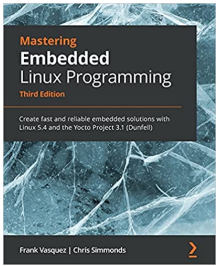
Getting started with Yocto Project

Chris Simmonds

NDC Techtown 2022

License

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/`

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

Embedded hardware

Types of hardware that are Linux capable

- Commodity hardware
  - limited flexibility
  - embedded PC, x86 - powerful, but expensive and high power requirement
  - Raspberry Pi - cheap, but not industry spec (see RPi CM though)
- Bespoke hardware
  - cheaper, less power dissipation (no fan), more flexible, physically smaller packages
  - mostly ARM
  - RISC-V is catching up
- Semi-bespoke
  - SBC, SoM

Embedded hardware

- Embedded hardware generally does not follow rigid standards as PCs do
- For reasons of cost and flexibility, each project is unique
- Hence, off the shelf is not usually an option
- Mostly using ARM SoCs - which are cheap, low power, have lots of peripherals integrated into the chip
- Bespoke is the majority of implementations

Embedded Linux

- For commodity hardware: Off the shelf distro, e.g. Debian (Raspberry Pi OS)
- For bespoke designs you need a tool to build the operating system, e.g. Yocto
- Embedded hardware generally does not follow rigid standards as PCs do
- For reasons of cost and flexibility, each project is unique
- Hence, off the shelf is not usually an option

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

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

Who uses Yocto?

These are open source projects that use Yocto to build every-day devices

A little history

- **2003** OpenEmbedded is born as a common build system and code base for iPaq, Zaurus and SIMpad mobile devices
- **2004** OpenedHand work with Nokia to create the N770 WebPad using a fork of OpenEmbedded.
  That fork was called Poky Linux
- **2008** Intel buy out OpenedHand
- **2010** Intel and Linux Foundation create the Yocto Project based on Poky Linux

Poky

- Why is it called Poky?

"Pocky" is a snack, popular in Japan

The poky reference distribution is named after the snack.
It's pronounced "poky" to rhyme with "hockey"

Yocto Project versions

Releases every 6 months in April and October (approximately)

YP version	Poky version	Code name	Release	Status
4.0	27	Kirkstone	May 22	LTS
3.4	26	Honister	Oct 21	EOL
3.3	25	Hardknott	Apr 21	EOL
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

Clone from the Yocto git repository. There is one branch for each release and tags for each minor update

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

The download is about 250 MiB, of which

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

Setting up the environment

Begin by sourcing this script

```
$ source ./oe-init-build-env <builddir>
```

- Creates a working directory for your project, default `build/`
- Changes into that directory
- The directory path is stored in shell variable `$BUILDDIR`
- To return to this directory later, just

```
$ cd $BUILDDIR
```

Note: you must `source ./oe-init-build-env <builddir>` **each time you use OpenEmbedded from a new shell**

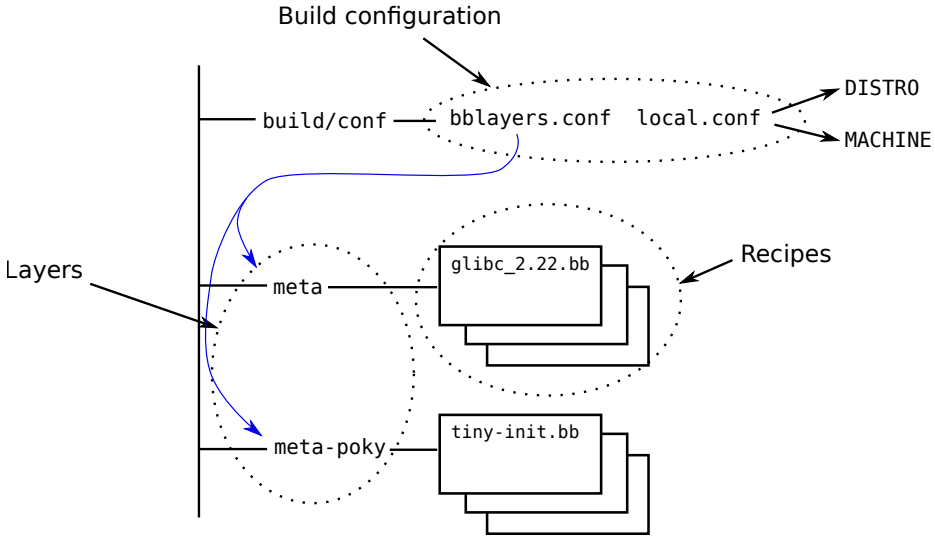
Yocto directory layout

```
poky/
|-- bitbake/
|-- build/
|   |-- conf/
|   |   |-- bblayers.conf
|   |   `-- local.conf
|-- documentation/
|-- meta/
|-- meta-poky/
|-- meta-yocto-bsp/
|-- oe-init-build-env
```

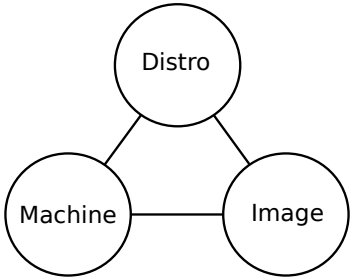
Setting the build configuration

- Build configuration files are in `$BUILDDIR/conf/`
- The initial set is:
  - `bblayers.conf`: list of directories containing layers
  - `local.conf`: local configuration

Config, layer and recipe



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

DISTRO

- In `conf/local.conf`, selected by `DISTRO`
- Example

```
DISTRO ?= "poky"
```

- This selects the Poky distro, defined in `meta-poky/conf/distro/poky.conf`

MACHINE

- In `conf/local.conf`, `MACHINE` selects target machine
- Example

```
MACHINE := "beaglebone-yocto"
```

