LuaCOM User Manual

(Version 1.3b2)

Vinicius Almendra Renato Cerqueira Fabio Mascarenhas

11th February 2005

1.2 Who Should Read What (or About the Manual)

This manual is mostly a reference manual. Here we document the behavior of LuaCOM in a variety of situations, some implementation decisions that affect the end-user of the library and its limitations.

Chapter 2

Tutorial

2.1 Using The LuaCOM library

LuaCOM is an add-on to the Lua language. To be used, either the binary library of LuaCOM must be linked with the host program, just like the Lua library and other add-ons, or you should load a LuaCOMdynamic library through Lua 5's require/loadlib mechanism. To use dynamic loading in Lua 4 you should implement a similar mechanism. There are different versions of the LuaCOM binary for the different versions of the Lua library, so pay attention to link the right one.

If you are linking LuaCOMto your program, the next step is to modify the source code of the host program to call LuaCOM's *and* COM initialization and termination functions, which are part of the C/C++ API. To do so, include the LuaCOM's header — `luacom.h`

```
  luacom_close(L);
  lua_close(L);
  CoUninitialize(NULL);
  return 0;
}
```

Notice that it's necessary to initialize COM before `lua_open` and to terminate it only after the last `lua_close`, otherwise fatal errors may occur.

Using Lua 5 to dynamically load LuaCOM is simpler. Just call `require("luacom")` in your Lua script, and make sure the file `luac.[(l51)]TJ/F31 10.909 T6-17422 0 Td[shein your`

type information. If the object has an associated help file, LuaCOM can launch it using the method `ShowHelp`:

```
word = luacom.CreateObject("Word.Application")
assert(word)
luacom.ShowHelp(word)
```

If the object has an associated type library, LuaCOM can generate and display an HTML file describing it. This information can also be read using other type library browsers, as OleView.

The method `DumpTypeInfo` can be used in console applications to list the methods and properties of the interface. It does not give much information, but can be helpful when playing with an object.

2.5 Methods and Properties

Standard Lua API

Method	Description
`CreateObject`	Creates a LuaCOM object.

`CrNewject`

Standard Lua API (continued)

Method	Description

Extended Lua API

Method	Description
`CreateLocalObject`	Creates a LuaCOM object as an out-of-process server.
`CreateInprocObject`	Creates a LuaCOM object as an in-process

Standard C/C++ API

Function	Description

3.3 Automation binding

The Automation binding is responsible for translating the table accesses to the LuaCOM object into COM interface calls. Besides that, it also provides a mechanism for implementing

```
end
-- property access
a = obj.TestData
-- property setting
obj.TestData = a + 1
```

Properties may also be accessed as methods. This is mandatory when dealing with parameterized properties, that it, ones that accept (or demand) parameters. A common example of this situation is

Default methods

A

Message loop To receive events, it is necessary to have a message loop in the thread that owns the object that is receiving the events. All events are dispatched through a Windows message queue created during COM initialization. Without a message loop, the event objects implemented by LuaCOM , will never receive method calls from the COM objects they are registered with. Out-of-process COM servers implemented with LuaCOM also need a message loop to be able to service method calls (one is provided by calling `luacom.DetectAutomation`).**+.3.4(e)000(P)10e50(P)10e**

com2lua situation When the called method finishes, LuaCOM translates the return value and the output values (that is, the values of the “out” and “in-out” parameters) to Lua return values. That is, the method return value is returned to the Lua code as the first return value; the output values are returned in the order they appear in the parameter list (notice that here we use the Lua feature of

is, all “in” an “in-out” COM parameters are translated to parameters to the Lua function call (the output parameters are ignored). When the call finishes, the first return value is translated as the return value of the COM method and the other return values are translated as the “in-out” and “out” values,

`abort_on_error` if false, errors inside method calls and property accesses are also ignored, possibly return

```
    window.Visible = 1
    -- this has the same result
    windows.Visible = -10
  else
    window.Visible = 0
  end
end
```

```
--
-- Sample use of enumerators
--
-- Gets an instance
word = luacom.GetObject("Word.Application")
-- Gets an enumerator for the Documents collection
docs_enum = luacom.GetEnumerator(word.Documents)
-- Prints the names of all open documents
doc = docs_enum:Next()
while doc do
  print(doc.Name)
  doc = docs_enum:Next()
end
```

The Extended Lua API method `pairs` allows the traversal of the enumeration using Lua's `for`

Chapter 4

Implementing COM objects and controls in Lua

4.1 Introduction

With LuaCOM it is possible to implement full-fledged COM objects and OLE controls using Lua. Here we understand a COM object as a composite of these parts:

- a server, which implements one or more COM objects;
- registry information, which associates a CLSID (Class ID) to a triple

4.4 Building a LuaCOM COM server

There are some steps to build a COM server using LuaCOM:

