Scripting in Embedded

Introducing Lua Programming Language

Powering the Supply Chain.SM

Wait a minute…

Scripts in embedded targets?

Have you lost your marbles?

General-purpose Dynamic Languages (GDL)

- Some languages, such as Perl, began as scripting languages but were developed into programming languages suitable for broader purposes
- Other similar languages (interpreted, memory-managed, or dynamic) have been described as "scripting languages"
- They are usually not called "scripting languages" by their own users

What is GDL?

- Modern GDL (Python, Perl, Tcl, Lua, Ruby etc) languages are now supersets of C/C++/Java
- All modern programming paradigms apply; OO, FP etc
- A GDL typically have a 'parser' build into the VM, eliminating the need of compilation. You run the source directly
- Many modern GDLs have powerful debuggers

Why GDL?

- Very flexible, no compilation tools required
- GDL programs live in a controlled VM, no memory leaks, no wild pointer bugs etc
- GDL have powerful data types and libraries which makes software development much more time efficient
- GDL hook into C/C++ code easily making them "embeddable"

Why GDL?

It's the future of application development!

- More and more common on server/desktop applications. Will trickle through to embedded, it always does

GDL in embedded

- What's the drawbacks for embedded systems?
  - Speed (code interpreted)
  - Size (the VM etc takes space)
  - Requires a OS infrastructure to live in
  - Memory usage (big scripts with much data usually requires big stacks/heaps)
    - Very dependent on the code
  - Non-deterministic due to garbage collection
    - GC controllable in many GDLs

GDL in embedded

Time to call in the Myth Busters!

GDL speed

- Myth confirmed; there are a significant speed penalty of running GDL applications compared to compiled ones
- HOWEVER:
  - Speed depends on GDL variant
  - As with Java/C# there are JITs
  - How much of your application needs to run very fast? Remember that GDL hooks into C/C++ beautifully

GDL size

- Were talking MBs of bloat right?
- Myth Busted!
- Some Lua (arm 32 bit) code sizes
  - Minimal (no parser): 114KiB
  - Full (with all libs): 214KiB
- For a full GDL! That's **insane**!

Requires an OS

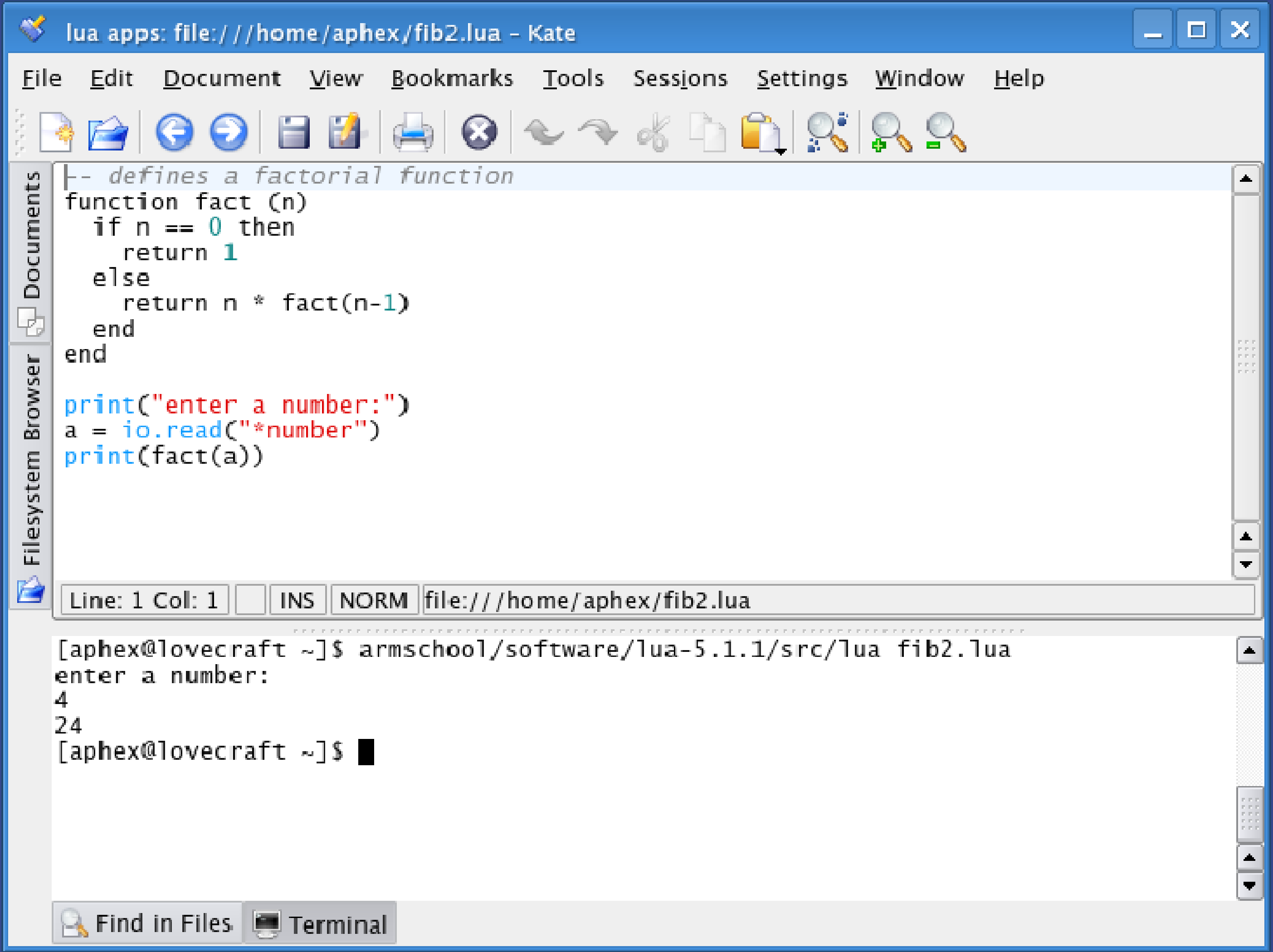
- Myth Totally Busted!
- A Lua “bare metal” example is provided with the Atmosfire kit
- Lua sits directly ontop of newlib
- A handful of platform dependent functions needed to make newlib's printf/malloc etc work; see syscalls.c
- That's **insane**!

