

they don't alter build process

python scripts



upm

Unofficial, opinionated
NuttX project management tool

Goals

slightly

- Simplify NuttX setup tasks
- Lead the user by suggesting next steps
- Change folder structure to keep project-dedicated code together
- Group configuration options into project-related, logical modules

Short command list

```
$ upm new [project_name]
$ upm generate
  --app [name]
  --app-template [example]
$ upm generate
  --lib [name]
  --lib-template [example]
$ upm config
$ upm run
$ upm clean
$ upm console
$ upm console --telnet
```

No need to be in *nuttx* directory.

Desired project structure

/apps – just custom apps
/libs – just custom libs
/boards – just custom board code
/os – upstream bundle

 /nuttx

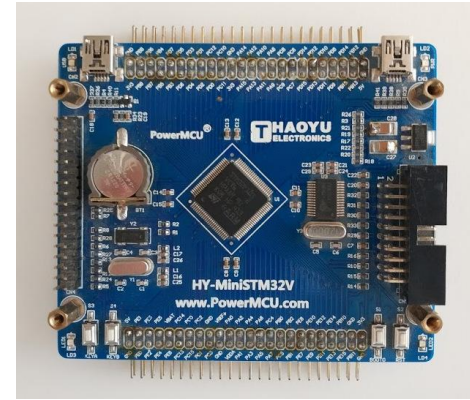
 /apps (custom_apps → ../../../../apps, custom_libs → ../../../../libs)

Why do we need CLI tool? Usual tasks.

- How to clone repositories and build the firmware?
- How to add a new app?
- How to add a custom board?
- How to add some library?
- How to use Git for new app, custom board and custom library?

How to clone, build, flash?

```
$ mkdir firmware  
$ cd firmware  
$ git clone https://bitbucket.org/nuttx/nuttx.git nuttx  
$ git clone https://bitbucket.org/nuttx/apps.git apps  
$ cd nuttx  
$ tools/configure.sh hymini-stm32v/usbnsb  
$ make  
$ st-flash write nuttx.bin 0x8000000
```

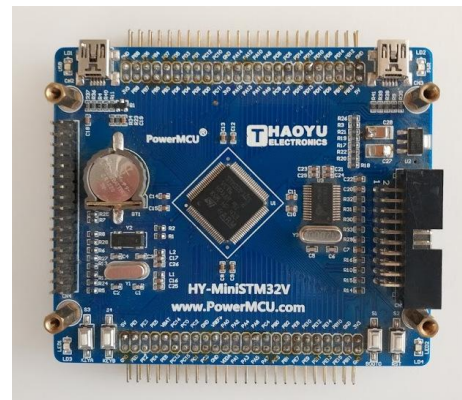


How to add a new app?

```
$ cd firmware  
$ cd apps/examples  
$ cp -r hello myapp  
$ cd ../../nuttx  
$ make
```

crash during linking because of duplicate names

```
$ cd ../../apps/examples/myapp  
$ find . -type f -exec sed -i "s/hello/myapp/g" {} \  
$ find . -type f -exec sed -i "s/HELLO/myapp/g" {} \  
$ mv hello_main.c myapp.cxx  
$ make
```



How to add a custom board code?

```
$ cd nuttx/configs  
$ cp -r stm32f4discovery custom_board
```

nuttx/configs/Kconfig

```
...  
+ config ARCH_BOARD_CUSTOM_BOARD  
+     bool "Custom board"  
+     depends on ARCH_CHIP_STM32F407VG || ARCH_CHIP_STM32F407ZG  
...  
+ default "custom_board" if ARCH_BOARD_CUSTOM_BOARD  
...  
+ if ARCH_BOARD_CUSTOM_BOARD  
+     source "configs/custom_board/Kconfig"  
+ endif  
...
```


Why do we need CLI tool?

