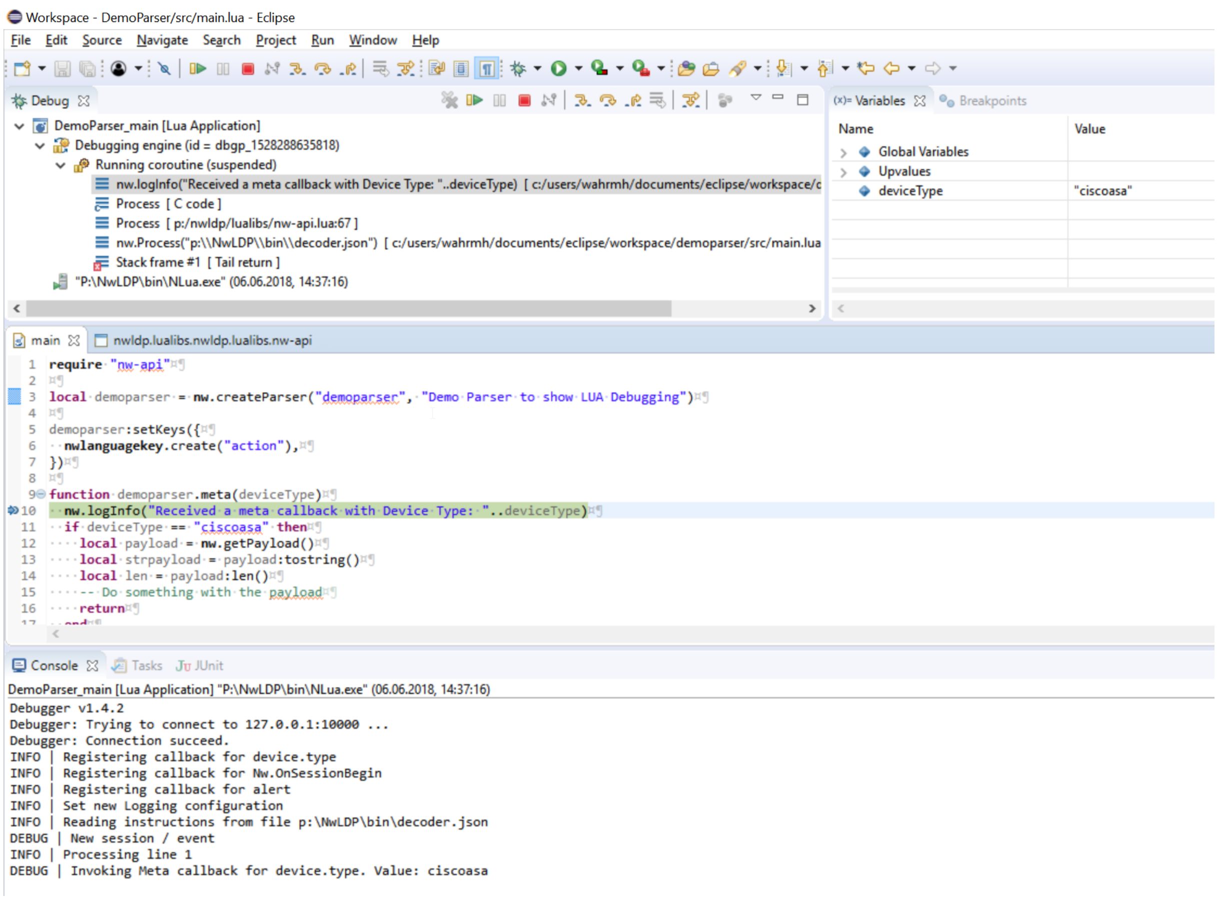
The NetWitness LUA Parser Debugger Project

supported by Eclipse and NLua

Helmut Wahrmann


Workspace - DemoParser/src/main.lua - Eclipse
File Edit Source Navigate Search Project Run Window Help
Debug
Variables
Breakpoints
main
nwldp.lualibs.nwldp.lualibs.nw-api
Console
Tasks
JUnit

```
DemoParser_main [Lua Application]
  Debugging engine (id = dbgp_1528288635818)
    Running coroutine (suspended)
      nw.logInfo("Received a meta callback with Device Type: "..deviceType) [ c:/users/wahrmh/documents/eclipse/workspace/c
      Process [ C code ]
      Process [ p:/nwldp/lualibs/nw-api.lua:67 ]
      nw.Process("p:\\NwLDP\\bin\\decoder.json") [ c:/users/wahrmh/documents/eclipse/workspace/demoparser/src/main.lua
      Stack frame #1 [ Tail return ]
  "P:\NwLDP\bin\NLua.exe" (06.06.2018, 14:37:16)
```