Lua Programming Language

Lua

- Lua is a proven and robust language
- Lua is fast
- Lua is portable
- Lua is embeddable
  - C/C++/Java/C#/Smalltalk/Fortran/Ada, even other GDLs like Python, Perl and Ruby
- Lua is simple and powerful
- Lua is **free**

Lua is beautiful



lua apps: file:///home/aphex/fib2.lua - Kate
File Edit Document View Bookmarks Tools Sessions Settings Window Help
Documents
Filesystem Browser
Line: 1 Col: 1
INS
NORM
file:///home/aphex/fib2.lua
Find in Files
Terminal

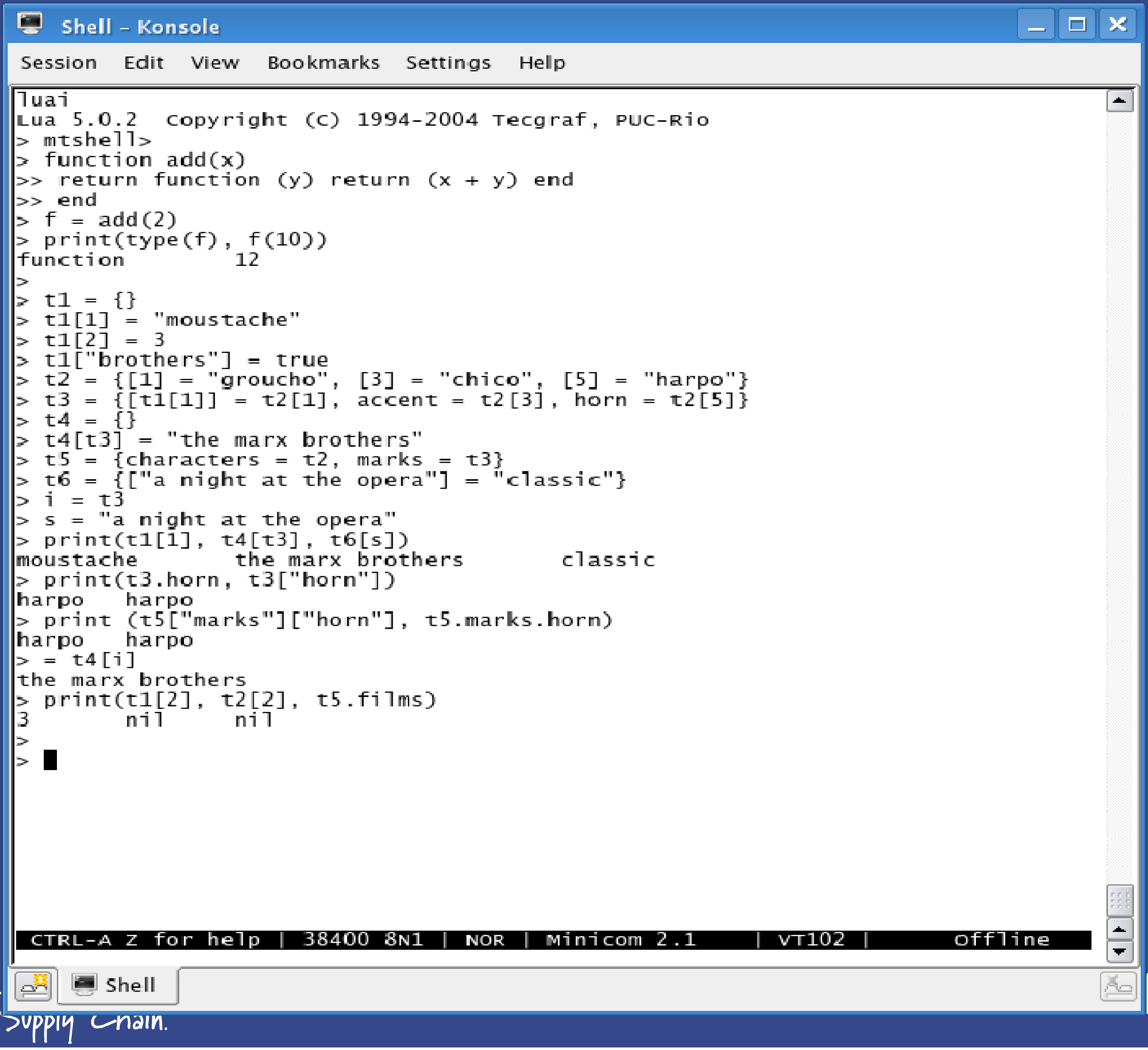
```lua
-- defines a factorial function
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end

print("enter a number:")
a = io.read("*number")
print(fact(a))
```

```
[aphex@lovecraft ~]$ armschool/software/lua-5.1.1/src/lua fib2.lua
enter a number:
4
24
[aphex@lovecraft ~]$ 
```

