

# Building Embedded Operating System with IMGUI Demo for *Raspberry $\pi$ - 4 - model B* with *Yocto*

Kaloyan Krastev\*

Monday 23<sup>rd</sup> October, 2023 15:14

revision 2893

---

\*Triple Helix Consulting[8]

# table of contents

<b>1</b>	<b>introduction</b>	<b>3</b>
<b>2</b>	<b>metadata</b>	<b>4</b>
<b>3</b>	<b>configuration</b>	<b>7</b>
3.1	MACHINE . . . . .	7
3.2	PACKAGE_INSTALL . . . . .	7
3.3	IMAGE_FSTYPES . . . . .	8
<b>4</b>	<b>build</b>	<b>9</b>
<b>5</b>	<b>install</b>	<b>11</b>
<b>6</b>	<b>run</b>	<b>12</b>
<b>7</b>	<b>outlook</b>	<b>14</b>

# 1 introduction

These instructions[4] follow the configuration and build of a Linux-based operating system for *Raspberry  $\pi$  - 4 - model B*[6] with *Yocto*[2]. Find project overview in [5].

The *operating system* (OS) build is done in several steps organized in corresponding sections as follows. Read in Section 2 how to fetch *metadata*. Section 3 shows how to configure the OS build. In Section 4 learn how to build the OS *image* and see how to copy *image* to *SD* card in Section 5. Section 6 is dedicated to post-install issues like the configuration of the WiFi interface from the command line.

## 2 metadata

*Metadata* is a set of instructions to build targets. It is organized in *recipe* files with the *.bb* suffix. Further there are *class* files with the suffix *.bbclass* with information shared between *recipes*. Finally, there are configuration files with the extension *.conf*. These define configuration variables to control the build process. *Metadata* is organized in *layers*. Layers logically separate information of a project. *OpenEmbedded*[1] defines the following layer types.

- base layers contain base *metadata* for the build
- machine aka *board support package* (BSP) layers include *hardware* (HW) support
- distribution layers hold the policy configuration
- *software* (SW) layers are used for additional SW
- miscellaneous layers do not fall in upper categories

The complete list of *github SW metadata* repositories used in this project includes *Yocto* layers, the *Raspberry  $\pi$  - 4 - model B* BSP layer, a SW layer with custom recipes, and the build configuration itself. Please refer [5] for details.

In short, users fetch *metadata* in contrast to the *real data* fetched by *bitbake* during OS build. See Section 4 for details. It is an user decision where to put fetched *metadata*. However, it is nice to have all layer sub-directories in one location. In these instructions this location is referred as <layer\_directory>. The second directory to create is the <build\_directory>. This is where the build and build configuration live. I suggest that this one is not inside the <layer\_directory> to not mix *data* and *metadata*.

It is very likely that you will need to install *Yocto* requirements[3] to be able to run *bitbake*. Make sure to have the following command-line tools on your host machine: *chrpath*, *diffstat*, *lz4c*, *rpcgen*; Also have a look at your storage device. Fetched data requires 400 *Mb*, but for the build prepare at least 10 *Gb* of free space.

I have a shell script to fetch *metadata* from public *github* repositories. This modification may serve people to build their own OS for *Raspberry  $\pi$  - 4 - model B*. Reading the script may help advanced users to follow the fetch, initialisation and a check of installed layers.

```
#!/bin/sh
# metafetch.sh
# fetch rpi metadata

FETCHER=git@github.com
BRANCH=kirkstone

tchouss() { echo $*; exit 111; }

while getopts ":l:b:r:h" option; do

    case $option in
        h) tchouss usage: $0 -l layerdir -b buildir -r branch;;
        \?) tchouss minimal usage: $0 -l layerdir -b buildir;;
        l) LAYER=$OPTARG;;
        b) BUILD=$OPTARG;;
        r) BRANCH=$OPTARG;;
        *)
            esac
    done

    [ -n "$LAYER" ] || tchouss specify layer directory
    [ -n "$BUILD" ] || tchouss specify build directory

    LAYER=$(realpath $LAYER) && echo $FETCHER $BRANCH in $LAYER
    BUILD=$(realpath $BUILD) && echo $FETCHER configuration in $BUILD

    read -p "confirm (y/n) " choice
    case "$choice" in
        y ) echo $FETCHER;;
        * ) echo refused; rmdir -v $LAYER $BUILD; exit 0;;
    esac

    git clone -b $BRANCH $FETCHER:yoctoproject/poky.git $LAYER/poky
    git clone -b $BRANCH $FETCHER:openembedded/meta-openembedded.git $LAYER/oe
    git clone -b $BRANCH $FETCHER:agherzan/meta-raspberrypi $LAYER/rpi/meta-raspberrypi
    git clone $FETCHER:kaloyanski/meta-thc.git $LAYER/thc/meta-thc
    git clone $FETCHER:TripleHelixConsulting/rpiconf.git $BUILD/conf

    sed -i s#/home/yocto/layer/$LAYER/g $BUILD/conf/bblayers.conf

    OEINIT=$LAYER/poky/oe-init-build-env
```

```
[ -x $OEINIT ] && . $OEINIT $BUILD || tchouss cannot find $OEINIT  
bitbake-layers show-layers  
# bitbake core-image-x11
```

