Table Logger Utility

Contact Us
Control4 Corporation
11734 S. Election Road
Salt Lake City, UT 84020 USA
http://www.control4.com

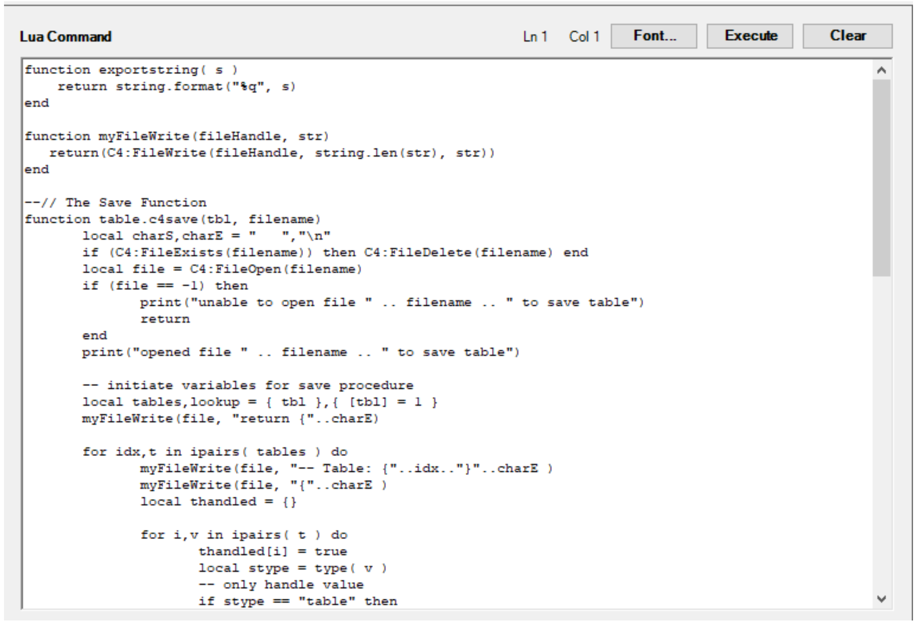
Identifying Driver Memory Leaks using the Table Logger Utility

The memory usage of a Lua driver and how it can potentially impact memory allotment is an important consideration when writing a device driver. Drivers that use large amounts of memory or grow over time can have significant, negative effects on performance within the Control4 Operating System. At times, this has been seen in Zigbee or mobile devices that do not shut down properly or in drivers that simply do not handle data correctly.

Beginning with Operating System 3.0.0, the Table Logger utility has been provided to assist driver developers in identifying areas within their driver that may be using a growing amount of memory. The utility is executed from within ComposerPro at the driver level on the Lua tab. The utility code is copied into the driver's Lua Command window. Once executed, the Lua output contains an iterated list of all Lua tables found within the driver at the time of execution. This list of tables can then be reviewed for suspect table entries or tables with anonymously added entries (using table.insert) that may be growing over time without any checks. The utility should be run several times over the course of a defined timeframe. The output should then be compared and differences examined further. Another way the utility can be used is to run it against a driver that is added to a project and not identified or connected to the final device, so the regular operation of the driver doesn’t begin. This defines a “base” state of the driver after it has initialized but before any actual interaction with the target device has occurred. The output could then be compared to another copy of the driver that has been added to the project, identified and connected to a device.

Executing the Utility

1. From within ComposerPro, open the driver's Properties/Advanced Properties screen.
2. Click on the Lua tab.
3. Copy and paste the Table Logger utility code into the Lua Command window.



```
function exportstring( s )
    return string.format("%q", s)
end

function myFileWrite(fileHandle, str)
   return(C4:FileWrite(fileHandle, string.len(str), str))
end

--// The Save Function
function table.c4save(tbl, filename)
	local charS,charE = "   ","\n"
	if (C4:FileExists(filename)) then C4:FileDelete(filename) end
	local file = C4:FileOpen(filename)
	if (file == -1) then
		print("unable to open file " .. filename .. " to save table")
		return
	end
	print("opened file " .. filename .. " to save table")

	-- initiate variables for save procedure
	local tables,lookup = { tbl },{ [tbl] = 1 }
	myFileWrite(file, "return {"..charE)

	for idx,t in ipairs( tables ) do
		myFileWrite(file, "-- Table: {"..idx.."}"..charE )
		myFileWrite(file, "{"..charE )
		local thandled = {}

		for i,v in ipairs( t ) do
			thandled[i] = true
			local stype = type( v )
			-- only handle value
			if stype == "table" then
```

4. Click on Execute.

The output of the utility will be saved to a file named with the current timestamp (e.g. 1521556723) on the controller at:

```
/control4/drivers/lua/sandbox/[driverid],
```

Where driverid is the driver ID value in ComposerPro.