Powering the Supply Chain.SM

Lua is dynamic



```
luai
Lua 5.0.2  Copyright (C) 1994-2004 Tecgraf, PUC-Rio
> mtshell>
> function add(x)
>> return function (y) return (x + y) end
>> end
> f = add(2)
> print(type(f), f(10))
function        12
>
> t1 = {}
> t1[1] = "moustache"
> t1[2] = 3
> t1["brothers"] = true
> t2 = {[1] = "groucho", [3] = "chico", [5] = "harpo"}
> t3 = {[t1[1]] = t2[1], accent = t2[3], horn = t2[5]}
> t4 = {}
> t4[t3] = "the marx brothers"
> t5 = {characters = t2, marks = t3}
> t6 = {["a night at the opera"] = "classic"}
> i = t3
> s = "a night at the opera"
> print(t1[1], t4[t3], t6[s])
moustache       the marx brothers       classic
> print(t3.horn, t3["horn"])
harpo   harpo
> print (t5["marks"]["horn"], t5.marks.horn)
harpo   harpo
> = t4[i]
the marx brothers
> print(t1[2], t2[2], t5.films)
3       nil     nil
>
> 
 CTRL-A Z for help | 38400 8N1 | NOR | Minicom 2.1    | VT102 |      Offline
```

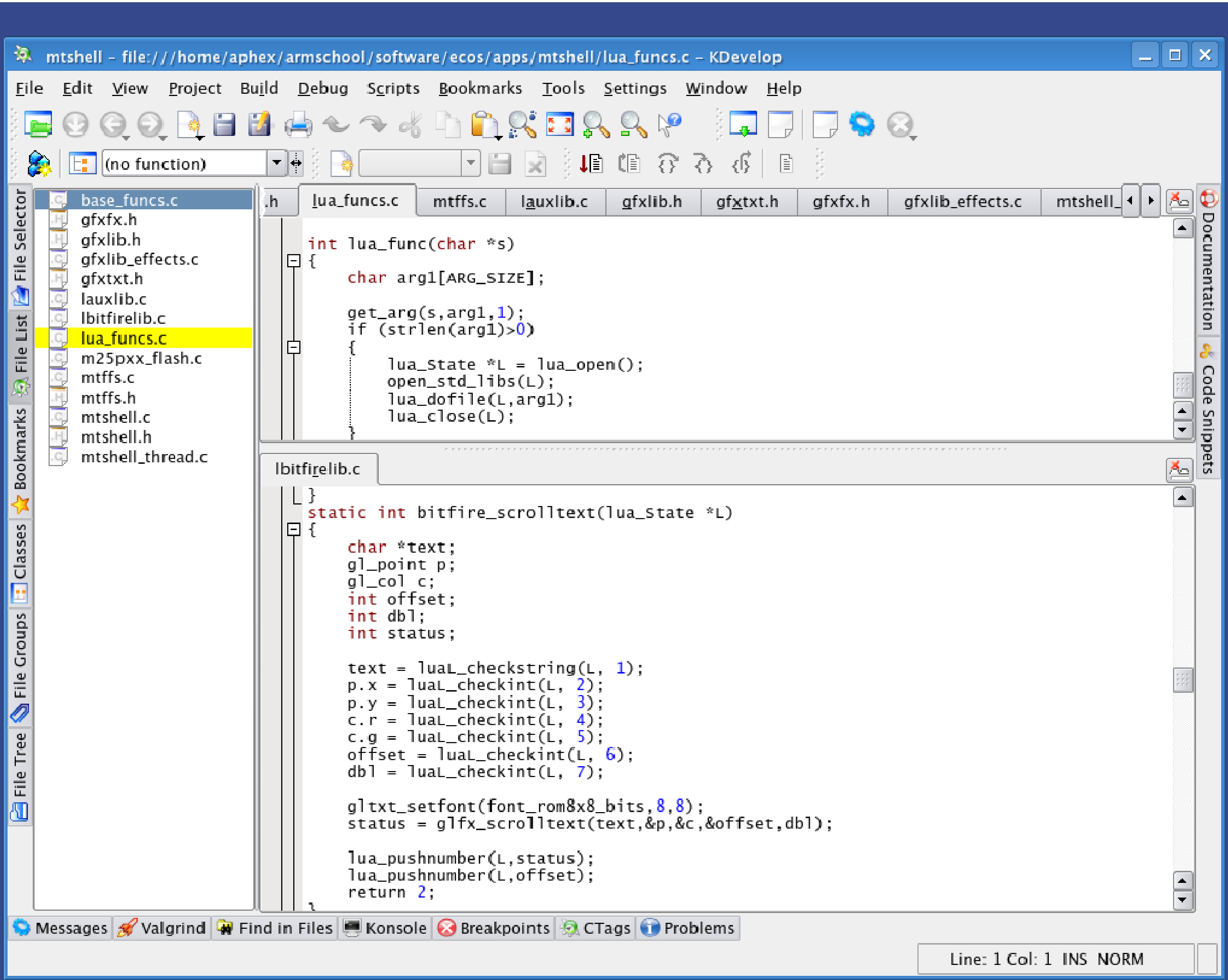
Lua is out of this world

- Lua is an incredibly easy language to pick up, but its simple syntax disguises its power.
- C API allows great integration and extension between scripts and the host language

And it’s from Brazil!

Luac compiler

- If you compile Lua without the parser, you have to run pre-parsed scripts.
- Use “luac” to created pre-parsed bytecode (very much like javac).
- This is also obscures your source if you're worried of safety.



```c
int lua_func(char *s)
{
    char arg1[ARG_SIZE];

    get_arg(s,arg1,1);