NuttX currently relies on simplicity

Make, nuttx/tools

Kconfig-frontends

C/C++ preprocessor

A few symlinks

Tool is not necessary

- + Simplicity

- + Full control over the process

But we could use one

- + Provides some guidance

- + Easier entry for new people

- + What is logically one thing should be one command

Web frameworks

React.js – \$ react

Vue.js – \$ vue

Ember.js – \$ ember

Embedded

Zephyr – \$ west

ESP32 – \$ idf.py

nuttx/tools

What CLI tools do in frameworks?

- Simplification of common tasks, like
 - starting new projects
 - building
 - versioning and distribution
 - handling updates
 - deploying
- Generation of boilerplate code
- Warning about issues, suggesting solutions
- They include some know-how which user don't need to remember

one task, one command

Example upm usage

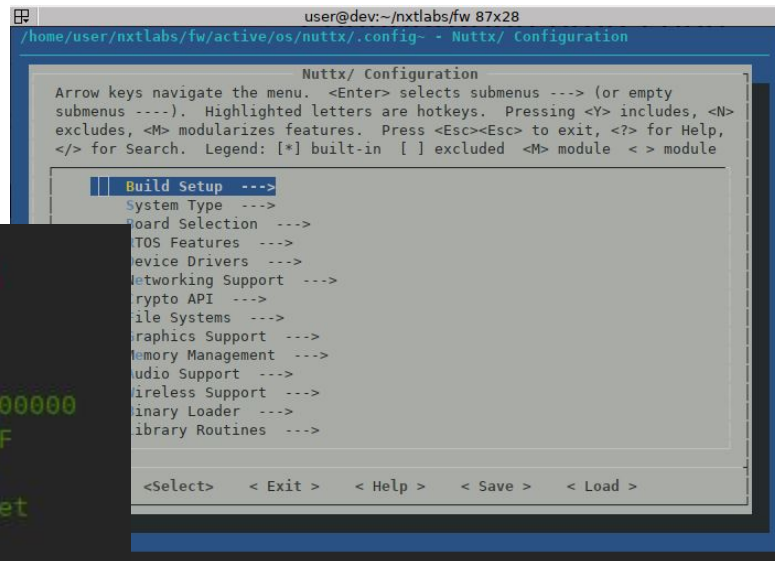
```
fw $ upm init .  
fw $ upm generate --app rest_api --app-template webserver  
fw $ upm use --board stm32f4discovery  
fw $ upm run  
fw $ upm console  
(enter)  
nsh>
```

- Generates directory structure
- Clones NuttX repositories
- Copies selected example app
- Links selected board
- Builds and flashes, connects to NuttX console

Configuration protection

```
fw $ upm config
```

```
1106c1106,1111
< # CONFIG_MTD_SST26 is not set
---
> CONFIG_MTD_SST26=y
> CONFIG_SST26_SPIMODE=0
> CONFIG_SST26_SPIFREQUENCY=80000000
> CONFIG_SST26_MANUFACTURER=0xBF
> CONFIG_SST26_MEMORY_TYPE=0x26
> # CONFIG_SST26_DEBUG is not set
1109,1113c1114
< CONFIG_MTD_GD25=y
< CONFIG_GD25_SPIMODE=0
< CONFIG_GD25_SPIFREQUENCY=80000000
< # CONFIG_GD25_READONLY is not set
< # CONFIG_GD25_SLOWREAD is not set
---
> # CONFIG_MTD_GD25 is not set
```



upm: do you accept the changes (y, n, q)? – rollback otherwise

Configuration modules – Kconfiglib

```
config/atoms/enc28j60.kconfig
```

```
NET=y
```

```
NET_TCP=y
```

```
NETDEVICES=y
```

```
ENC28J60=y
```

```
ENC28J60_LPWORK=y
```

```
ENC28J60_HALFDUPPLEX=y
```

```
NET_HOSTNAME=nuttx
```

```
NET_ROUTE=y
```

```
    # In order to reach destinations through default gateway
```

```
NET_ARP_SEND=y # For instant connection
```

```
config/modules/webserver
```

```
enc28j60
```

```
http
```

```
$ upm config --batch webserver
```

Philosophy

- Do not perform actions, just suggest them, be verbose

```
$ upm init firmware
```

```
upm: created .nuttx, active, apps, boards, dist, libs, oses, tmp
```

```
upm: as a next step, you might wish to download upstream NuttX
```

```
    cd firmware
```

```
    upm clone --os default
```

- Stay simple
- Always ask before doing intrusive change
- Accept commands from any subdirectory (similar to git)
- Show state when asked (similar to git)

- Tries to get a bit bigger picture than scripts in nuttx/tools

How upm could handle Spresense™?

