How to write a really good board support package for Yocto Project

Chris Simmonds

NDC Techtown 2021

License

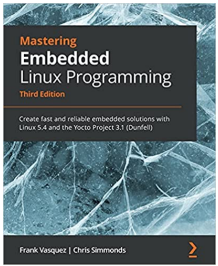
These slides are available under a Creative Commons Attribution-ShareAlike 4.0 license. You can read the full text of the license here
http://creativecommons.org/licenses/by-sa/4.0/legalcode
You are free to

- copy, distribute, display, and perform the work
- make derivative works
- make commercial use of the work

Under the following conditions

- Attribution: you must give the original author credit
- Share Alike: if you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one (i.e. include this page exactly as it is)
- For any reuse or distribution, you must make clear to others the license terms of this work

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at https://2net.co.uk/

@2net_software

https://uk.linkedin.com/in/chrisdsimmonds/

Board Support Packages

- In Yocto Project, a Board Support Package (BSP) is the meta layer which contains all the configuration specific to a particular board
- Every project needs a BSP layer
- The BSP is provided by either:
  - the company that sells the board
  - you, if you work for the company that makes the board
- Either way, the BSP makes a big difference to the project

Agenda

- Overview of Yocto Project
- Board Support Packages
- First rule of BSP layers: keep it simple
- Second rule of BSP layers: don’t break things
- Q & A

Yocto Project

https://www.yoctoproject.org/

- Yocto Project is a build system that creates packages from source code
  - based on Bitbake and OpenEmbedded meta data
  - Yocto Project and OpenEmbedded have been used to create the software running on many millions of devices
- Allows you to create your own tailor-made distro
- You only need to build and deploy the packages you need

Yocto Project versions

Releases every 6 months in April and October (approximately)

YP version	Poky version	Code name	Release	Status
3.4	26	Honister	Oct 21	
3.3	25	Hardknott	Apr 21	Stable
3.2	24	Gatesgarth	Oct 20	EOL
3.1	23	Dunfell	Apr 20	LTS
3.0	22	Zeus	Oct 19	EOL
2.7	21	Warrior	May 19	EOL
2.6	20	Thud	Nov 18	EOL
2.5	19	Sumo	Apr 18	EOL
2.4	18	Rocko	Oct 17	EOL
2.3	17	Pyro	May 17	EOL

To find the version installed, look at `meta-poky/conf/distro/poky.conf`
List of versions and support levels:
https://wiki.yoctoproject.org/wiki/Releases

Getting Yocto Project

```
$ git clone git://git.yoctoproject.org/poky -b hardknott
```

The download is about 250 MiB, of which

- 55 MiB is tools, documentation and meta data
- 195 MiB is git history

It does not contain the upstream code that will compile and construct the images for your chosen platform

Setting up the environment

Begin by sourcing this script

```
$ cd poky
$ source oe-init-build-env [build dir]
```

- Creates a working directory for your project, default `build/`
- Changes into that directory

Local configuration

- Local configuration is in `[build dir]/conf/local.conf`
- Can contain many configuration variables, including

Variable	Description	Example
`MACHINE`	Target board	`MACHINE = "beaglebone"`
`DISTRO`	Distribution	`DISTRO = "poky"`
`PACKAGE_CLASSES`	Package format	`PACKAGE_CLASSES = "package_rpm"`
`EXTRA_IMAGE_FEATURES`	Additional features	`EXTRA_IMAGE_FEATURES = "debug-tweaks"`

Recipes

- The core meta data consists of recipes grouped together into **meta layers**
- The recipes are processed by a task scheduler named **BitBake**
- Recipes generate binary packages
- Packages combine to make images which can be copied to a device

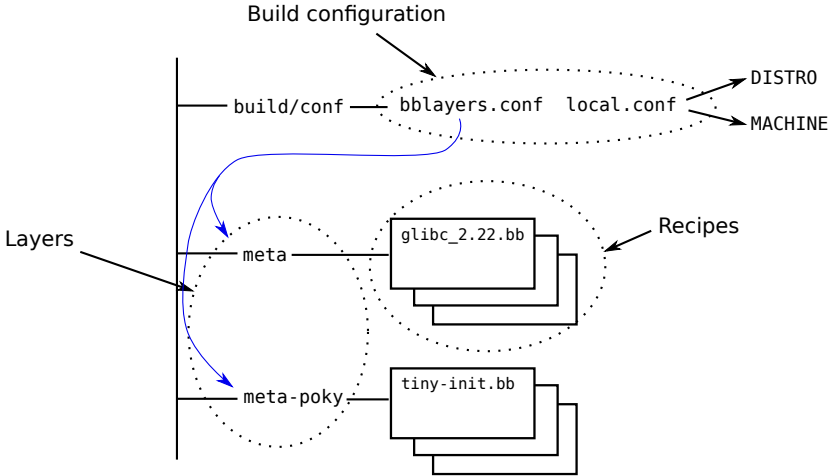
Recipes

- Here is a simple recipe that builds a "helloworld" program

poky/recipes-skeleton/hello-single/hello_1.0.bb

```
DESCRIPTION = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
    ${CC} ${LDFLAGS} helloworld.c -o helloworld
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 helloworld ${D}${bindir}
}
```