Name	Value
Global Variables	
Upvalues	
deviceType	"ciscoasa"

```lua
require "nw-api"

local demoparser = nw.createParser("demoparser", "Demo Parser to show LUA Debugging")

demoparser:setKeys({
  nwlanguagekey.create("action"),
})

function demoparser.meta(deviceType)
  nw.logInfo("Received a meta callback with Device Type: "..deviceType)
  if deviceType == "ciscoasa" then
    local payload = nw.getPayload()
    local strpayload = payload:tostring()
    local len = payload:len()
    -- Do something with the payload
    return
  end
```

```
DemoParser_main [Lua Application] "P:\NwLDP\bin\NLua.exe" (06.06.2018, 14:37:16)
Debugger v1.4.2
Debugger: Trying to connect to 127.0.0.1:10000 ...
Debugger: Connection succeed.
INFO | Registering callback for device.type
INFO | Registering callback for Nw.OnSessionBegin
INFO | Registering callback for alert
INFO | Set new Logging configuration
INFO | Reading instructions from file p:\NwLDP\bin\decoder.json
DEBUG | New session / event
INFO | Processing line 1
DEBUG | Invoking Meta callback for device.type. Value: ciscoasa
```

Contents

Preface

RSA NetWitness allows the writing of custom parsers. Those parsers are written in LUA (https://www.lua.org).

Bill Motley has written a Parsers Book, which explains the concepts of building a custom LUA Parser to be used inside RSA NetWitness. See here for the latest version: https://community.rsa.com/docs/DOC-41370

While it is not too complicated to write a parser, it is sometimes very hard to find logic errors in the parser. Bill has some tips in his book for syntax checking, but to work with "real" results, you need to run the parser in the live environment, which means injecting Logs or Pcaps.

And to find logic errors in the Parser, you can only use Debug output to the syslog console, which makes it very time consuming.

I was spending already too much time in POCs, where some stupid programming errors on my side led to numerous wasted hours, so I was looking for an alternative and found an Open Source Project called NLua, which allows using LUA from C#.

With that I had the ability to write a Backend Helper class in .Net, which is emulating a RSA NetWitness Decoder and that can interact with the LUA Parser, which runs inside the Eclipse IDE.

The rest of the document describes how to install the required components and finally how to debug a parser.

This project uses several Free Software and Open Source components, namely ECLIPSE IDE, ECLIPSE LUA Development Tools, NLua, Newtonsoft JSON Parser, NLog as well as LuaSocket.

With that in mind I decided to release my code also as Open Source under GPLv3 license.

Note: While I try my best to emulate a RSA NetWitness Decoder as close as possible, there might still be some differences. So if running a parser on a RSA NetWitness Decoder gives different results than in the debugging environment, it is "most probably" an error on my side. ☺

Setup

The project makes use of Eclipse and several Open Source components. This section references the needed components and how to install them.

Required Components

ECLIPSE IDE

Eclipse is used as the IDE to write / change the parser. It is also used to interact with the developer during the debugging of the project.

Eclipse is available here: https://www.eclipse.org/ide/

During the time of development ECLIPSE Oxygen.3a Release (4.7.3a) was the current version. The project will be tested against later versions once they are released.

ECLIPSE LUA Development Tools (LDT)

In order to develop LUA code in the ECLIPSE IDE, the LUA Development Tools (LDT) need to be installed. LDT is available here: http://www.eclipse.org/ldt/

Either download the standalone installer or add the LDT repository to eclipse. Both methods are described in the installation section of the above page.

After LDT has been installed, the ECLIPSE IDE contains everything for LUA Development.

NLua

NLua is an open source project, which opens LUA to the .Net world. It allows LUA programs to call methods written in .Net and .Net code is able to reference LUA functions and variables.

So this is a perfect environment to allow the LUA Parser e.g. Register Callbacks in a .Net component and the .Net component can call functions inside the parser.

NLua is distributed together with the Installer.

If someone wants to compile it from the source code, it is available here:
https://github.com/NLua/NLua

Attention! I needed to make a change to NLUA. The following patch needs to be applied to NLua before compiling it.

The patch is available on Github:
https://github.com/hwahrmann/NwLUADebugProject/blob/master/NLua/MetaTable.patch

NwLuaDebugHelper

NwLuaDebugHelper is distributed as binary installer and installs all the needed components, NLua to be able to run the parser inside ECLIPSE IDE as well as the code to emulate RSA NetWitness Decoder events. Once installed, the system is ready to debug parsers.

The latest version can be downloaded from GitHub:

https://github.com/hwahrmann/NwLUADebugProject/releases/latest

Extract and run the installer from the Zip File. Then follow the instructions for Configuration in the next section.