Download *metafetch.sh* [here](#). It is designed in a way that after a succesful run you may start a build with *bitbake*. The script takes `<layer_directory>` and `<build_directory>` from the command-line. You may use next commands to run *metafetch.sh*. The first one is a minimal example. The default *Yocto branch* is *kirkstone*. You may want to specify another *branch* with the second command.

```
./metafetch -l <layer_directory> -b <build_directory>  
./metafetch -l <layer_directory> -b <build_directory> -r <branch_name>
```

## 3 configuration

Build configuration is kept in two files located in `<build_directory>/conf`, namely *local.conf* and *bblayers.conf*. *Yocto* layers are specified in *bblayers.conf*. The build configuration is in *local.conf*. Variables in this file control the build. Sometimes I call these *directives* to avoid repetitions. Many directives are not covered in these instructions. Please refer *bitbake*[7] documentation for details. It is not always easy to understand the meaning and the relations between different directives. What is more, *bitbake* syntax is pretty complicated. In short, your life can easily become unbearable if the build configuration is too long.

### 3.1 MACHINE

No doubt, this is the most important directive, set here to *raspberrypi4-64*. You may want to change this value if you build an [OS](#) for a different [HW](#). If you want to examine [OS](#) built for *Raspberry  $\pi$  - 4 - model B* on your host machine with *qemu*, set *MACHINE* to *qemuarm64*. I confirm that this works although I did not find this approach very useful to test a *graphical user interface* ([GUI](#)).

### 3.2 PACKAGE\_INSTALL

This is where to specify additional [SW](#) packages. This is useful for packages not included in the *image* by default. In my experience, the default [OS](#) has all necessary programs or compact alternatives. However this is the directive used to append *imgui*.

### 3.3 IMAGE\_FSTYPES

This is another important directive. Here I have removed archived *images* that I do not need to decrease the built time and added the *wic* format to have an *image* file ready to be copied to the *SD* card immediately after the build. See [Section 5](#) for details.



## 4 build

*Yocto* provides a list of image types. As I want to have a compact OS and I need a *X* server to run a GUI, I rely on *core-image-x11*<sup>[2]</sup>. This is a very basic *X11* image.

The primary build tool of *OpenEmbedded* based projects, such as the *Yocto* project, *bitbake*, works in the `<build_directory>`. Here is a list of the most important sub-directory names by default. These are configurable but usually there is no need to change their default names.

- `<build_directory>/conf` - build (*local.conf*) and layer (*bblayers.conf*) configuration files
- `<build_directory>/downloads` - fetched source code archives
- `<build_directory>/tmp/work` - working directory where source code is extracted, configured, compiled and installed
- `<build_directory>/tmp/deploy/ipk` - final SW packages in *ipk* format
- `<build_directory>/tmp/deploy/images/raspberrypi4-64` - boot files, compiled kernels and OS images.

First, you need to initialize build environment.

```
source <layer_directory>/poky/oe-init-build-env <build_directory>
```

This will change your system path to `<build_directory>`. You may run now next command to check the project layers.

```
bitbake-layers show-layers
```

If this is fine, the following command is going to build the OS image.

task	description
do_fetch	fetch the source code
do_unpack	unpack the source code
do_patch	apply patches to the source
do_configure	source configuration
do_compile	compile the source code
do_install	copy files to the holding area
do_package	analyse holding area
do_package_write_ipk	create <i>ipk</i> package
do_package_qa	quality checks on the package

Table 1: A list of *bitbake* tasks

```
bitbake core-image-x11
```

Be patient because, unless your host machine is a supercomputer, this will take hours. Find a list of tasks performed by *bitbake* for a typical [SW](#) package in Table 1.

## 5 install

The [OS](#) includes a kernel *ARM*, 64 *bit* boot executable *image* of 23MB, a *Raspberry  $\pi$  - 4 - model B* configuration of Linux 5.15. The total size of kernel modules is 21MB. Happily this kernel release has a *long – term support* ([LTS](#)).

*Yocto* provides multiple package and *image* formats. Further, different ways exist to install *images* on *SD* card. The result is an [OS](#) with two partitions - */root* and */boot*. There are not *swap* and *home* partitions. I recommend the classic command-line tool *dd* to copy data. It works fine with different *image* formats like *rpi – sdimg*, *hddimg* and *wic*. The last format is recommended. Find the card device name, usually */dev/sda*, unmount it with *umount* if mounted, and do copy data with a simple command

```
dd if=core-image-x11-raspberrypi4-64.wic of=/dev/sda status=progress
```

note 1: run this command in `<build_directory>/tmp/deploy/images/raspb`  
64

note 2: run this command with *root* privileges

note 3: be careful to not specify the device name of your hard drive (see note 2)

The transfer is going to take a while. Once it is over, put the card in you *Raspberry  $\pi$  - 4 - model B* and turn it on. That's it.