- suggest cloning of **SDK** and cloning it for user
- suggest downloading of **firmware**, **flash_writer**
 - ask user to accept licenses and download files to some folder
 - say where to find them
 - warn if something is still missing
- etc.
 - warn if `python3-serial` module is missing

Maybe some plugin approach for platforms with extra requirements.

Challenges?

- some part still work in progress
- maybe disabling what is not polished would be a way to go
- dependency management?
- plugins
- git submodule management

REST APIs

Express-like + UI synchronization

REST APIs – Node.js Express

```
let express = require('express')
let mem = require('mem')
let app = express()

let memory_route = function (req, res) {
  res.json({memory : {
    total: mem.total,
    used: mem.used,
    largest: mem.largest
  }})
}

app.get('/memory', memory_route)

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```

Sinatra, Flask

REST APIs – same route in NuttX webserver

```
static void memory_route(struct httpd_state *pstate, char *ptr) {
    static const int memory_answer_max_size = 113; // content length with all INT_MIN, +1
    char body[memory_answer_max_size];
    int body_len;

    body_len = snprintf(body, memory_answer_max_size, "\
{\"memory\" : {\n\
  \"total\": %d,\n\
  \"used\": %d,\n\
  \"largest\": %d\n\
}}\n",
        mem.arena, mem_used.get(), mem_largest.get()
    );

    send_reply(pstate, 200, body, body_len);
}
```

REST APIs – Express-like list of routes

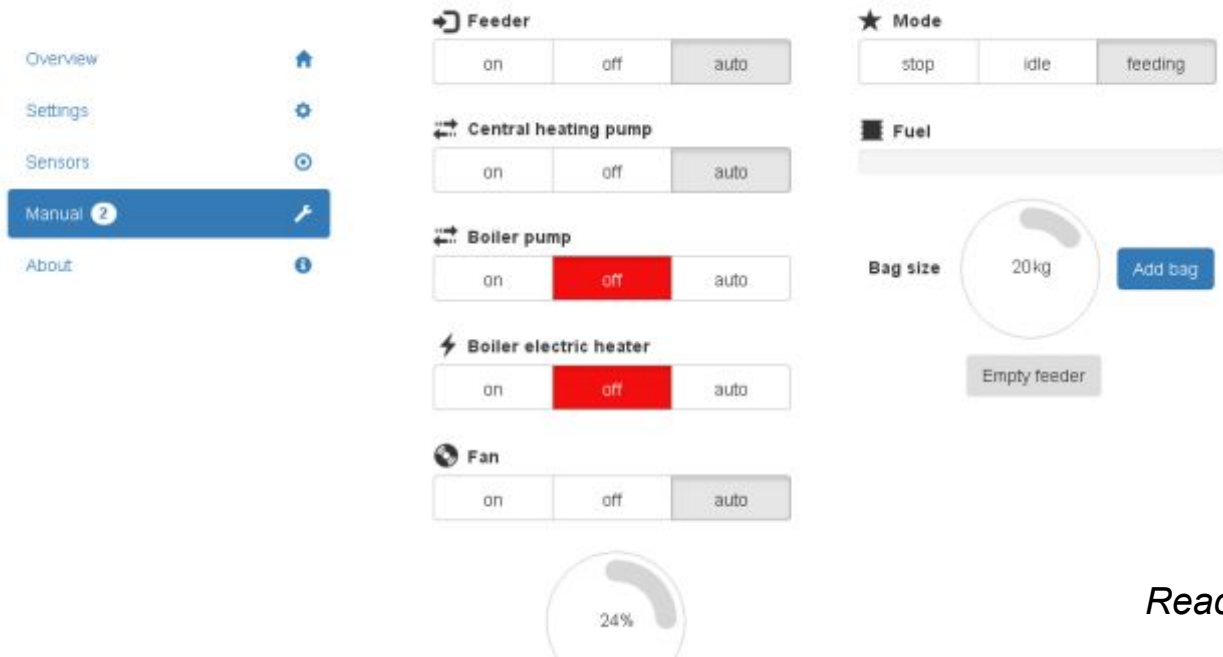
```
static struct httpd_cgi_call routes[] = {
    { NULL, "/"           , catchall_route },
    { NULL, "/os/memory", memory_route  },
    { NULL, "/os/time"   , time_route   },
    { NULL, "/console"   , console_route },
    { NULL, "/sensors"   , sensors_route },
    { NULL, "/adc"        , adc_route    },
    { NULL, "/state"     , state_route  }
};