Configuration

After a successful installation of the required components, it is time to configure ECLIPSE to make use of the new components.

ECLIPSE LDT is installed to make use of the standard LUA interpreter. We need to use NLua as the interpreter to be able to communicate with the .Net component.

We set the interpreter in the ECLIPSE ide in **Window -> Preferences -> Lua -> Interpreters**.

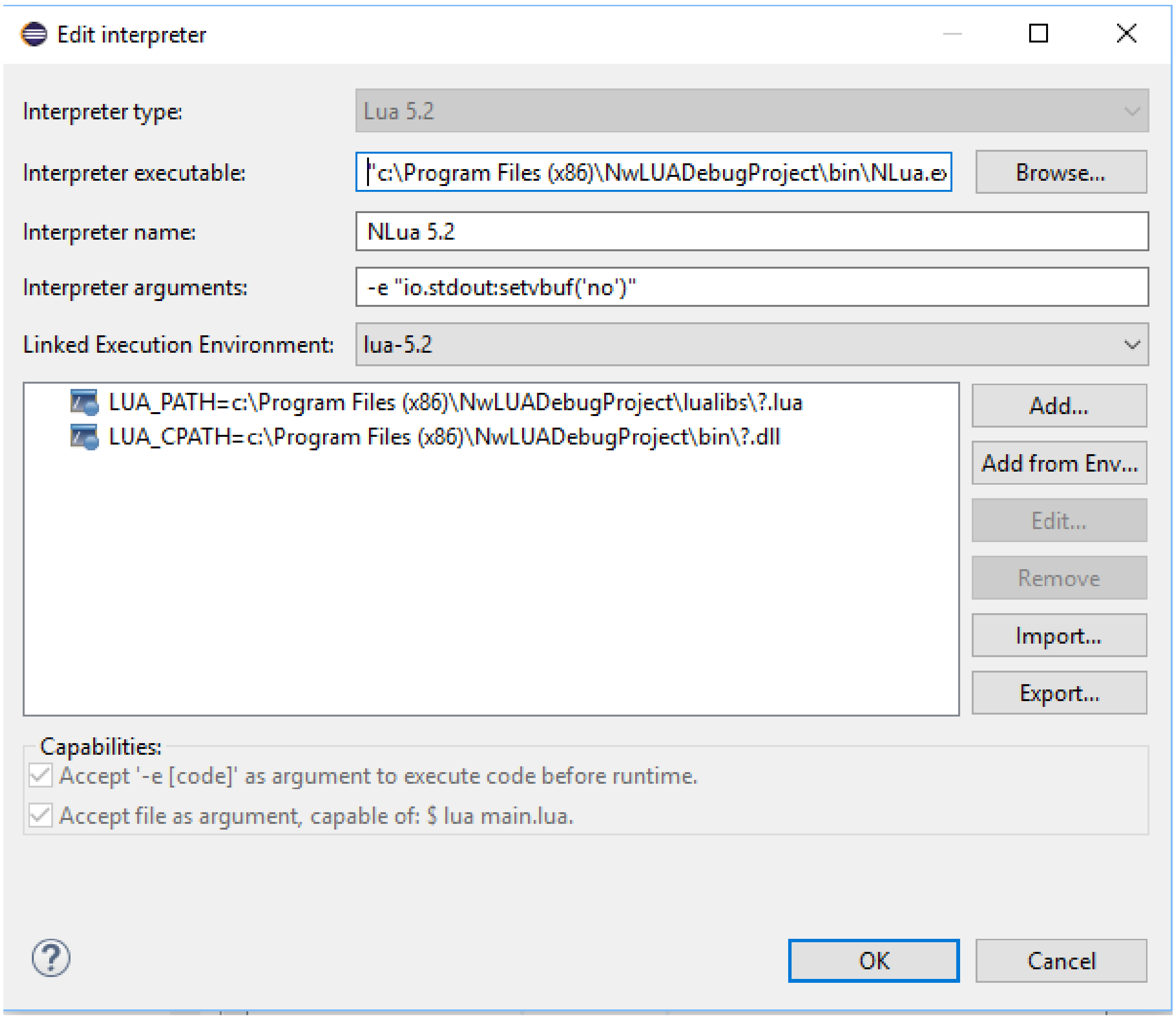
For the Interpreter type select “Lua 5.2” and set the other parameters as shown in the screenshot below:

Interpreter executable: This should reference “NLua.exe” found in the location used by the installer in the previous section.

Interpreter name: Use any name you like.