## 6 run

Wireless connection is established via classic command-line tools like *ip* and *iw*. I use a *dynamic host configuration protocol* (DHCP) client, *udhcpd*, and *wpa\_supplicant* to store WiFi connection. Custom shell scripts are installed in */usr/bin*, as well as a running GUI example to demonstrate the usage of the *Dear ImGui* library.

```
#!/bin/sh
# wifini.sh
# wifi connection
# requirements:
# wpa_passphrase wpa_supplicant, ip, iw, grep, awk
# contact kaloyansen at gmail dot com
# copyleft triplehelix-consulting.com
#####

WINTERFACE=$(iw dev | grep Interface | awk '{print $2}')
SSID=PuzlCowOrKing
WPACONF=/etc/wpa_supplicant.conf
WPASOCKET=/run/wpa_supplicant/$WINTERFACE
UDHCPID=/run/udhcpd.$WINTERFACE.pid

error() { echo $*; exit 1; }

while getopts ":i:s:h" option; do
    case $option in
        i) WINTERFACE=$OPTARG;;
        s) SSID=$OPTARG;;
        h) error usage: $0 -s SSID;;
        *) error warning: option unknown;;
    esac
done

ip link show $WINTERFACE | grep UP || ip link set $WINTERFACE up
iw $WINTERFACE scan | grep $SSID || error warning: cannot find network $SSID;
grep $SSID $WPACONF || wpa_passphrase $SSID >> $WPACONF
[ -S "$WPASOCKET" ] || wpa_supplicant -B -D wext -i $WINTERFACE -c $WPACONF
# [ -f "$UDHCPID" ] ||
udhcpd -i $WINTERFACE || error $?

# ip addr show $WINTERFACE
echo interface $WINTERFACE SSID $SSID
# iw $WINTERFACE link
# ip route show
```

Download script [here](#). Usage

```
./wifini -s <SSID>
```

Connected embedded systems can communicate to one another

and to cloud-based *platform-as-a-service* ([PaaS](#)) solutions. In addition, a remote control may be required. *secure shell* ([SSH](#)) server is a standard solution for both problems. Once an *internet protocol* ([IP](#)) address is assigned, the [SSH](#) server by *Dropbear* allows for a secured remote login, remote control and file transfer.

## 7 outlook

This reports the progress in the development of a custom Linux-based OS for *Raspberry  $\pi$  - 4 - model B*[6]. The kernel version of this embedded OS is Linux release 5.15. An example GUI application using the *Dear ImGui* library is built as a part of the OS image. In addition, an SSH server provides remote connection, data transfer and device control. As the OS is now functional, performance and real-time tests are ongoing.

## acronyms

**BSP** *board support package*

**SSH** *secure shell*

**GUI** *graphical user interface*

**SW** *software*

**HW** *hardware*

**OS** *operating system*

**DHCP** *dynamic host configuration protocol*

**IP** *internet protocol*

**PaaS** *platform-as-a-service*

**LTS** *long – term support*

# bibliography

- [1] Yocto Project community. *OpenEmbedded*. 2017.  
URL: <https://www.openembedded.org> (visited on 2023).
- [2] Yocto Project community. *Yocto Project*. 2023.  
URL: <https://www.yoctoproject.org> (visited on 2023).
- [3] Yocto Project community. *Yocto Project Quick Build*. 2023. URL:  
<https://docs.yoctoproject.org/brief-yoctoprojectqs/index.htm>  
(visited on 2023).
- [4] Kaloyan Krastev. *Building Embedded Operating System with  
IMGUI Demo for Raspberry  $\pi$  - 4 - model B with Yocto*. 2023. URL:  
<https://kaloyanski.github.io/meta-thc/thchowto.html>  
(visited on 2023).
- [5] Kaloyan Krastev. *Embedded Operating System for Raspberry  $\pi$   
- 4 - model B with Yocto*. 2023. URL:  
<https://kaloyanski.github.io/meta-thc/thcport.html>  
(visited on 2023).
- [6] Raspberry  $\pi$  Ltd. *Rasperi  $\pi$* . 2023.  
URL: <https://www.raspberrypi.com> (visited on 2023).
- [7] Richard Purdie, Chris Larson, and Phil Blundell.  
*BitBake User Manual*. 2023. URL:  
<https://docs.yoctoproject.org/bitbake> (visited on 2023).
- [8] Atanas Rusev. *Triple Helix Consulting*. 2023. URL:  
<https://triplehelix-consulting.com> (visited on 2023).