httpd_serve(routes, (sizeof routes / sizeof *routes));
```

Simple helpers:

- <https://gitlab.com/w8jck/upm/snippets/1876008> (http.cxx)
- <https://gitlab.com/w8jck/upm/snippets/1876010> (http.h)

Reaching μc from JavaScript framework



React, Vue, Ember

What with two windows open?

Overview

Settings

Sensors

Manual

About

Feeder

on off auto

Central heating pump

on off auto

Boiler pump

on off auto

Boiler electric heater

on off auto

Fan

on off auto

24%

Mode

stop idle feeding

Fuel

Bag size 20kg

Empty feeder

Polling

Overview

Settings

Sensors

Manual

About

Feeder

on off auto

Central heating pump

on off auto

Boiler pump

on off auto

Boiler electric heater

on off auto

Fan

on off auto

24%

Mode

stop idle feeding

Fuel

Bag size 20kg

Empty feeder

Reporting only changed state

```
...
for (i = 0; i < reported_count; i += 1) {
    if (same_boot && previous_update_time >= (*all_reported[i]).get_changed_time()) {
        continue;
    }

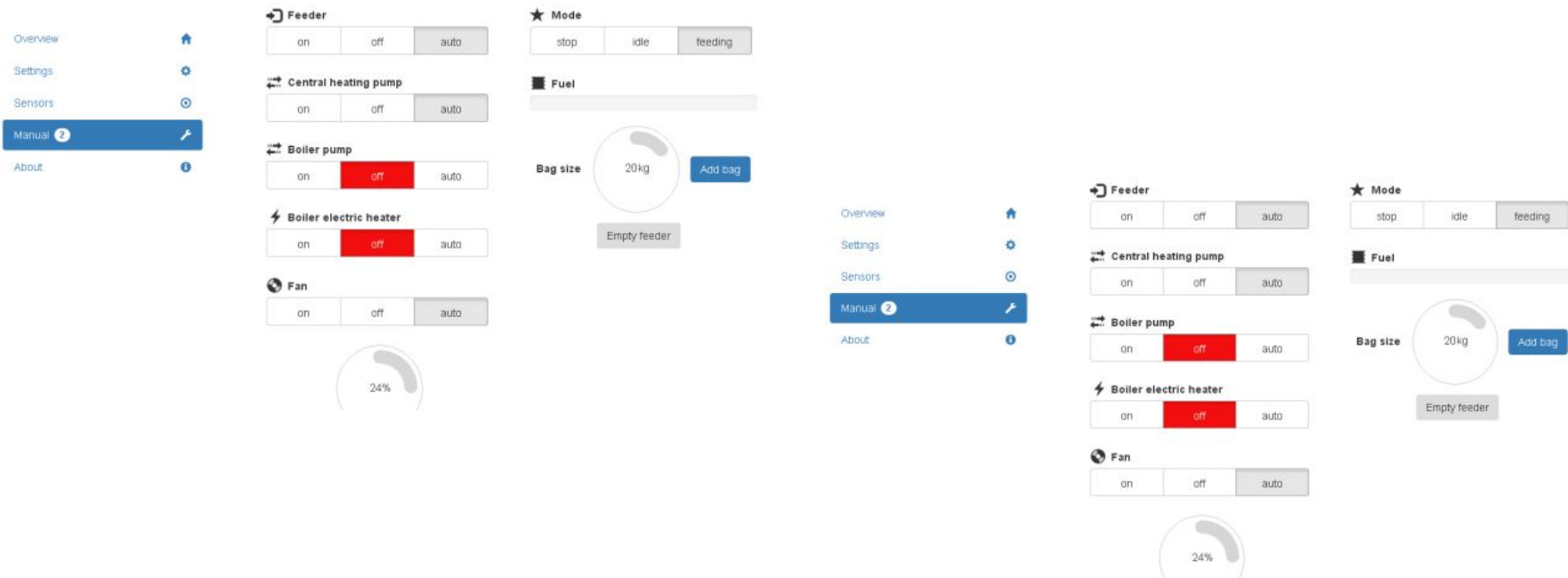
    any_reported = true;

