
t = {{900} = 3, [900] = 4} Sparse array, two elements.
t = (x=5, y=10) t = (x,y) Mixed fields: t.x, t.y. t[1], t[2].
t = ("choice", "yes", "no") Nested table.
See table.insert() etc. below for additional info.

Function definition
Functions can return multiple results.
function name (args) body [return values] end
Global function.
local function name (args) body [return values] end
Function local to chunk.
f = function (args) body [return values] end
Anonymous function assigned to variable f

function (...) body [return values] end
(...) indicates variable args and ... places them in a table 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 getting extra arg self.

Function call
f (args) Simple call, returning zero or more values.
f arg Calling with a single string or table argument
f.t (args) Calling function stored in field f of table t.
tf (args) Short for tf (t, args).
argf Short for f (arg).

Metatable operations
Base library required. Metatable operations allow redefining and adding of new table behaviours.
setmetatable (t, mt) Sets mt as metatable for t, unless fs metatable has a __metatable field. Returns t
getmetatable (t) Returns __metatable field of t's metatable, or t's metatable, or nil.
rawget (t, l) Gets l of a table without invoking metathematics.
rawset (t, l, v) Sets l = v on a table without invoking metathematics.
rawequal ( t1, t2 )
   Returns boolean (t1 == t2) without invoking metamethods.

Metatable fields for tables and userdata

__add__ Sets handler h (a, b) for '+'.
__sub__ Sets handler h (a, b) for binary '-'.
__mul__ Sets handler h (a, b) for '*'.
__div__ Sets handler h (a, b) for '/'.
__pow__ Sets handler h (a, b) for '**'.
__unm__ Sets handler h (a) for unary '-'.
__concat__ Sets handler h (a, b) for '..'.
__eq__ Sets handler h (a, b) for '=='.
__le__ Sets handler h (a, b) for '<='.
__lt__ Sets handler h (a, b) for '<'.
__ge__ Sets handler h (a, b) for '>='. 
__gt__ Sets handler h (a, b) for '>'.
__index__ Sets handler h (t, k) for non-existing field access.
__newindex__ Sets handler h (t, k) for assignment to non-existing field.
__call__ Sets handler h (f, t) for function call, using the object as a function.
__tostring__ Sets handler h (a) to convert to string, e.g. for print () .
__gc__ Set finalizer h (ud) for userdata (can be set from the C side only).
__mode__ Table mode: 'k' = weak keys, 'v' = weak values, 'kv' = both.
__metatable__ Set value returned by getmetatable ( ).

The Basic Library

The Basic Library provides many standard functions and does not require a prefix as with add-on libraries.

Environment and global variables

getfenv ( f )
   If f is a function, returns its environment; if f a number, returns the environment of function at level f (1 = current [default], 0 = global); if the environment has a field __fenv, that is returned.

setfenv ( f, t )
   Sets environment for function f or function at level t (0 = current thread); Returns f or nothing if f 0; if the original environment has a field __fenv, raises an error.

G
   Variable whose value = global environment.

__VERSION__
   Variable with interpreter’s version.

Loading and executing

require ( mod )
   Loads mod  and returns final value of package.loaded [mod] or raises error. In order, checks if already loaded, for Lua module, for C library.

module ( name [ , ... ] )
   Creates a module. If a table in package.loaded[name] this is the module, else if a global table t of name, that table is the module, else creates new table t assigned to name. Initializes t._NAME to name, t, M to be t and t PACKAGE with package name. Optional functions passed to be applied over module.

dofile ( filename )
   Loads and executes the contents of filename [default: standard input]. Returns file’s returned values.

load ( function [ , name ] )
   Loads a chunk using function to get its pieces. Each function call to return a string (last = nil) that is concatenated. Returns compiled chunk as a function or nil and error message. Optional chunk name for debugging.

loadfile ( filename )
   Loads contents of filename, without executing. Returns compiled chunk as function, or nil and error message. Optional chunk name for debugging.

loadstring ( string [ , name ] )
   Returns compiled string chunk as function, or nil and error message. Sets chunk name for debugging.

loadlib ( library, func )
   Links to dynamic library (.so or .dll). Returns function name func, or nil and error message.

pcall ( function [ , args ] )
   Calls function in protected mode; returns true and results or false and error message.

xpcall ( function, handler )
   As pcall () but passes error handler instead of extra args; returns as pcall () but with the result of handler () as error message, (use debug.traceback () for extended error info).

Simple output and error feedback

print ( args )
   Prints each of passed args to stdout using tostring.