Config, layer and recipe



bblayers

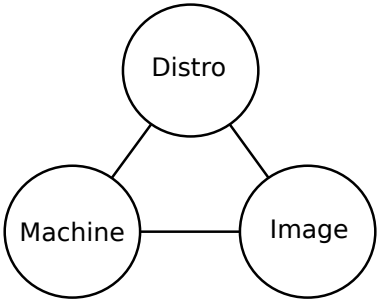
- The layers actively used in a project are listed in `conf/bblayers.conf`

Initially `bblayers.conf` looks like this:

```
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/chris/poky/meta \
  /home/chris/poky/meta-poky \
  /home/chris/poky/meta-yocto-bsp \
  "
```

- Overview of Yocto Project
- Board Support Packages
- First rule of BSP layers: keep it simple
- Second rule of BSP layers: don’t break things
- Q & A

The trinity of OE: Distro, Machine, Image



- **DISTRO**: how I want to put my system together
- **MACHINE**: the board I want to build for
- **Image**: the selection of packages I want

Three types of layer

- **BSP**: defines a MACHINE and related board-specific packages
  - contains `conf/machine/[MACHINE].conf`
- **Distribution**: defines a DISTRO such as Poky or Ångström
  - contains `conf/distro/[DISTRO].conf`
- **Software**: everything else
  - contains neither `conf/machine/[MACHINE].conf` nor `conf/distro/[DISTRO].conf`
  - libraries, e.g. qt5
  - languages, e.g. Java
  - tools, e.g. virtualisation or selinux

The unmentioned fourth type

- There **are** layers that contain **both** `conf/machine/[MACHINE].conf` and `conf/distro/[DISTRO].conf`
- This is not good practice
- Really, they should not exist

- Overview of Yocto Project
- Board Support Packages
- First rule of BSP layers: keep it simple
- Second rule of BSP layers: don’t break things
- Q & A

What goes into a BSP layer?

- MACHINE configuration
- Bootloader selection and configuration
- Kernel selection and configuration
- Firmware binaries of various kinds
- Hardware enabling components, such as Gstreamer
- AND NOTHING ELSE

meta-raspberrypi is a good, non-trivial example

Create a layer for your BSP

The BSP should be in a layer of its own

For example, for my **nova** range of boards:

```
$ cd $BUILDDIR
$ bitbake-layers create-layer ../meta-nova
$ bitbake-layers add-layer ../meta-nova
```

The layer should have a `README` file that describes the BSP

It should also have a license which covers the use of the metadata in the layer. `COPYING.MIT` is a common choice

Create the machine configuration

The configuration for the board (or boards) supported by your BSP goes into `conf/machine/[MACHINE].conf`

For example I have two boards, nova and nova-pro, so `conf/machine` contains:

```
conf/machine/nova.conf
conf/machine/nova-pro.conf
```

I can select which one by setting in my `conf/local.conf`

```
MACHINE := "nova-pro"
```

These files contain all the settings unique to the board

Bootloader

Select bootloader, usually by adding something like this to

`conf/machine/nova.conf`

```
EXTRA_IMAGEDEPENDS += "u-boot"
```

`EXTRA_IMAGEDEPENDS` is a list of packages that need to be built, but not included in the rootfs

Then, any modifications to the u-boot recipe would go into (using version 2021.01 as an example):

`meta-nova/recipes-bsp/u-boot/u-boot_2021.01.bbappend`

Digression 1: package versions

- Each recipe has a version
  - Usually part of the recipe file name, e.g. `u-boot_2021.01.bb`
- If there are several versions, BitBake selects the **latest**
- You can override using `PREFERRED_VERSION`
- e.g. `PREFERRED_VERSION_u-boot = "2020.11"`
- "%" is a wildcard,
- e.g. `PREFERRED_VERSION_linux-yocto = "5.4%"`

Digression 2: bbappend

- You can modify an existing recipe, e.g. `u-boot_2021.01.bb` with a file named `u-boot_2021.01.bbappend`
  - the contents of `u-boot_2021.01.bbappend` are literally appended to `u-boot_2021.01.bb` before parsing the recipe
- You can use wildcards in bbappend filenames, e.g. `u-boot_%.bbappend`

Kernel 1/2

- Select the upstream kernel and version
- Select the kernel config

These will be set in machine.conf, e.g.

```
PREFERRED_PROVIDER_virtual/kernel ?= "linux-nova"
PREFERRED_VERSION_linux-nova ?= "5.10.%"
```

Note '?=' allows the user of this layer to override them if needed

Digression: BitBake assignment operators

=	lazy - expand only when referenced
:=	immediate - as in C or C++
?=	assign only if value currently empty
??=	similar to ?=, but assign only if value still empty after parsing other assignments

'?=' and '??=' are prefereable in base layers such as the BSP because they can be overridden in a higher level bbappend if the situation demands

