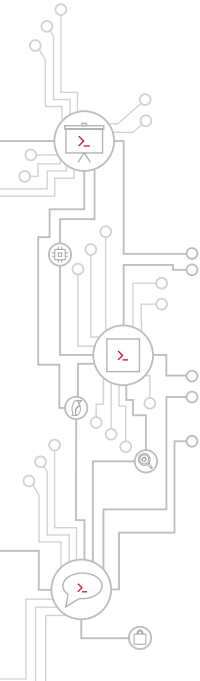


Yocto Crash Course

2021-04-10

Richard Weinberger

Hello

Richard Weinberger

- Co-founder of sigma star gmbh
- Linux kernel developer and maintainer
- Strong focus on Linux kernel, low-level components, virtualization, security

Agenda

- Goals and non-goals of this talk
- Motivation
- Yocto Overview
- Case Study: Raspberry Pi
- Best Practices
- Further Reading

Non-goals

- Promoting Yocto as the ultimate solution
- Giving you a full documentation on Yocto, in fact I'll skip a *lot*

Goals

- Give you a rough overview
- You get a feeling for Yocto
- You know what to google for

Motivation

- Your most recent embedded boards needs an operating system
- Can it run a regular Linux distribution or any other read-to-use OS?
- If so, problem solved
- Sadly this is seldom the case
- Embedded systems usually have special requirements

Motivation: Why creating your own Linux OS?

- Custom core components: bootloader, device tree, kernel
- Non-standard filesystem setup: e.g. readonly using squashfs or erofs, encrypted and/or authenticated
- Need to modify core packages such as libc, qt, Xorg
- Strict package selections: e.g. no GPLv3 components on the device
- Custom upgrade mechanism, plus recovery tooling

Yocto Overview: Nomenclature

- Yocto is the name of the project
- Uses the OpenEmbedded build framework
- OpenEmbedded itself uses the BitBake build tool
- Yocto includes a reference implementation of all together: poky
- Confused? ;-)

Yocto Overview: It's not an embedded Linux Distribution

- You can create your own
- Not just a root filesystem
- You get packages (deb, rpm, ipkg, tar), images, sdk, ...
- Includes a reference distro: poky
- Usually you base your on distro on poky

Yocto Overview: Components

- Layers
- Recipes (.bb)
- Appends (.bbappend)
- Classes (.bbclass)
- Config files
- Don't panic

Yocto Overview: Layers

- Host for recipes, classes, configs
- Layers are stackable and can override recipes from other layers
- Minimal setup: just poky
- Common setup: poky, BSP (board support package), your own layer

Yocto Overview: Recipes

- Think of Makefiles
- Describe how to build a package and dependencies
- Can emit more than one package
- You can include other files and inherit from other classes
- e.g. if the package needs cmake, inhert the cmake bbclass
- You can use both shell and python

Yocto Overview: Appends

- Allows you to change existing recipes
- e.g. `rng-tools_%.bbappend` will match any rng-tools recipe version
- Usually you append patch files to the source variable, SRC_URI

Yocto Overview: Classes

- Host common code for BitBake recipes
- Common use cases: How to build a package with buildsystem XY or generate a filesystem

Yocto Overview: Config files

- Describes a machine (the target you build for)
- Describes the local build settings
- Describes the layer itself

Yocto Overview: Anatomy of a typical project

- Poky plus a set of layers
- A target machine
- A target image
- Local build configuration

Yocto Overview: Machine

- Selects kernel, bootloader and device tree
- Select maschine specific packages. e.g. display drivers
- But also the cross compiler target

Yocto Overview: Image

- Basically a set of packages
- Selects one or more output formats. e.g. ext4, tar, sdcard image

Case Study: Raspberry Pi

- meta-raspberrypi is a BSP layer for the RPi
- So we can use Yocto to build a Linux distribution with the Raspberry Pi as target

Raspberry Pi

```
$ apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
     build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
     xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
     pylint3 xterm

$ mkdir rpi
$ cd rpi
$ git clone --branch dunfell git://git.yoctoproject.org/poky
$ git clone --branch dunfell git://git.openembedded.org/meta-openembedded
$ git clone --branch dunfell git://git.yoctoproject.org/meta-raspberrypi

# will cd to rpi-build/
$ source poky/oe-init-build-env rpi-build

# edit conf/bblayers
# add full paths to meta-openembedded/meta-oe, meta-openembedded/meta-networking,
# meta-openembedded/meta-multimedia, meta-openembedded/meta-python and meta-raspberrypi

# edit conf/local.conf
# set MACHINE to raspberrypi4

$ bitbake core-image-base

$ bmaptool copy rpi-build/tmp/deploy/images/raspberrypi4/core-image-base-raspberrypi4.wic.bz2 /dev/your_sd_card
```