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 ( i, ... )
   For numeric index i, returns the i th argument from the ... argument list. For i = string “#” (including quotes) returns total number of arguments including nil’s.

type ( x )
   Returns type of x as string e.g. “nil”, “string”, “number”.

 tostring ( x )
   Converts x to a string, using table’s metatable’s __tostring if available.
	onumber ( 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] as separate values, where n = #t.

Iterators

 ipairs ( t )
   Returns an iterator getting index, value pairs of array t in numeric order.

pairs ( t )
   Returns an iterator getting key, value pairs of table t in no particular order.

next ( t [ , index ] )
   Returns next index-value pair (nil when finished) from index (default nil, i.e. beginning of table t).

Garbage collection

collectgarbage ( option [ , v ] )
   where option can be:
   "stop"
      Stops garbage collection.
   "restart"
      Restart garbage collection.
   "collect"
      Initiates a full garbage collection.
   "count"
      Returns total memory used.
   "step"
      Perform garbage collection step size v, returns true if it finished a cycle.
   "setpause" 
      Set pause (default 2) to v’100. Larger values is less aggressive.
   "setstepmul"
      Sets multiplier (default 2) to v’100. Controls speed of collection relative to memory allocation.

Coroutines

coroutine.create ( function )
   Creates a new coroutine with function, and returns it.

coroutine.resume ( coroutine, arg )
   Starts or continues running coroutine, passing arg s to it. Returns true (and possibly values) if coroutine calls coroutine.yield () or terminates, or returns false and error message.

coroutines.running ()
   Returns current running coroutine or nil if main thread.

coroutine.yield ( arg )
   Suspends execution of the calling coroutine (not from within C functions, metamethods or iterators), any arg s become extra return values of coroutine.resume ( ).
coroutine.status ( co )
    Returns the status of coroutine co as a string: either "running", "suspended" or "dead".

coroutine.wrap ( function )
    Creates coroutine with function as body and returns a function that acts as coroutine.resume () (without first arg and first return value, propagating errors).

Modules and the Package Library
A package is a collection of modules. A module is a library that defines a global name containing a table that contains everything the module makes available after being require()’d module ( module, ... )
    Creates module which is a table in package.loaded[module], a global table named module or a new global table is created package.path, package.cpath
    Variable used by require () for a Lua or C loader. Set at startup to environment variables LUA_PATH or LUA_CPATH. (see Path Formats below).

package.loaded
    Table of packages already loaded. Used by require ()

package.loadlib ( library, function )
    Dynamically links to library, which must include path. Looks for function and returns it, or 0 and error message.

package.preload
    A table to store loaders for specific modules (see require).

package.secall ( module )
    Sets a metatable for module with _index field referring to global environment.

Path Formats
A path is a sequence of path templates separated by semicolons. For each template, require ( filename ) will substitute each “?” by filename, in which each dot replaced by a "directory separator" ("\" in Linux); then it will try to load the resulting file name. Example:
require ( dog.cat ) with path /usr/share/lua/.lua/lua/.lua/lua will attempt to load cat.lua from /usr/share/lua/dog or lua/dog

The Table Library
Tables as arrays (lists)
table.insert ( table, [ i ], v )
    Inserts v at numerical index i [default: after the end] in table, increments table size.
table.remove ( table, [ i ] )
    Removes element at numerical index i [default: last element] from table, decrements table size, returns removed element.
table.maxn ( table )
    Returns largest positive numeric index of table. Slow.
table.max ( table )
    Returns largest positive numeric index of table. Slow.
table.sort ( table [, cf] )
    Sorts (in-place) elements from table[1] to table[#1], using compare function cf (e1, e2) [default: <]. May swap equals.
table.concat ( table [, string, [ i [, i ]]])
    Returns a single string made by concatenating table elements table[i] to table[j] (default: i=1, j = table length) separated by string (default = nil), Returns empty string if no given elements or i > j

Iterating on table contents
Use the pairs or ipairs iterators in a for loop. Example:
for k, v in pairs(table) do print (k, v) end
    will print the key (k) and value (v) of all the table’s content.