    body_len += sprintf(&body[body_len], answer_max_size, "\n%s\n": %d,",
        (*all_reported[i]).get_name(),
        (*all_reported[i]).get()
    );
}
...
```

Variable.get_changed_time()
Variable.get_name()
Variable.get()

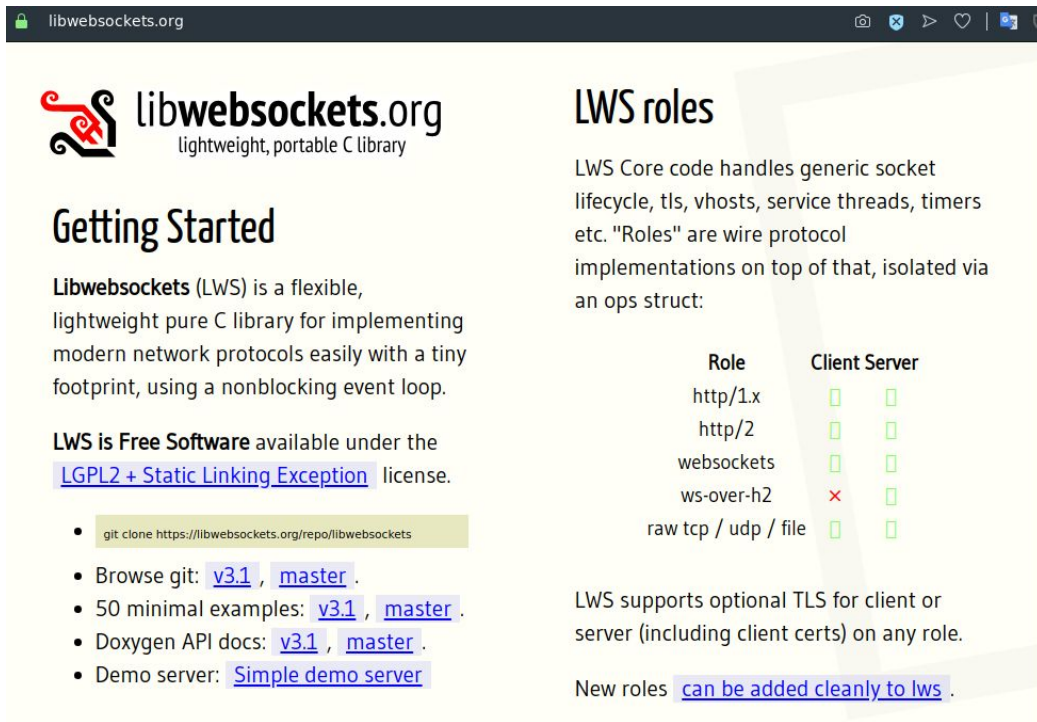
A lot of variables.

Two different browser windows synchronize



More efficient than polling

NuttX 2019
experience exchange :) →



The screenshot shows the libwebsockets.org website. The main heading is "Getting Started". Below it, a paragraph describes libwebsockets as a flexible, lightweight pure C library for implementing modern network protocols. A list of instructions for getting started is provided, including cloning the repository and browsing the code. To the right, a table titled "LWS roles" shows the support for various protocols on client and server sides.

libwebsockets.org
lightweight, portable C library

Getting Started

Libwebsockets (LWS) is a flexible, lightweight pure C library for implementing modern network protocols easily with a tiny footprint, using a nonblocking event loop.

LWS is Free Software available under the [LGPL2 + Static Linking Exception](#) license.

- git clone <https://libwebsockets.org/repo/libwebsockets>
- Browse git: [v3.1](#) , [master](#) .
- 50 minimal examples: [v3.1](#) , [master](#) .
- Doxygen API docs: [v3.1](#) , [master](#) .
- Demo server: [Simple demo server](#)

LWS roles

LWS Core code handles generic socket lifecycle, tls, vhosts, service threads, timers etc. "Roles" are wire protocol implementations on top of that, isolated via an ops struct:

Role	Client	Server
http/1.x	☐	☐
http/2	☐	☐
websockets	☐	☐
ws-over-h2	✗	☐
raw tcp / udp / file	☐	☐

LWS supports optional TLS for client or server (including client certs) on any role.

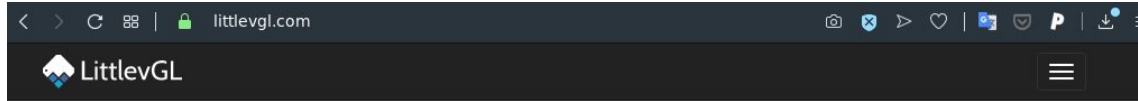
New roles [can be added cleanly to lws](#) .

LittlevGL

Small UI apps and emulation

LittlevGL 6.0 released this week

Screen rotation
Multiple displays



Open-source Embedded GUI Library

Fork me on GitHub



New Release:

v6.0

Download

GitHub

Release notes

Try in your Browser!

Donate

LittlevGL 5.3

Is it recognized enough?

- Present in apps/examples
- Example board configuration for /dev/fb0
- But can be also used without framebuffer
 - External RAM not required
 - Display connected in any way (SPI, FSMC)

Running in emulator – Mocks

Makefile

