

# luerl\_new(3)

Jean Chassoul, Robert Virding

2018-2023

## Name

luerl\_new - New basic interface to the Luerl system

## Interface functions - New Version

The **Lua State** parameter is the state of a Lua VM instance. It must be created with the **luerl\_new:init()** call and be carried from one call to the next.

As it is possible in Lua to create self-referencing data structures, indeed the standard libraries have many instances of this, then using the functions which decode their return values will generate an error when they would cause an infinite loop during the decoding. An simple example is the top level table which contains a key **\_G** which references the top-level table.

Note that Lua **Chunks** (see definition below) can travel between different States. They are precompiled bits of code, independent of State. That you can ‘carry around’ this is no unique to Luerl but a low-level implementation detail of the standard Lua [language](#), for more on chunks [read](#) the official Lua 5.3 [reference manual](#).

## Spec Definitions

**Binary** means an Erlang binary string.

**Chunks** means a portion of precompiled bytecode.

**State** means a Lua State, this *is* a Lua VM instance.

**Path** means a file system path and file name.

**KeyPath** means an Erlang list of **atoms** representing nested names, e.g. [table,pack] for table.pack.

**Keys** means Lua table keys, the keys of a key-value structure.

**CompileOptions** means a list of compiler options. Currently supported options are ‘return’, which returns the errors and warnings, and ‘report’ which will log the errors and warnings.

**LuaCallReturn** = {ok, Result, State} | {lua\_error, Error, State}

This is the return value from evaluating a Lua call.

## Functions

**luerl\_new:init()** -> State.

Get a new Lua State = a fresh Lua VM instance.

**luerl\_new:gc(State)** -> State.

Runs the garbage collector on a state and returns the new state.

**luerl\_new:load(String|Binary[, CompileOptions], State)** -> {ok, Function, State} | CompileError;

Parse a Lua chunk as string or binary, and return a compiled chunk (‘form’).

**luerl\_new:loadfile(FileName[, CompileOptions], State) -> {ok, Function, State} | CompileError.**

Parse a Lua file, and return a compiled chunk ('form').

**luerl\_new:path\_loadfile([Path, ], FileName[, CompileOptions], State) -> {ok, Function, FullName, State} | {error, Reason}.**

Search Path until the file FileName is found. Parse the file and return a compiled chunk ('form'). If Path is not given then the path defined in the environment variable LUA\_LOAD\_PATH is used.

**luerl\_new:load\_module(KeyPath, ErlangModule, State) -> State.**

Load ErlangModule and install its table at KeyPath which is **NOT** encoded.

**luerl\_new:load\_module\_dec(EncodedKeyPath, ErlangModule, State) -> State.**

Load ErlangModule and install its table at KeyPath which is encoded.

**luerl\_new:do(String|Binary|Form, State) -> {ok, Result, NewState} | {lua\_error, Error, State} | CompileError.**

Evaluate a Lua expression and return its result which is **NOT** decoded, and the new Lua State.

**luerl\_new:do\_dec(String|Binary|Form, State) -> {ok, Result, NewState} | {lua\_error, Error, State} | CompileError.**

Evaluate a Lua expression and return its result which is automatically decoded, and the new Lua State.

**luerl\_new:dofile(Path, State) -> {ok, Result, NewState} | {lua\_error, Error, State} | CompileError.**

Load and execute the Lua code in the file and return its result which is **NOT** decoded, and the new Lua State. Equivalent to doing luerl:do("return dofile('FileName')").

**luerl\_new:dofile\_dec(Path[, State]) -> {ok, Result, NewState} | {lua\_error, Error, State} | CompileError.**

Load and execute the Lua code in the file and return its result which is automatically decoded, and the new Lua State.

**luerl\_new:call(FuncRef, ArgRefs, State) -> {ok, Result, State}**

**luerl\_new:call\_chunk(FuncRef, ArgRefs, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Call a compiled chunk or function. Use the call\_chunk, call has been kept for backwards compatibility.

**luerl\_new:call\_function(FuncRef, ArgRefs, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Call a function already defined in the state. Result is **NOT** decoded.

**luerl\_new:call\_function\_dec(KeyPath, Args, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Call a function already defined in the state. KeyPath is a list of keys to the function. KeyPath, Args and Result are automatically encoded/decoded.

**luerl\_new:call\_method(ObjRef, Method, ArgRefs, State) -> {ok, Result, State} |**

**{lua\_error, Error, State}.**

Call a method already defined in the state.

**luerl\_new:call\_method\_dec(KeyPath, Method, Args, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Call a method already defined in the state. KeyPath is a list of keys to the method. KeyPath, Method, Args and Result are automatically encoded/decoded.

**luerl\_new:get\_table\_keys(KeyPath, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Gets a value inside the Lua state. KeyPath and Result are **NOT** encoded/decoded.

**luerl\_new:get\_table\_keys\_dec(KeyPath, State) -> {ok, Result, State} | {lua\_error, Error, State}.**

Gets a value inside the Lua state. KeyPath and Result are automatically encoded/decoded.

**luerl\_new:set\_table\_keys(KeyPath, Value, State) -> State.**

Sets a value inside the Lua state. KeyPath and Value are **NOT** encoded.

**luerl\_new:set\_table\_keys\_dec(KeyPath, Value, State) -> State.**

Sets a value inside the Lua state. KeyPath and Value are automatically encoded.

**luerl\_new:get\_stacktrace(State) -> [{FuncName,{file,FileName},{line,Line}}].**

Return a stack trace of the current call stack in the state.

**luerl\_new:encode(Term, State) -> {LuerlTerm,State}.**

Encode the Erlang representation of a term into Luerl form updating the state when necessary.

**luerl\_new:encode\_list([Term], State) -> {[LuerlTerm],State}.**

Encode a list of Erlang term representations into a list of Luerl forms updating the state when necessary.

**luerl\_new:decode(LuerlTerm, State) -> Term.**

Decode a term in the Luerl form into its Erlang representation.

**luerl\_new:decode\_list([LuerlTerm], State) -> [Term].**

Decode a list of Luerl terms into a list of Erlang representations.