The Math Library
Basic operations
math.abs ( x )
    Returns the absolute value of x.

math.fmod ( x, y )
    Returns the remainder of x / y as a rounded-down integer, for y != 0.

math.floor ( x )
    Returns x rounded down to integer.

math.ceil ( x )
    Returns x rounded up to the nearest integer.

math.min ( args )
    Returns minimum value from args.

math.max ( args )
    Returns maximum value from args.

math.huge
    Returns largest represented number

math.modf ( x )
    Returns integer AND fractional parts of x

Exponential and logarithmic
math.sqrt ( x )
    Returns 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.

math.exp ( x )
    Returns e to the power of x, i.e. e^x.

math.log ( x )
    Returns natural logarithm of x, for x > 0.

math.log10 ( x )
    Returns base-10 log of x, for x > 0.

math.frexp ( x )
    If x = m2^n, returns m (0, 0.5-1) and integer e

math.logexp ( x, y )
    Returns x2^y with y an integer.

Trigonometrical
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 )
    Sine of angle a in radians.

math.cos ( a )
    Cosine of angle a in radians.

math.tan ( a )
    Tangent of angle a in radians.

math.asin ( x )
    Arc sine of x in radians, for x in [-1, 1].

math.acos ( x )
    Arc cosine of x in radians, for x in [-1, 1].

math.atan ( x )
    Arc tangent of x in radians.

Pseudo-random numbers
math.random ( [ n ], m )
    Pseudo-random number in range [0, 1], [1, n] or [n, m].

math.randomseed ( n )
    Sets a seed n for random sequence. Same seed, same sequence.

The String Library
Basic operations
string indices start from 1. Negative indices from end of string so -1 is last element of string. String element values 0-255.

string.len ( string )
    Returns length of string, including embedded zeros.

string.sub ( string, i, j )
    Returns substring of string from position i to j [default: -1 which is to end].

string.rep ( string, n )
    Returns a string of n concatenated copies of string.

string.upper ( string )
    Returns a copy of string converted to uppercase.

string.lower ( string )
    Returns a copy of string converted to lowercase.

string.reverse ( string )
    Returns a string that is the reverse of string.

Character codes
string.byte ( string [, i ]]]
    Numeric ascii code of character at position i [default: 1] in string, or nil if invalid i.

string.char ( args )
    Returns a string from ascii codes passed as args.

Formatting
string.format ( string [, args ]]
    Returns a copy of string where formatting directives beginning with %’ are replaced by the value of [, args]:
    % [flags] [field_width] [.precision] type
Types
%d
    Decimal integer.
%o
    Octal integer.
%x  %X  Hexadecimal integer lowercase, uppercase.
%f  Floating-point in the form [-]n.nnnn.e[+|-]n,nnn.
%e  Floating-point in exp. form [-]n.nnnn e[+|-]n,nnn, uppercase if %E.
%g  Floating-point as %e if exp. < -4 or >= precision, else as %f; uppercase if %G.
%c  Character having the code passed as integer.
%s  String with no embedded zeros.
%q  String between double quotes, with special characters escaped.
%n  Field width and precision
    Prepends sign if negative, else space.
    Prepends sign (applies to numbers).
    Anchor pattern to string start, must be first in pattern.
    The '%c' character (escaped)

Flags
-  Left-justifies, default is right-justify.
+  Prepends sign (applies to numbers).
(spaces)  Prepends sign if negative, else space.
#  Adds "0x" before %x, force decimal point; for %e, %f, leaves trailing zeros for %g.

Field width and precision
n  Puts at least n characters, pad with blanks.
0n  Puts at least n characters, left-pad with zeros.
.n  Use at least n digits for integers, rounds to n decimals for floating-point or no more than n chars. for strings.

Formatting examples
string.format("%d", 7)  7
string.format("%d", 13)  < 13>
string.format("%05d", 13)  <00013>
string.format("%s", "Lua is great!")  "Lua is great!"
string.format("%.2f", math.pi)  <3.14>
string.format("%e", math.pi)  <3.141593e+00>
string.format("%.6f", math.pi)  <3.141600>
string.format("%06.3f", math.pi)  <0003.142>
string.format("%09d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>
string.format("%0.9d", 13)  <000000013>

Finding, replacing, iterating

Function storage

The I/O Library

The I/O functions return nil and a message on failure unless otherwise stated; passing a closed file handle raises an error.

Complete I/O

file:close ()
Closes file.

file:read ( formats )
Returns a value from file for each of the passed formats: "n" reads a number, "a" reads whole file as a string from current position ("n" at end of file), "r" 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 reading line-by-line from file; the
iterator does not close the file when finished.

file:write (values)
Write each of values (strings or numbers) to file, with no
added separators. Numbers are written as text, strings can
contain binary data (may need binary mode read).

file:seek ([pl,] offset)
Sets current position in file relative to p ("set" start of file
[default], "cur" current, "end" end of file) adding offset

file:flush ()
Writes to file any data still held in memory buffers.