Interpreter arguments: should be set to: -e "io.stdout:setvbuf('no')"

Linked Execution Environment: select “lua-5.2” as shown in screen shot



Edit interpreter

Interpreter type: Lua 5.2

Interpreter executable: "c:\Program Files (x86)\NwLUADebugProject\bin\NLua.e

Interpreter name: NLua 5.2

Interpreter arguments: -e "io.stdout:setvbuf('no')"

Linked Execution Environment: lua-5.2

LUA_PATH=c:\Program Files (x86)\NwLUADebugProject\lualibs\?.lua
LUA_CPATH=c:\Program Files (x86)\NwLUADebugProject\bin\?.dll

Capabilities:
[✓] Accept '-e [code]' as argument to execute code before runtime.
[✓] Accept file as argument, capable of: $ lua main.lua.

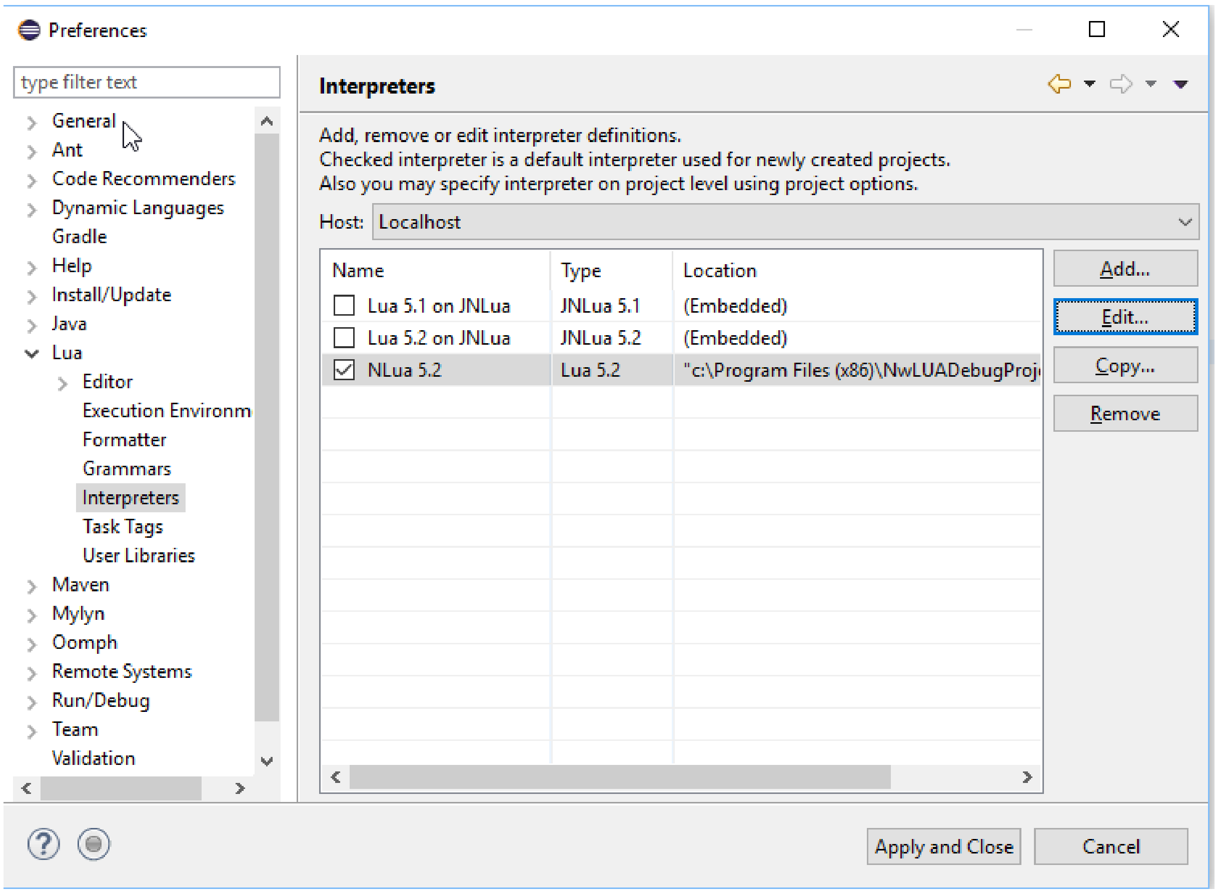
Finally it important to set 2 environment variables, so that the LUA interpreter finds LUA files and DLLs, which are stored outside of our Parser project. They should point to the “lualibs” and “bin” subfolders of the Netwitness LUA Debug Helper.

If the default installation path has been used, they should be set to:

LUA_PATH	c:\Program Files (x86)\NwLUADebugProject\lualibs\?.lua
LUA_CPATH	c:\Program Files (x86)\NwLUADebugProject\bin\?.dll

Please adjust them accordingly to your installation.