```
CFLAGS += -DINSIDE_NUTTX=1
```

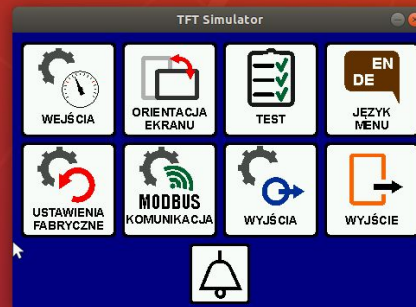
```
CXXFLAGS += -DINSIDE_NUTTX=1
```

```
#ifdef INSIDE_NUTTX
# include <graphics/lvgl.h>
#else
# include "lvgl/lvgl.h"
#endif
```

```
static int adc_read(adc_msg_t *sample) {
#ifdef INSIDE_NUTTX
    ...
#endif
}
```

```
#ifdef INSIDE_NUTTX
# include <sys/ioctl.h>
# include <nuttx/ioexpander/gpio.h>
# include <sys/boardctl.h>
# include "watchdog/watchdog.h"
#endif
```

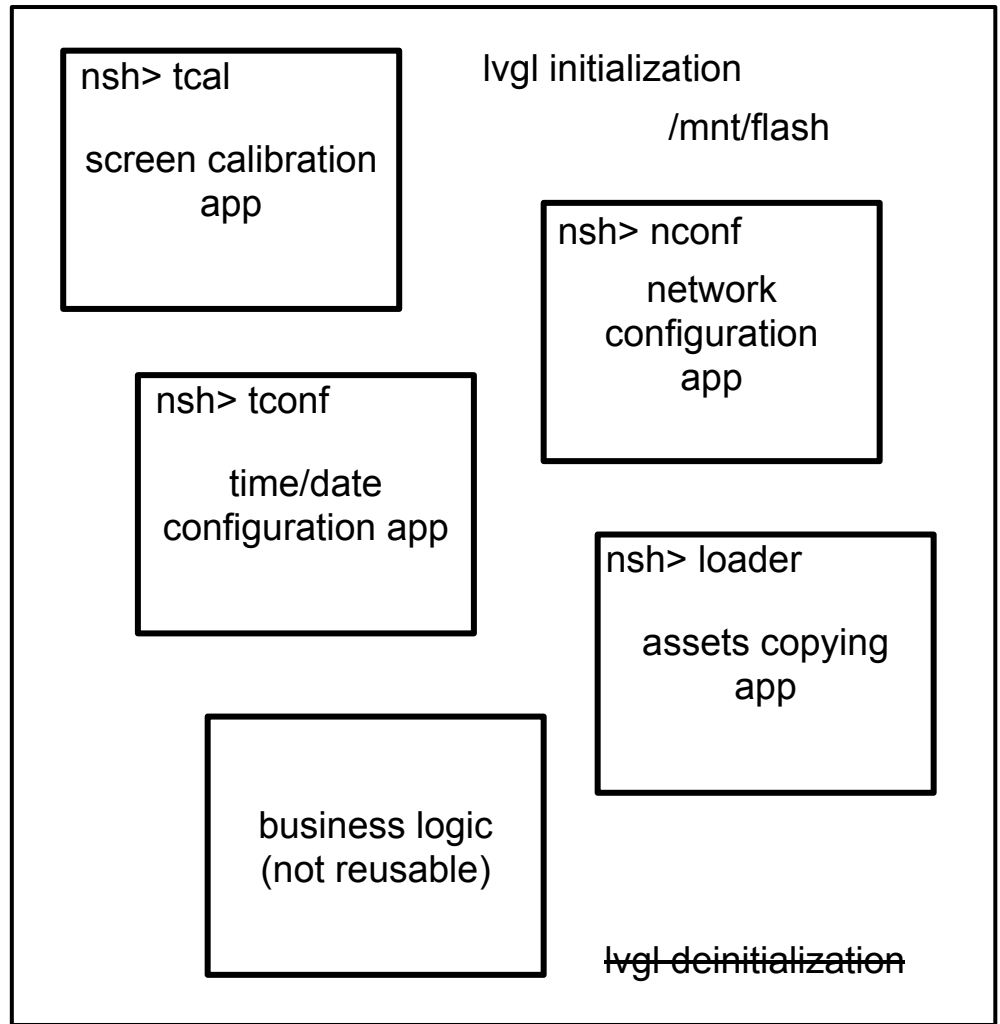