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 (current output file is not
closed); file can be either an open file object or a file name;
in the latter case 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)

io.read (formats)
Reads from default input file, same as file:read ()

io.lines (fn)
Opens file name fn for reading. Returns an iterator function
reading from it line-by-line. Iterator closes file when finished. If
no fn, returns iterator reading lines from default input file.

io.write (values)
Writes to the default output file, same as file:write ()

io.flush ()
Writes to default output file any data in buffers.

Standard files and utility functions

io.stdin Predefined input file object.

io.stdout Predefined output file object.

io.stderr Predefined error output file object.

io.type (x)
Returns string "file" if x is an open file, "closed file" if x is a
closed file, nil if x is not a file object.

io.tmpfile ()
Returns file object for temporary file (deleted when program
ends).

The OS Library

Many classes of this library are determined by operating system support. Unix and Unix like systems are assumed.

Date/Time

Time and date accessed via time-table tt = {year = 1970-2135 ,
month = 1-12, day = 1-31, [hour = 0-23], [min = 0-59], [sec =
0-59], [isdst = true-false]}

os.time (tt)
Returns date/time, in seconds since epoch, described by table
tt [default: current]. Hour, min, sec, isdst fields optional.

os.difftime (t2, t1)
Returns difference t2 - t1 between two os.time () values.

os.date (fmt [, , fmt])
Returns a table or string describing date/time t (that should be
a value returned by os.time), according to the format string
fmt:

- %H %d Hour (00..23), (01..12).
- %M %H Minute (00..59).
- %m Month (01..12).
- %p %m Either "am" or "pm".
- %S %S Second (00..61).
- %s %S Weekday (0..6), 0 is Sunday.
- %x %S Date only, time only.
- %y %y Year (nm), (nmm).
- %Z %Z Time zone name if any

os.clock ()
Returns the approx. CPU seconds used by program.

System interaction

os.execute (string)
Catches system shell to execute string, returning status code.

os.exit ([code])
Terminates script, returning code [default: success].

os.getenv (variable)
Returns a string with the value of the environment variable, or
nil if no variable exists.

os.setlocale (string, category)
Sets the locale described by string for category:
"all" (default), "collate", "ctype", "monetary", "numeric" or
"time". Returns name of new locale, or nil if not set.

os.remove (file)
Deletes file, or returns nil and error description.

os.rename (file1, file2)
Renames file1 to file2, or returns nil and error message.

os.tmpname ()
Returns a string usable as name for a temporary file. Subject
to name conflicts - use io.tmpfile () instead.

The Stand-alone Interpreter

Command line syntax

lua [options] [script [arguments]]

Options
- executes script from standard input, no arg allowed
-stats Executes Lua statements contained in literal string
    stats, can be used multiple times on same line.
-file Loads and executes filename if not already loaded.
-i Enters interactive mode after execution of script.
-v Prints version information.
-p Prompts the user for input.

Recognized environment variables

LUA_INIT If it contains a string in form @filename, loads and
executes filename, else executes the string itself.

Special Lua variables

arg nil if no command line arguments, else table containing
command line arguments starting from arg[1]. arg.n is
number of arguments, arg[0] script name as given on
command line and arg[-1] and lower indexes contain
fields of command line preceding script name.

The Compiler

Command line syntax

luac [options] [scripts]

Options
- Compiles from standard input.
-i Produces a listing of the compiled bytecode.
-o 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.

Compiled chunks portable on machines with same word size.
The Debug Library

The debug library functions are inefficient and should not be used in normal operation. In addition to debugging they can be useful for profiling.

**Basic functions**

debg.debug ()
Enters interactive debugging shell (type "cont" to exit); local variables cannot be accessed directly.

debg.getenv (object)
Returns the environment of object

debg.getinfo ([coroutine], function [, w])
Returns table with information for function in coroutine or for function at level function [1 = caller], or nil if invalid level.

Table keys are:

- **source**: Name of file (prefixed by '@') or string where function defined.
- **short_src**: Short version of source, up to 60 chars.
- **lindedef**: Line of source where function was defined.
- **what**: "Lua" = Lua function, "C" = C function, "main" = part of main chunk.
- **name**: Name of function, if available, or reasonable guess if possible.
- **namewhat**: Meaning of name: "global", "local", "method", "field" or "".
- **nups**: Number of upvalues of the function.
- **func**: The function itself.