Once you are done with the above, you have a LUA environment, which is ready to run. Make sure that you select your newly defined configuration as the default:



Interpreters

Add, remove or edit interpreter definitions.
Checked interpreter is a default interpreter used for newly created projects.
Also you may specify interpreter on project level using project options.

Host: Localhost

Name	Type	Location
☐ Lua 5.1 on JNLua	JNLua 5.1	(Embedded)
☐ Lua 5.2 on JNLua	JNLua 5.2	(Embedded)
☑ NLua 5.2	Lua 5.2	"c:\Program Files (x86)\NwLUADebugProj

Debugging a RSA NetWitness Parser

The first version supports callbacks of all NetWitness Events like session begin, session end, etc. It also supports Session callbacks for Meta Creation as well as Tokens.

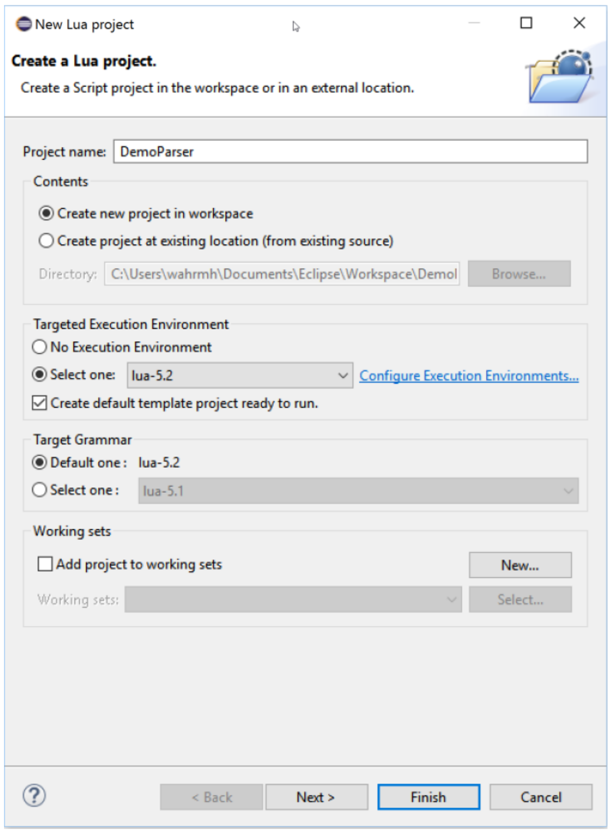
The input is delivered via a File, which is in JSON format. I am still trying to figure out, how I can deal with sessionized Payload, like it comes from a RSA NetwitNess Packet Decoder.

Therefor the first version addresses mainly Payload as it comes from a RSA NetWitness Log Decoder.

Creating the ECLIPSE Project

Debugging a parser requires an ECLIPSE LUA Project, which you create using **File -> New -> Project**.

Then select "LUA Project" and fill in the required information as shown below.



New Lua project

Create a Lua project.

Create a Script project in the workspace or in an external location.

Project name: DemoParser

Contents

[✓ Create new project in workspace]

Create project at existing location (from existing source)

Directory: C:\Users\wahrmh\Documents\Eclipse\Workspace\DemoI

Targeted Execution Environment

No Execution Environment

[✓ Select one:] lua-5.2 Configure Execution Environments...

[✓ Create default template project ready to run.]

Target Grammar

[✓ Default one :] lua-5.2

Select one : lua-5.1

Working sets

Add project to working sets

Working sets:

This creates a LUA Project in your ECLIPSE work space with main.lua open in your editor. Erase the offered content. You won't need it anymore.

Interfacing with the Debug Helper

Your parser interfaces with the .Net component, which simulates the RSA NetWitness Decoder. The first statement in the parser MUST be a reference to the API:

```
require "nw-api"
```

This reference needs to be removed, once you have finished testing and put the parser into the live RSA NetWitness environment. The nw-api.lua has been installed as part of the package and is available in the lualibs folder. There should be no need to change the API.

After the reference to the API, your parser code follows.

In order to start the processing and simulate RSA NetWitness Decoder callbacks, you need to insert a call to “nw.Process” at the very end of your parser:

```
nw.Process(“<location of input file>”)
```

For example:

```
nw.Process(“c:\\NwLDP\\bin\\decoder.json")
```

The input file needs to be in JSON Format and has the following Format:

```
{
 "Meta": {"device.type":"ciscoasa","alert":"Test Alert"},
 "Payload":"Jun 06 2018 12:00:23 ASA5520-out-ServCenter : %ASA-7-609002: Teardown local-host inside:192.168.31.253 duration 0:00:03"
}
{
 "Meta": {"device.type":"ciscoasa"},
 "Payload":"Jun 06 2018 12:00:23 ASA5520-out-ServCenter : %ASA-7-710005: UDP request discarded from 192.168.31.101/138 to inside:192.168.31.255/138"
}
```

In the above Sample, we have the payload in “**Payload**”. In this case it is a Event, which is processed by the Log Decoder.