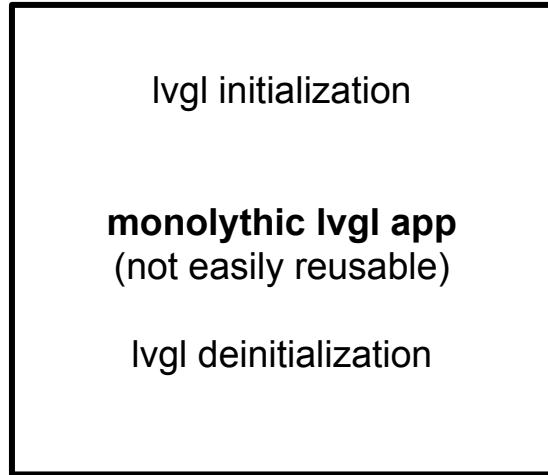
Emulator



Microservices

Microapps

Letting projects grow



Slight analogy to microservices and android intentions

Development UI development stages

(discussion and review)

UI prototyping tool



InVision



LittlevGL
Emulator



Microcontroller
NuttX

(visually correct prototype)

◆ Component

Button

Challenges?

– integration of LittlevGL emulator with upm

/oses

/default (/nuttx, /apps)

/lvgl_emulator

– lack of libraries for UI prototyping software

– check the process