Characters in string w select one or more groups of fields (default is all):
- **n**: Returns fields name and namewhat.
- **f**: Returns field func.
- **S**: Returns fields source, short_src, what and lindedef.
- **I**: Returns field currenttime.
- **u**: Returns field nup.

debg.getlocal ([coroutine], stack_level, i)
Returns name and value of local variable at index i (from 1, in order of appearance) of the function at stack_level (1 = caller) in coroutine: returns nil if i is out of range, raises error if n is out of range.

debg.gethook (coroutine)
Returns current hook function, mask and count set with debug.sethook for coroutine.

debg.getmetatable (object)
Returns metatable of object or nil if none.

debg.getregistry ()
Returns registry table that contains static library information.

debg.getupvalue (function, i)
Returns name and value of upvalue at index i (from 1, in order of appearance) of function. If i is out of range, returns nil.

debg.traceback (c) [msg]
Returns a string with traceback of call stack, prepended by msg. Coroutine c may be specified.

debg.setenv (object, i)
Sets environment of object to table t. Returns object.

debg.sethook ([coroutine], hook, mask [, nil])
For coroutine, sets function hook as hook, called for events given in mask string: "C" = function call, "r" = function return, "f" = new code line, optionally call hook (every n instructions. Event type received by hook (as first argument: "call", "return", "tail return", "line" (line number as second argument)) or "count". Use debug.getinfo (2) inside hook (for info (not for "tail_return").

Assigns value v to the local variable at index i (from 1, in order of appearance) of the function at stack_level (1 = caller); returns nil if i is out of range, raises error if n is out of range. Coroutine may be specified.

debg.setmetatable (object, table)
Sets metatable of object to table, which can be nil.

debg.setupvalue (function, i, v)
Assigns value v to upvalue at index i (from 1, in order of appearance) of function f. Returns nil if i is out of range.

Independent Libraries

Lua core is designed to be a minimalist. As such its standard libraries are basic. Independent libraries add functionality. Some useful libraries include:

**bitlib library**
A small elegant library by Reuben Thomas has a useful set of bit-wise functions. Function arguments should be integers. Non-integers can return unexpected results.

- bit.cast (a) Casts a to the internally used integer type.
- bit.bnot (a) One's complement of a.
- bit.band (w1, ...) Bitwise "and" of the w's
- bit.bor (w1, ...) Bitwise "or" of the w's
- bit.bxor (w1, ...) Bitwise "exclusive or" of the w's
- bit.lshift (a, b) a shifted left b places
- bit.rshift (a, b) a shifted logically right b places
- bit.arshift (a, b) a shifted arithmetically right b places

**lua file system library**
The lua file system library was written by Roberto Ierusalimschy, André Carrelo and Tomás Guisasolapovides. It is a convenient set of machine dependent file access functions:

- lfs.attributes ([filepath], [aname]) Returns a table with the file attributes single attribute (aname) or nil and error message. Attributes include:
  - dev: the device that the inode resides on.
  - ino: the inode number.
  - mode: string representing associated protection mode (file, directory, link, socket, named pipe, char device, block device or other)
  - nlink: number of hard links to the file
  - uid: user-id of owner
  - gid: group-id of owner
  - rdev: device type, for special file inodes.
  - access: time of last access
  - modification: time of last data modification
  - change: time of last file status change
  - size: file size, in bytes
  - blocks: block allocated for file.
  - bsize: optimal file system I/O block size.
- lfs.chdir (path)
  Change dir to path. Returns true or nil and error string.
- lfs.getcwd ()
  Current working directory string or nil and error string.
- lfs.dir (path)
  Returns iterator function that returns a string for each directory, nil at end. Raises error if path not a directory.
- lfs.lock ([filehandle, mode], [start], [length])
  Locks an open filehandle or a part of it. Mode "r" for read/shared lock or "w" for write/exclusive lock. Returns true or nil and error string.
- lfs.mkdird (dirname)
  Creates a new directory dirname. Returns true or nil and error string.
- lfs.realpath (dirname)
  Removes dirname. Returns true or nil and error string.
- lfs.mkdir (dirname)
  Removes directory. Returns true or nil and error string.
- lfs.touch (filepath [, atime, mtimel])
  Set access atime and modification mtimel times of file filepath. Times in seconds as os.date().Defaults to current time. Returns true or nil and error string.
- lfs.unlock (filehandle, [start], [length])
  Unlocks an open filehandle or a part of it. Start and length both numbers. Returns true or nil and error string.

**Examples**

- lfs.attributes("/var/spool/mail/root", "size") returns the size of "root" in bytes
- for f in lfs.dir("/tmp") do print (f) end prints all files and directories in /tmp directory