1. specify the component;
2. identify what is going to be exported: Lua application object and its sub-objects;
3. build a type library for the component;
4. define the registration information for the component;
5. register the Component object;
6. implement and expose the COM objects;

```
  reginfo.CoClass = "Test"
  reginfo.ComponentName = "Test Component"
  reginfo.Arguments = "/Automation"
  reginfo.ScriptFile = path_to_script .. "testobj.lua"
  -- stores component information in the registry
  local res = luacom.RegisterObject(reginfo)
  if res == nil then
     error("RegisterObject failed!")
  end
end

function COM:UnRegister()
  -- fills table with registration information
  local reginfo = {}
  reginfo.VersionIndependentProgID = "TEST.Test"
  reginfo.ProgID = reginfo.VersionIndependentProgID..".1"
  reginfo.TypeLib = "test.tlb"
  reginfo.CoClass = "Test"
  -- removes component information from the registry
  local res = luacom.UnRegisterObject(reginfo)
  if res == nil then
     error("UnRegisterObject failed!")
  end
end

-- Starts automation server
return luacom.DetectAutomation(COM)
```

4.8 Building a Lua OLE control

Most of what is needed to build an OLE control was already covered in the last section. Controls are like ordinary LuaCOM objects, but they are created by the

The `demo/control` directory of the LuaCOM distribution has an example of a control.

Chapter 5

Release Information

Here is provided miscellaneous information specific to the current version of LuaCOM. Here are recorded the current limitations of LuaCOM, its known bugs, the history of modifications since the former version, technical details etc.

5.1 Limitations

Here are listed the current limitations of LuaCOM, as of the current version, and information about future relaxation of this restrictions.

- LuaCOM currently supports only exposes COM objects as “single use” objects. That might be circumvented by exposing many times the same object. This restriction might be removed under request;
- LuaCOM doesn’t support COM methods with variable number of parameters. This could be circumvented passing the optional parameters inside a table, but this hasn’t been tested. This may be implemented under request;
- LuaCOM doesn’t provide access to COM interfaces that doesn’t inherit from `IDispatch` interface. That is, only Automation Objects are supported. This restriction is due to the late-

- when a table of LuaCOM objects (that is, a SAFEARRAY of `IDispatch` pointers) is passed as a parameter to a COM object, these LuaCOM objects might not be disposed automatically and may leak;
- when a COM object implemented in Lua is called from VBScript, the “in-out” parameters of type SAFEARRAY cannot be modified. If they are, VBScript will complain with a COM error.

```
Public WithEvents obj_dummy as MyCOMObject.Application
Public obj as Object
Set obj_dummy = CreateObject("MyCOMObject.Application")
Set obj = obj_dummy
```

This way the client may call methods of the COM object using the `obj` variable.

5.5 History

- implemented a log mechanism to simplify debugging;
-

Version 1.0

-

Chapter 6

Reference

6.1 The C/C++ API

luacom

Description

Sample

```
/*
 * com_object.cpp
 *
 * This sample C++ code initializes the libraries and
 * the COM engine to export a COM object implemented in Lua
 */

Ua(0*/)]T)]TJ490Td[(3s)-600(sample)-6le2.h>*/
```

```
switch(result)
{
```

returns nil.

Parameters

Parameter	Type
ProgID	String

Return Values

Return Item	Possible Values
luacom_obj	LuaCOM object nil

Sample

Sample

```
events_handler = {}
function events_handler:NewValue(new_value)
  print(new_value)
end
events_obj = luacom.Connect(luacom_obj, events_handler)
```

ImplInterface

Use

```
  implemented_obj = luacom.ImplInterface(impl_table, ProgID, interface_name)
```

Description

This method finds the type library associated with the ProgID and tries to find the type information of an interface called “interface_name”. If it does, then creates an object whose implementation is “impl_table”, that is, any method call or property access on this object is translated to calls or access on the members of the table. Then it makes a LuaCOM object for the implemented interface and returns it. If there are any problems in the process (ProgID not found, interface not found, interface isn’t a `dispinterface`), the method returns nil.

```
luacom_obj:MyMethod()
print(luacom_obj.Property)
```

```
-- these are done via Lua
myobject:MyMethod()
print(myobject.Property)
-- this call is done through COM
luacom_obj:MyMethod()
print(luacom_obj.Property)
```

Description

This method is analogous to `ImplInterface`, doing just a step further: it locates the default interface for the ProgID and uses its type information. That is, this method creates a Lua implementation of a COM object's default interface. This is useful when implementing a complete COM object in Lua. It also creates a connection point for sending events to the client application and returns it as the second return value. If there are any problems in the process (ProgID not found, default interface is not a `dispinterface`

VersionIndependentProgID

UnRegisterObject

Use

```
result = luacom.UnRegisterObject(registration_info)
```

Description

addConnection

Use

Parameters

Sample

Description

This method returns a userdata holding the `IUnknown` interface pointer to the COM object behind `luacom_obj`. It's important to notice that Lua does not duplicates userdata: many calls to `GetIUnknown` for the same LuaCOM object will return the same userdata. This means that the reference count for the `IUnknown`

Return Values

Return Item	Possible Values

Methods

GetTypeLib returns the containing type library object.

GetFuncDesc(n) returns a table describing the n-th function of the type description. This table