Raspberry Pi: Confusion++

- Where did I know all these layers from?
- And the machine name?
- And the image name?
- Every BSP layer should have a documentation
- meta-raspberrypi has a nice README file

Raspberry Pi: Interesting files in this layer

- README.md
- conf/machine/include/rpi-base.inc
- conf/machine/raspberrypi4.conf

Raspberry Pi: Our own layer: Leia

```
$ mkdir -p meta-leia/conf
$ add full path to meta-leia to bblayer.conf
# open meta-leia/conf/layer.conf and insert:

# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "meta-leia"
BBFILE_PATTERN_meta-leia = "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-leia = "8"

LAYERSERIES_COMPAT_meta-leia = "dunfell"
```

Raspberry Pi: Our own distro

```
$ mkdir meta-leia/conf/distro
# edit meta-leia/conf/distro/leia.conf and insert:

require conf/distro/poky.conf

VENDOR = "leia"
TARGET_VENDOR = "-${VENDOR}"
MAINTAINER = "Richard Weinberger <richard@sigma-star.at>"

DISTRO = "leia"
DISTRO_NAME = "Princess Leia"
DISTRO_VERSION = "v1"
DISTRO_CODENAME = "science"

DISTRO_FEATURES += "pam wifi x11 vulkan opengl systemd"

DISTRO_FEATURES_remove = " 3g bluetooth irda nfc"

DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_dev_manager = "udev"
```

Raspberry Pi: Our own image

```
$ mkdir meta-leia/recipes-leia/images/
$ edit meta-leia/recipes-leia/images/leia-image.bb and insert:

DESCRIPTION = "Leia main image"

require recipes-core/images/core-image-minimal.bb

IMAGE_INSTALL = "packagegroup-core-boot \
    packagegroup-core-x11 \
    packagegroup-xfce-base \
    kernel-modules \
"
inherit features_check
REQUIRED_DISTRO_FEATURES = "x11"

IMAGE_LINGUAS ?= " "

LICENSE = "MIT"

export IMAGE_BASENAME = "leia-image"

RPI_STUFF = " \
        gpiozero linux-firmware-rpidistro-bcm43455 python3-image \
        python3-pip python3-spidev raspi-gpio rpi-gpio udev-rules-rpi userland \
"

IMAGE_INSTALL += "${RPI_STUFF}"
IMAGE_FEATURES += "package-management splash x11-base"
```

Raspberry Pi: Leia

```
# edit conf/local.conf and set DISTO to "leia"
$ bitbake leia-image
```

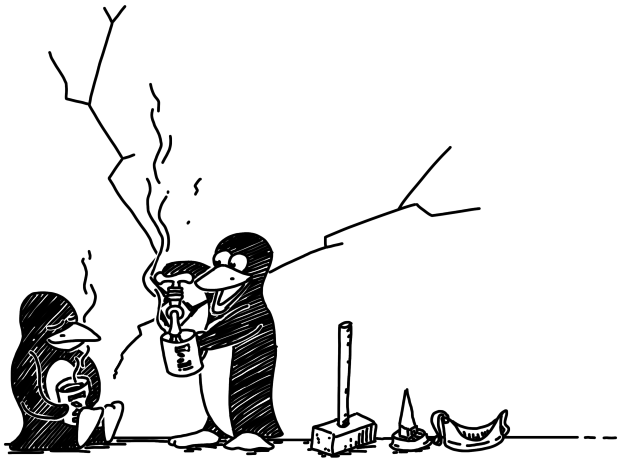
Best Practices

- Use tools such as KAS to manage build configurations and layers
- Never ever modify a layer except your own
- Build within a docker container, have source and build dir as volumes
- Share DL_DIR across other Yocto projects
- Inherit from poky and create your own distro
- Inherit from a core image to create your own image, e.g. core-image-minimal.bb

Reading list

- https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html
- https://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html
- https://kas.readthedocs.io/en/latest/
- https://elinux.org/Bitbake_Cheat_Sheet
- http://layers.openembedded.org/layerindex/branch/master/layers/
- https://docs.yoctoproject.org/what-i-wish-id-known.html
- https://wiki.yoctoproject.org/wiki/TipsAndTricks/Patching_the_source_for_a_recipe

FIN



Thank you!

Questions, Comments?

Richard Weinberger
richard@sigma-star.at

Backup slides: Devtool

- Often you need to modify a package
- Use case: Fixing the latest OpenSSL drama
- `devtool modify RECIPE`
- Will prepare the source and let you edit the git repository
- Change source code and commit
- `devtool update-recipe -a your-layer RECIPE`
- Will create a bbappend file in your layer and adds your changes as patch

Backup slides: SDK

- In a large company usually not everyone touches the Yocto project
- Application developers expect just a software development kit
- Yocto can generate you one, it will include a cross compiler plus userspace libraries
- `bitbake -c populate_sdk` your-image-name
- See tmp/deploy/sdks/
- A ready to use installer for Linux