Kernel 2/2

- If the kernel is unique to your board ("linux-nova") add the recipe in the layer

```
meta-nova/recipes-kernel/linux/linux-nova_5.10.bb
```

- Otherwise, use a bbappend to supply any changes to the kernel config

```
recipes-kernel/linux/linux-yocto_5.10.bbappend
```

Device trees

- Often, you need to make changes to the device tree to accommodate hardware on your board
- The device tree is part of the kernel code
  - The new dts goes into (for example)
    `recipes-kernel/linux/linux-nova/dts/nova.dts`
- You select the device tree(s) to be built and deployed in the [MACHINE].conf

```
KERNEL_DEVICETREE = "nova.dtb"
```

Firmware

- Proprietary binary blobs are (alas) quite common

Example for bcm43430 wifi chip found in meta-raspberrypi3

In `conf/machine/raspberrypi3.conf`

```
MACHINE_EXTRA_RRECOMMENDS += "\
    linux-firmware-rpidistro-bcm43430
```

Then, recipe
`/recipes-kernel/linux-firmware-rpidistro/linux-firmware-rpidistro_git.bb`
downloads and installs the blob

Image format

- The ultimate output of the build is a set of image files that can be programmed into the flash memory of the target device
- Yocto can generate different image formats, e.g.
  - tar file: extract into formatted partition
  - partition image (e.g. ext4, jffs2): raw copy to disk or MTD partition
  - disk image (wic): raw copy to disk

Setting the image type

- List the formats you want in the machine conf file
- IMAGE_FSTYPES

```
IMAGE_FSTYPES = "ext3 tar.bz2"
```

Creating images with WIC

- **WIC** creates partitioned images ready to be copied directly to eMMC, SD cards, etc [1]
- Understands various partition table formats, including pcbios, GPT, CDROM and SDcard
- Layout is controlled by **wks** (WIC Kick Start) files
- Search path for wks files:
  - `wic/` in each layer
  - `scripts/lib/wic/canned-wks` in each layer

[1] Originally this was the OpenEmbedded Image Creator, **OEIC**, which was impossible to pronounce so it was changed to **WIC**

Enabling WIC

- Add wic to `IMAGE_FSTYPES`
- Specify name of wks file in `WKS_FILE`
- Example (from `meta-yocto-bsp/conf/machine/beaglebone-yocto.conf`)

```
IMAGE_FSTYPES += "tar.bz2 jffs2 wic"
WKS_FILE ?= "beaglebone-yocto.wks"
```

- Overview of Yocto Project
- Board Support Packages
- First rule of BSP layers: keep it simple
- Second rule of BSP layers: don’t break things
- Q & A

Don’t break things

- Your BSP layer will be used with many other layers
  - You should not interfere with them (e.g. with unnecessary bbappends)

Check you can build for another machine

- Check that you can include your BSP layer and still build for a machine **not** in your layer
  - Mostly tests that your bbappends don't break anything

Dependencies between layers

- Your BSP layer should depend on other layers that contain recipes it needs
- We don't want to see messages like this:

```
ERROR: ParseError at /home/chris/poky/meta-nova/recipes-kernel/linux/linux-nova
_4.19.15.bb:5: Could not include required file recipes-kernel/linux/linux-imx.inc
```

- Need something like this in the BSP `conf/layer.conf`

```
LAYERDEPENDS_nova = "core freescale-layer"
```

Check the layer

Use the `yocto-check-layer` tool to make sure everything is set up correctly

```
$ yocto-check-layer ../meta-nova
INFO: Detected layers:
INFO: meta-example: LayerType.BSP, /home/chris/poky/meta-nova
[...]
INFO: Summary of results:
INFO:
INFO: meta-example ... PASS
```

What about my demos apps?

- The BSP layer is NOT a marketing tool to showcase the company
- Put demos (e.g. my home automation image) into other layers
- Consider packaging your layers (including the BSP) using repo, git submodules, etc
- For example Digi provide a repo manifest for their boards at `https://github.com/digi-embedded/dey-manifest`

Installs these layers:

```
meta-digi  meta-freescale  meta-fsl-demos  meta-openembedded  meta-qt5  meta-swupdate
```

Yocto compatibility

- It is a great idea to be clear which version of Yocto your layer is compatible with
  - Please don't mask it with your own versioning scheme
- Put it in your layer.conf, e.g.

```
LAYERSERIES_COMPAT_nova = "sumo thud warrior zeus"
```

- Better still, certify and get a Yocto compatible badge

- Even better, put it on the OE Layer index index
  http://layers.openembedded.org/layerindex

Advantages in getting it right

- Simpler BSP = reduced development costs
- The community will help out
- Users will be able to use your BSP alongside other layers - reduced support costs
- There is a wealth of information online to help people develop OE/YP solutions

- Overview of Yocto Project
- Board Support Packages
- First rule of BSP layers: keep it simple
- Second rule of BSP layers: don’t break things
- Q & A

Questions?

Slides at
https://2net.co.uk/slides/yocto-bsp-csimmonds-ndctechtown-2021.pdf

@2net_software

https://uk.linkedin.com/in/chrisdsimmonds/