    if (strlen(arg1)>0)
    {
        lua_State *L = lua_open();
        open_std_libs(L);
        lua_dofile(L,arg1);
        lua_close(L);
    }
```

```c
}
static int bitfire_scrolltext(lua_State *L)
{
    char *text;
    gl_point p;
    gl_col c;
    int offset;
    int dbl;
    int status;

    text = luaL_checkstring(L, 1);
    p.x = luaL_checkint(L, 2);
    p.y = luaL_checkint(L, 3);
    c.r = luaL_checkint(L, 4);
    c.g = luaL_checkint(L, 5);
    offset = luaL_checkint(L, 6);
    dbl = luaL_checkint(L, 7);

    gltxt_setfont(font_rom8x8_bits,8,8);
    status = glfx_scrolltext(text,&p,&c,&offset,dbl);

    lua_pushnumber(L,status);
    lua_pushnumber(L,offset);
    return 2;
}
```

Lua

- Many standard libs for table/math/os/thread etc functions.
- Documentation and Tutorials are on your CD.
- Labs available if you're interested

You should be! :-)

Lua in your system

- The “monolith” is sill in C.
  - Make generic APIs and Lua libs.
- Convert control logic to Lua, have all parameterization code call “tweak” scripts.
  - No need to rebuild everything everytime a simple setting is changed.
- Very fast prototyping/field upgrades of new functionality.
  - No firmware upgrades.

Lua in your system

- Have customer customization code be Lua.
  - Customers can tweak their system themselves.
  - Customer can't break the system with bad code.
  - Customer doesn't see your monolith source.
  - Customer doesn't need a C/C++ compiler.
- Lua is very usable on 60Mhz ARM7tdmi!

Obrigado