In “**Meta**” we specify, which Meta keys and their values the RSA NetWitness Decoder would have extracted out of the payload.

For the first record we have a Meta key of “device.type” with a value of “ciscoasa” and Meta “alert” with a value of “Test Alert”.

The second record only has a Meta key for “device.type” defined.

If our parser has registered a callback like this:

```
[nwlanguagekey.create("device.type")] = demoparser.meta
```

This would call the function “meta” in our demoparser. The NLua interpreter, which is executing inside the ECLIPSE IDE is supportzing us with debugging. We can set breakpoints inside the function and when receiving the callback, we can examine variables and also step through the statements to find out problems in the logic.

Logging

The system writes debug information into a log file. The default logging level is set to “Debug” and the log file is named “NwLuaDebugHelper.log” and resides in the install folder.

As you would need admin privileges for writing into the Program Files folder, it is best practice to set the Log file location to a different foder. For this, please put a call before the “nw.Process”. It should be in the form of:

```
nw.SetLogger("<logLevel>", "<Path to logfile>")
```

For example:

```
nw.SetLogger("Debug", "c:\\NwLDP\\logs\\log.txt")
```

Valid Log Levels are: Trace, Debug, Info, Warn, Error and Fatal

Besides logging into a file, the output of the logger is also shown in the Console Window of the ECLIPSE IDE.

The system displays various information, like registering of Callbacks and invoking the callback function.

If the parser contains statements, like nw.logInfo or nw.logDebug, those lines will appear as well.

Debugging a Demo Parser

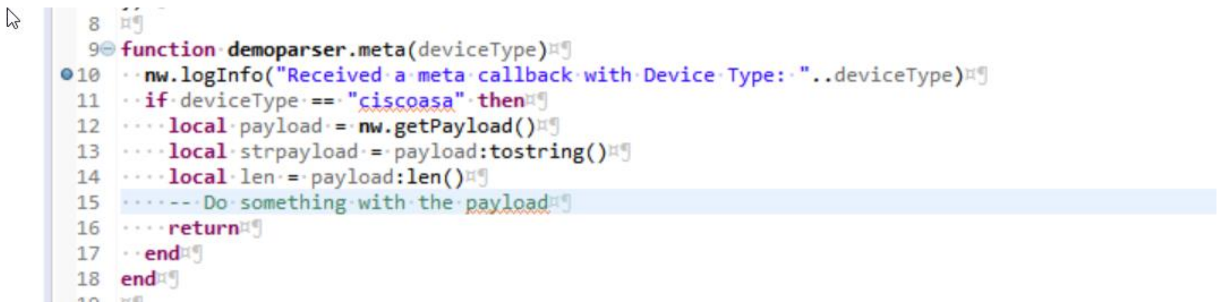
So let’s crate a sample parser with a name of Demoparser, by creating a LUA Project and specifying the following in main.lua:

```
require "nw-api"

demoparser = nw.createParser("demoparser", "Demo Parser to show LUA Debugging")

demoparser:setKeys({
  nwlanguagekey.create("action"),
})

function demoparser.meta(deviceType)
  nw.logInfo("Received a meta callback with Device Type: "..deviceType)
  if deviceType == "ciscoasa" then
    local payload = nw.getPayload()
    local strpayload = payload:tostring()
    local len = payload:len()
    -- Do something with the payload
    return
  end
end

function demoparser.alert(alert)
    nw.logInfo("Received a meta callback with Alert: "..alert)
end

function demoparser.onbegin()
    nw.logDebug("New session / event")
end

function demoparser.tokenmatch(token, first, last)
   nw.logInfo("Token match: "..token.." at position "..first.." ending at "..last)
end

demoparser:setCallbacks({
    [nwevents.OnSessionBegin] = demoparser.onbegin,
    [nwlanguagekey.create("device.type")] = demoparser.meta,
    [nwlanguagekey.create("alert")] = demoparser.alert,
    ["UDP"]  = demoparser.tokenmatch,
    ["^Jun"]  = demoparser.tokenmatch,
    ["138$"]  = demoparser.tokenmatch
})

nw.SetLogger("Debug", "c:\\Users\\wahrmh\\Downloads\\log.txt")
nw.Process("p:\\NwLDP\\bin\\decoder.json")
```

WARNING: in Netwitness you need to code the function with a colon, “:”, as separator. In the current version I still have a problem with that, so you need to have the function separated with dots. This will be fixed soon. Hopefully.