- Each MACHINE has a corresponding configuration file in `<layer>/conf/machine/<machine>.conf`
- For BeagleBone Black it is `meta-yocto-bsp/conf/machine/beaglebone-yocto.conf`

BitBake and recipes

- OpenEmbedded uses **BitBake** to build the target
- Bitbake reads **recipes** to create a dependency tree
- Then executes all recipes required to build the final target
- To begin with we will build an **image** recipe

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

Some standard image recipes

- **core-image-minimal**: small console-based image, useful for tests and as the basis for custom images
- **core-image-base**: a console-only image that fully supports the target device
- **core-full-cmdline**: a console-only image with full-featured Linux system functionality installed
- **core-image-x11**: small X11 server based graphical system, including xterminal terminal app
- **core-image-sato** full graphical system based on Sato (a mobile GUI built on X11, and GNOME)

Building an image

- To build the image, simply run BitBake and the image name
  - generally, to build any recipe, give it as a parameter to BitBake
- For example

```
$ bitbake core-image-minimal
```

Demo - building an image for ARM QEMU

Build artifacts 1/3

```
|-- cache/
|-- conf/
|-- downloads/
|-- sstate-cache/
`-- tmp/
    |-- deploy/
    |   |-- images/
    |   |   `-- qemuarm
    |   |-- licenses/
    |   `-- rpm/
    `-- work/
```

Build artifacts 1/3

- In `$BUILDDIR`:
  - **cache**/: locally cached state
  - **downloads**/: Source code and other upstream tarballs
  - **sstate-cache**/: shared state: you can make this an NFS export and share amongst a group of developers
  - **tmp**/: the build artifacts

Build artifacts 2/3

- In `$BUILDDIR/tmp`:
  - **buildstats**/: useful information about CPU usage, time taken ...
  - **deploy**/: code to be deployed to the target
  - **work**/: the staging (build) area for packages
    - work/ is not needed after a build: can be safely deleted

Build artifacts 3/3

- In `$BUILDDIR/tmp/deploy`:
  - **rpm**/: the packages built by bitbake
  - **images**/: the images, ready to be copied to the target
  - **licenses**/: license for each package

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"
```

Shared state cache

- Binary build artifacts are put into the **shared state cache**
  - Speeds up subsequent builds
  - Can be shared with other developers
- When running BitBake, you will notice
  - Building from shared state cache:

```
NOTE: Executing SetScene Tasks
```

  - Building from source:

```
NOTE: Executing RunQueue Tasks
```

- Pruning the sstate `sstate-cache-management.sh --cache-dir=sstate-cache -d`

Build directory paths

- Location of build directories is set by these variables (which you can change in `local.conf`)
- The default settings are relative to BitBake variable `$TOPDIR`, which is identical to shell variable `$BUILDDIR`

Variable	Default	Purpose
`SSTATE_DIR`	`"${TOPDIR}/sstate-cache"`	Shared state
`TMPDIR`	`"${TOPDIR}/tmp"`	Build artifacts
`DL_DIR`	`"${TOPDIR}/downloads"`	Downloads

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

Layers

- 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 \
  "
```

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

Other layers

- Layers make OpenEmbedded extensible
- There are many third party layers
- Official list at `http://layers.openembedded.org`
- Some examples:
  - **meta-webos** WebOS distribution
  - **meta-qt5**: Qt5 libraries and utilities
  - **meta-raspberrypi**: BSPs for Raspberry Pi
  - **meta-python** Python language support

Demo - Adding a layer

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

Recipes

- Contain instructions on how to fetch, configure, compile, and install a software component
- The **body** contains BitBake metadata (assignment of variables, mostly); the **tasks** are written in shell script or Python
- Recipe files have suffix **.bb**
- May be extended with append recipes with **.bbappend** suffix

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}
}
```

Packages and recipes

- The majority of recipes produce packages
- Often one recipe produces several packages, for example libz:

```
$ cd tmp/deploy/rpm/cortexa57
$ ls libz*
```

`libz1-1.2.11-r0.cortexa57.rpm`	runtime
`libz-dbg-1.2.11-r0.cortexa57.rpm`	binaries with debug info
`libz-dev-1.2.11-r0.cortexa57.rpm`	header and linker files
`libz-doc-1.2.11-r0.cortexa57.rpm`	documentation (e..g. man pages)
`libz-src-1.2.11-r0.cortexa57.rpm`	source code
`libz-staticdev-1.2.11-r0.cortexa57.rpm`	archive (for static linking)

Adding a package

- Image recipes, such as `core-image-minimal` contain a list of packages in `IMAGE_INSTALL`
- You can add extra packages by appending to `IMAGE_INSTALL` in `conf/local.conf`
- For example, to add the Dropbear (ssh) and lighttpd (web) servers:

```
IMAGE_INSTALL:append = " dropbear lighttpd"
```

Note the leading space before the package name (dropbear)

- Another way to achieve the same result for core image recipes is:

```
CORE_IMAGE_EXTRA_INSTALL += "dropbear lighttpd"
```

Demo - Adding my own recipe

- Embedded hardware
- Yocto Project
- Images
- Layers
- Recipes
- Conclusion

Conclusion

- Yocto Project is the industry-standard way of constructing bespoke Linux distros for embedded devices
- Generates reproducible image builds (with license manifest and SBOM)
- Layers is a key concept - allows you to add recipes from 3rd parties
- Scales well to large teams (each working on their own layer)
- Yocto SDK (not mentioned in this talk) provides tools to app developers: they don't have to learn all Yocto to be productive

Questions?

Slides at
https://2net.co.uk/slides/
getting-started-with-yocto-csimmonds-ndctechtown-2022.pdf

@2net_software

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