The output can also be accessed on the Control4 controller using Windows network file sharing in the project at
`\\[controller_ip_address]\drivers\lua\sandbox\[driverid]`

Using the Output
The utility iterates through every table in the driver, starting with the global table _G and lists its contents. As an example (only showing the first 15 lines):

```
    1. return {
          ■  Table: {1}
    2. {
    3. ["tPictureModeCommandMap"]={2},
    4. ["gNetworkIPAddress"]="192.168.1.196",
    5. ["COMMAND_QUEUE_SIZE"]=100,
    6. ["TV_PROXY_BINDINGID"]=5001,
    7. ["sqlite3"]={3},
    8. ["JSON"]={4},
    9. ["debug"]={5}
   10.  ["C4SystemEvents"]={6},
   11.  ["socket"]={7},
   12.  ["gSendTimer"]={8},
   13.  ["CMDS"]={9},
   14.  ["TVAppDeviceID"]=1150,
   15.  ["OneShotTimer"]={10},
```

Each item listed is a key/value pair on one line.
If the value is a string, it is enclosed in quotes, as on line 4.
If the value is a number, or is a Boolean value, it is shown without quotes, as on line 5.
If the value is a table, it is listed as a number in curly braces, as on line 3. This number in curly braces is a unique index that is used later in the output to show the key/value pairs of that table. The listing for that table can be found by searching the output for the listed value (in this case, searching for {2} would find the tPictureModeCommandMap table).

It is expected that there will be many tables in a reasonable driver, and these are required for operation and are unlikely to indicate a memory leak. The most likely memory leak will be found in tables that have entries added to them with the expectation that the entry will be used later to handle a response. These entries are often added anonymously (using table.insert) rather than having a specific key. Tables of this format will appear slightly differently, and will be listed just as the values rather than as key/value pairs. Table 15 of the sample output shows this:

```
- Table: {15}
 {
    {214},
    {215},
    {216},
    {217},
    {218},
    {219},
    {220},
```

This table is comprised entirely of subtables, and each of those subtables is keyed by an increasing (not shown) integer value (this would traditionally be called an array, except Lua doesn't do arrays, only tables).

`Table: {62}` also looks suspect as a series of indexed entries. To verify what the purpose of this table is, search through the output for `{62}`.

The search locates the following: `["tInputConnMapByID"]={62},`
As the driver writer, we know that the tInputConnMapByID table is a valid table within the driver and is a map of the input connection IDs to device IDs. This table is unlikely to grow further with regular use.

Continuing to review the output, we encounter the following:

```
- Table: {34}
{
  {77}
  {78},
  {79},
  {80},
  {81},
  {82},
  {83},
  {84},
  {85},
  {86},
  {87},
  {88},
  {89},
  {90},
```

Like the previous example, this is a list of tables which have no keys. To verify what the purpose of this table is, search through the output for `{34}`.

The search locates the following: `["gGlobalTicketHandlers"]={34},`

As the driver writer, we know that the gGlobalTicketHandlers table contains the tickets with information on the URL calls that are made to the device. These tickets should be removed from this table when the response comes back from the device or the call times out. There are unlikely to be so many tickets waiting to be handled (our URL timeout is 20 seconds), so this indicates that there is likely something wrong with our URL call response handling code that should be further investigated. In this particular example, there was a specific API call that was being made that returned an error code that wasn't being handled correctly in the driver, and so the ticket info was never being removed from our gGlobalTicketHandlers table. Over time, this would lead to the driver memory usage growing unchecked until Director restarted with an Out Of Memory error.