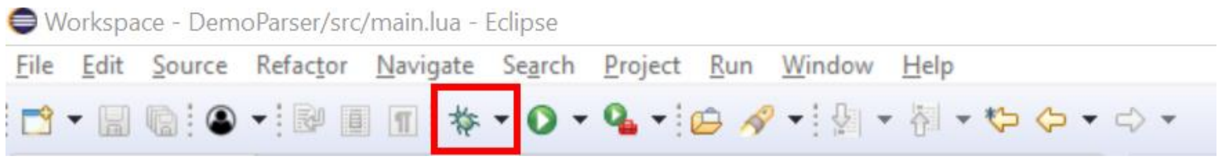
Remember to specify “nw-api” as the first line of the parser and set the location of the log file as well as calling the Process loop at the very end of the file.

Now let’s set a breakpoint in the function for the meta callback. In the IDE we click outside the iine numbers on line #10 and have now a breakpoint:

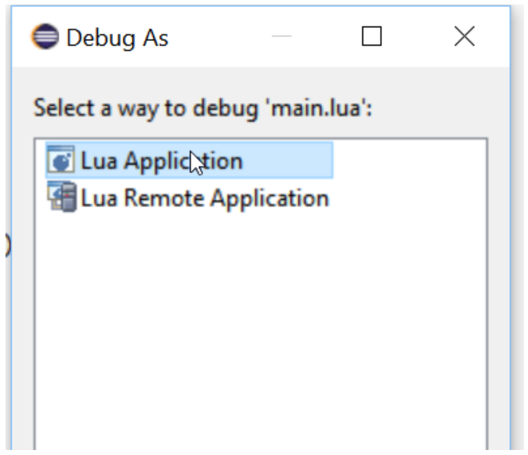


```
 8
 9 function demoparser.meta(deviceType)
10   nw.logInfo("Received a meta callback with Device Type: "..deviceType)
11   if deviceType == "ciscoasa" then
12     local payload = nw.getPayload()
13     local strpayload = payload:tostring()
14     local len = payload:len()
15     -- Do something with the payload
16     return
17   end
18 end
```

Now start debugging, by clicking on the “bug” symbol in the toolbar:


Workspace - DemoParser/src/main.lua - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Starting a debug session for the first time for a project requires you to specify the type of project.



Debug As

Select a way to debug 'main.lua':

