Getting Started with Lua

Netburner, Inc.

November 12, 2012

Contents

Introduction 2

The Lua Language 2

Language References 2

NetBurner Implementation 2

Building your first Lua app - NBEclipse 3

Building your first Lua app – Make 4

Complua Reference 5

Introduction

This document will explain how to go from a fresh install to building a project that uses the Lua library, both using NBEclipse and on the command line. It also contains more general information regarding the Lua language, idiosyncrasies of the NetBurner implementation of the interpreter, and references for where to learn more.

The Lua Language

To borrow the words directly from the Lua project:

> *Lua is a powerful, fast, lightweight, embeddable scripting language.*
>
> *Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.*

Lua is a table-based object-oriented language, based around a central environment table. (The Wikipedia page makes for a good overview)

Language References

Lua Tutorial

Lua Language Official Site

Lua 5.1 Reference Manual

NetBurner Implementation

The NetBurner implementation of the Lua interpreter is based on the official 5.1.4 version of Lua (with some configuration settings coming from the eLua (Embedded Lua) project). There are some key things to note with using the interpreter in an embedded environment:

- The interpreter does dynamic memory allocation. It does not currently have a soft limit (though this is well within the realm of reason to add).

- All numbers are the same data type. The type is set by a compile time flag. It is set by default to double
- Strings are internally cstrings: memory efficient, but resizing, concatination, etc. are expensive per usual.
- Lua does not do arrays in the C/C++ sense. It is possible to use tables as arrays in Lua, but they do not store their data in contiguous memory like arrays. When used this way, elements can be accessed with standard array notation (array[i]). **NOTE: Lua array notation indexes from 1!**

The NetBurner implementation also has a slightly odd method for handling finding and opening files. As the NetBurner library does not currently support "mounting" disks and drives at specific file paths, it was decided to search all potential drives when attempting to open a file. The order of drive in which Lua searches is configurable via an array in either the library or the app. The interpreter is, as a consequence, tightly coupled with the filesystem, requiring the user to configure predef macros for which filesystems to support and recompiling after configuring it.

Technical Specifications:

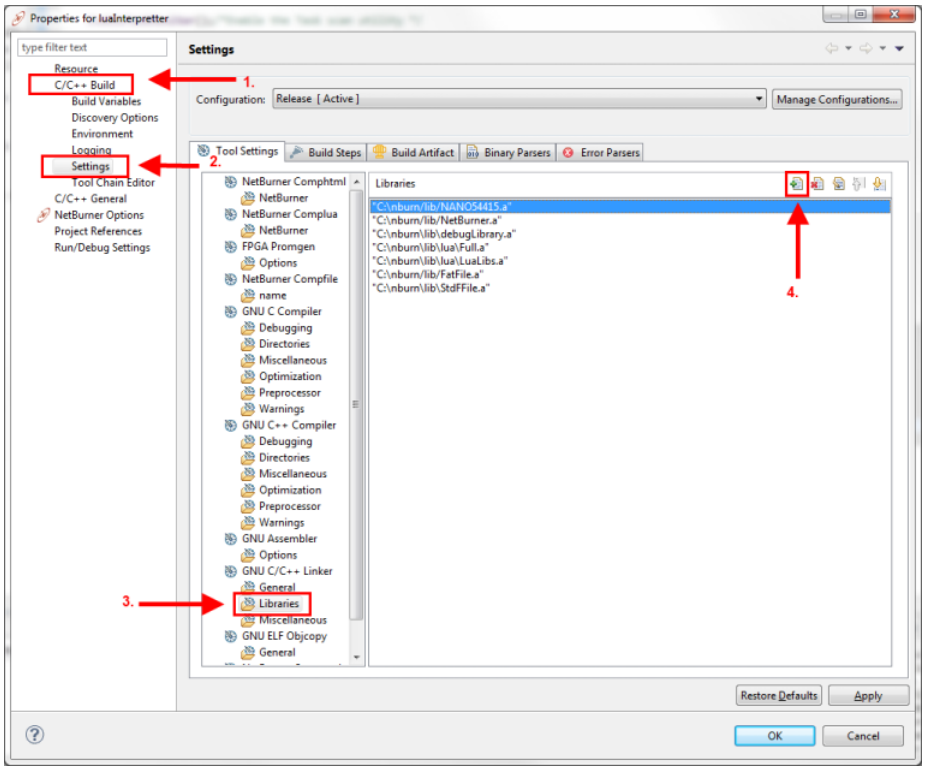
- Flash usage: ∼200k
- RAM usage: ∼200k + 20k per instance of the interpreter

Note: While this is small enough to fit on devices with 512KB of flash, when combined with the main Netburner libraries, this will consume roughly 50% of available flash before any additional application code (C/C++ or Lua) has been added.

Building your first Lua app - NBEclipse

After installing the NNDK, you will first want to create a new project (see the NBEclipse getting started guide for this). We will use this project as the basis for building one of the example Lua apps. After creating the project, you will want to import one of the Lua example applications. These are located in the 'examples/lua/' directory of your main nburn installation (usually C:, meaning C: for the examples directory). If you are prompted to overwrite "main.cpp", choose "yes".

Next comes configuring the libraries NBEclipse will link to during the project's build. Go to the project's properties menu (right clicking on the project in the project explorer and selecting "properties" or selecting it and hitting alt+enter). Navigate to the C/C++ Build->Settings page of the menu.



You should see a page with a tab labeled "Tool Settings" containing a list with subelements. Select the list subelement "Libraries" under the element "GNU C/C++ Linker". The right side of the tab will then show a list of the currently used libraries. At this point, click on the document icon with a green "+" on it to add a library. At this stage you will need to add, at the very least, the following libraries (located in the "lib" directory of your install):

- lua/Full.a
- lua/LuaLibs.a, if you imported the ShellApp example (this contains the current Lua libraries for interfacing with the main NetBurner libraries)
- FatFile.a and/or StdFile.a, if you would like to use the filesystem

Just hit the "OK" button to save the changes and build the project.

Building your first Lua app – Make

You can simply copy one of the Lua examples mentioned in the previous section into a new directory and run '`C:\projectdir> make`'. If you would like to compile in Lua scripts into the application in the same manner that web resources are compiled in with comphtml, include the following lines to your makefile:

```
CXXSRCS += luadata.cpp
CREATEDARGS := luadata.cpp
luadata.cpp: $(wildcard lua/*.lua)
    "$(LUA_DEV)/lua.exe" "$(NBROOT)/pcbin/complua/complua.lua" -i lua -o luadata.cpp
```

Complua Reference

Full usage information for complua is:

```
complua:
    Usage: complua [-h | --usage] [-i <inpath>] [-f <infiles>]
            [-o <outfilename>] [-c <path_to_lua_compiler>] [--rawmode]

    -h or --usage: print this help
    -i: sets the 'include path' where complualooks for lua source files
    -f: comma separated list of source files to process.
            Used in addition to the include path
    -o: sets the path/filename of the output file. Defaults to 'luadata.cpp'

    -c: sets the compiler to use for compiling lua source.
            Defaults to 'C:\Program Files (x86)\Lua\5.1\luac.exe'
    --rawmode: puts complua in rawmode. Rawmode will bypass compiling
            the lua source into bytecode. This will require the parser
            to be included on the device to run the code.
```