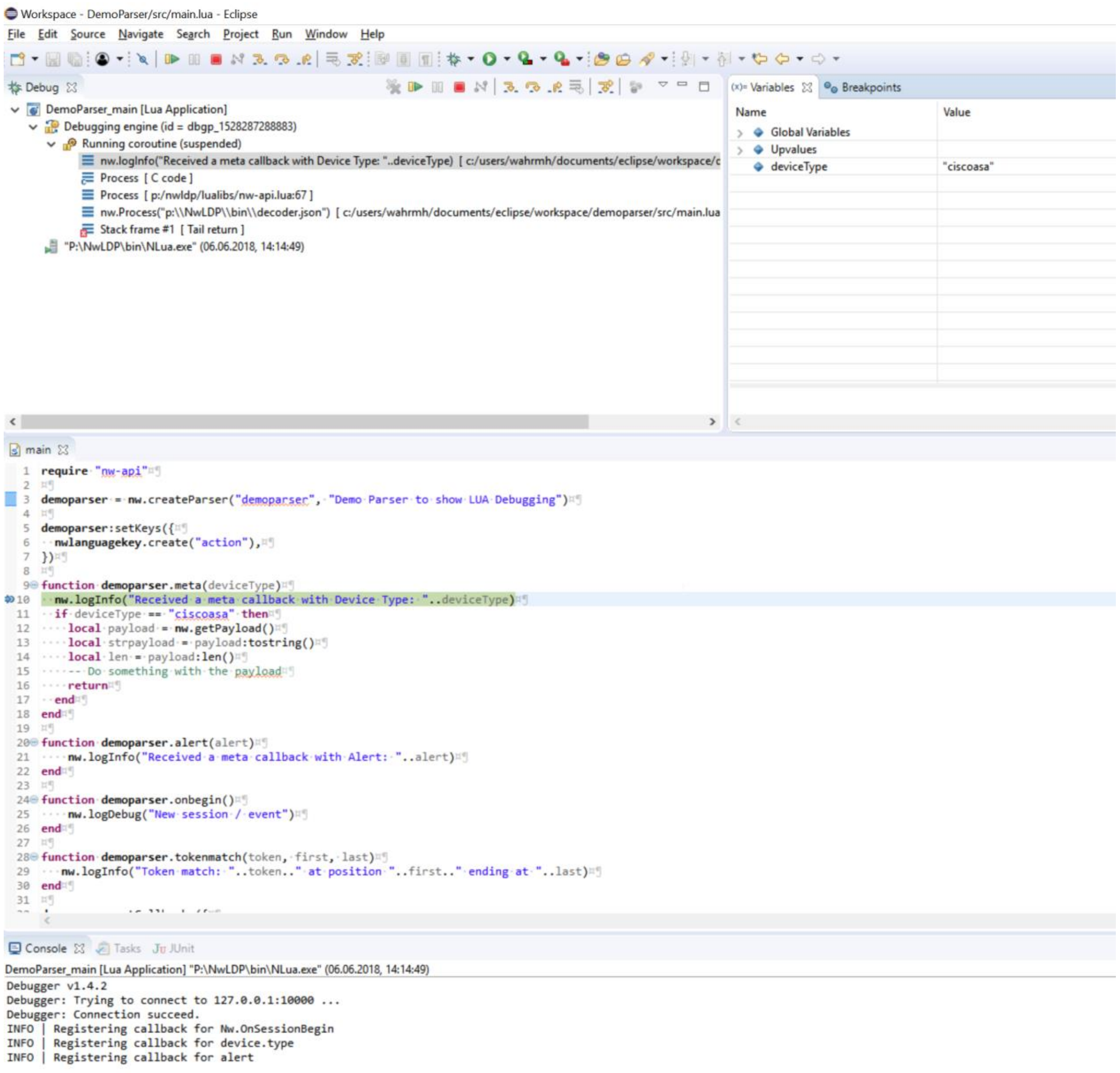
Lua Application

Lua Remote Application

Select “Lua Application”.

If your input file is ok and you followed the install instruction, the debugging session is started and the breakpoint in line #10 is hit.

The IDE switched to the debug Layout and you are now able to examine the various values.



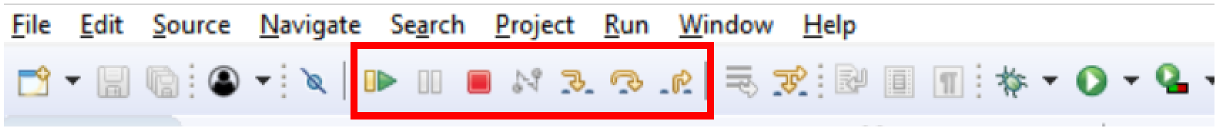
```
DemoParser_main [Lua Application]
  Debugging engine (id = dbgp_1528287288883)
    Running coroutine (suspended)
      nw.logInfo("Received a meta callback with Device Type: "..deviceType)  [ c:/users/wahrmh/documents/eclipse/workspace/c
      Process  [ C code ]
      Process  [ p:/nwldp/lualibs/nw-api.lua:67 ]
      nw.Process("p:\\NwLDP\\bin\\decoder.json")  [ c:/users/wahrmh/documents/eclipse/workspace/demoparser/src/main.lua
      Stack frame #1  [ Tail return ]
  "P:\NwLDP\bin\NLua.exe" (06.06.2018, 14:14:49)
```

```lua
require "nw-api"

demoparser = nw.createParser("demoparser", "Demo Parser to show LUA Debugging")

demoparser:setKeys({
  nwlanguagekey.create("action"),
})

function demoparser.meta(deviceType)
  nw.logInfo("Received a meta callback with Device Type: "..deviceType)
  if deviceType == "ciscoasa" then
    local payload = nw.getPayload()
    local strpayload = payload:tostring()
    local len = payload:len()
    -- Do something with the payload
    return
  end
end

function demoparser.alert(alert)
    nw.logInfo("Received a meta callback with Alert: "..alert)
end

function demoparser.onbegin()
    nw.logDebug("New session / event")
end

function demoparser.tokenmatch(token, first, last)
   nw.logInfo("Token match: "..token.." at position "..first.." ending at "..last)
end
```

```
DemoParser_main [Lua Application] "P:\NwLDP\bin\NLua.exe" (06.06.2018, 14:14:49)
Debugger v1.4.2
Debugger: Trying to connect to 127.0.0.1:10000 ...
Debugger: Connection succeed.
INFO | Registering callback for Nw.OnSessionBegin
INFO | Registering callback for device.type
INFO | Registering callback for alert
```

You notice that "device.type" has a value of "ciscoasa", as seen on top right of the screenshot and you will also see debugging output in the Console Window, shown at the bottom. The same output is also written to the log file.

Now you can use the buttons in the toolbar, or the assigned keys, to step through the code, or resume execution:


File Edit Source Navigate Search Project Run Window Help