



Yocto Project/OpenEmbedded Advanced Course



Introduction to Advanced features of the OpenEmbedded Build System in the Yocto Project



Behan Webster
behanw@converseincode.com

The Linux Foundation

Mar 05, 2017

Yocto Project Dev Day Wifi information

If you want to connect to the Internet:
SSID:



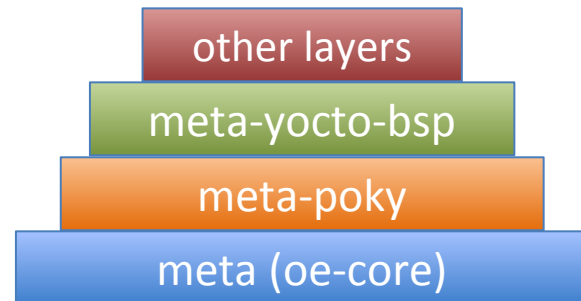
The URL for this presentation

<http://bit.ly/2mpHMLu>



Yocto Project Overview

- **Collection of tools and methods enabling**
 - ◆ Rapid evaluation of embedded Linux on many popular off-the-shelf boards
 - ◆ Easy customization of distribution characteristics
- **Supports x86, ARM, MIPS, Power**
- **Based on technology from the [OpenEmbedded Project](#)**
- **Layer architecture allows for easy re-use of code**



What is the Yocto Project?

- Collaborative Project under Linux Foundation
 - Backed by many companies interested in making Embedded Linux easier for industry
 - Split governance model
 - Technical Leadership Team
 - Advisory Board made up of participating organizations
- Co-maintains OpenEmbedded Core and other tools (including opkg)

Yocto Project Release Versions

Name	Revision	Poky	Release Date
Bernard	1.0	5.0	Apr 5, 2011
Edison	1.1	6.0	Oct 17, 2011
Denzil	1.2	7.0	Apr 30, 2012
Danny	1.3	8.0	Oct 24, 2012
Dylan	1.4	9.0	Apr 26, 2013
Dora	1.5	10.0	Oct 19, 2013
Daisy	1.6	11.0	Apr 24, 2014
Dizzy	1.7	12.0	Oct 31, 2014
Fido	1.8	13.0	April 22, 2015
Jethro	2.0	14.0	Oct 31, 2015
Krogoth	2.1	15.0	April 29, 2016
Morty	2.2	16.0	Oct 28, 2016
Pyro	2.3	17.0	April, 2017

Intro to OpenEmbedded

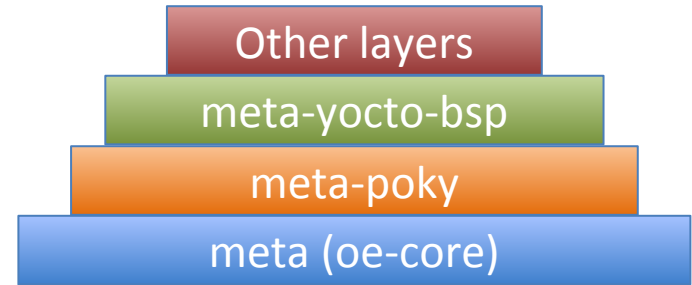
- **The OpenEmbedded Project co-maintains OE-core build system:**
 - ◆ bitbake build tool and scripts
 - ◆ Metadata and configuration
- **Provides a central point for new metadata**
 - ◆ (see the OE Layer index)

OK, so what is Poky?

- **Poky is a reference distribution**
- **Poky has its own git repo**
 - ◆ git clone git://git.yoctoproject.org/poky

- **Primary Poky layers**

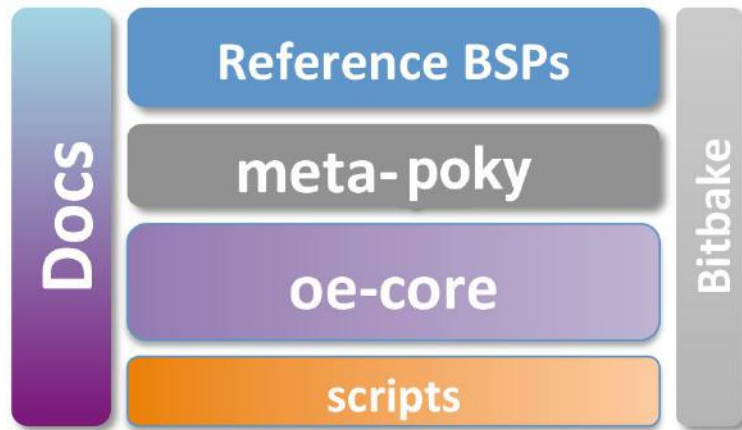
- ◆ oe-core (poky/meta)
- ◆ meta-poky (poky/meta-poky)
- ◆ meta-yocto-bsp



- **Poky is the starting point for building things with the Yocto Project**

Poky in Detail

- **Contains core components**
 - ◆ Bitbake tool: A python-based build engine
 - ◆ **Build scripts (infrastructure)**
 - ◆ **Foundation package recipes (oe-core)**
 - ◆ meta-poky (Contains distribution policy)
 - ◆ Reference BSPs
 - ◆ Yocto Project documentation



Yocto is based on OpenEmbedded-core

index : openembedded-core

OpenEmbedded Core layer

summary refs log **tree** commit diff about

path: [root/meta](#)

Mode	Name	Size		
-rwxr-xr-x	COPYING.GPLv2	17987	log	plain
-rwxr-xr-x	COPYING.MIT	1035	log	plain
d-----	classes	7261	log	plain
d-----	conf	794	log	plain
d-----	files	236	log	plain
d-----	lib	60	log	plain
d-----	recipes-bsp	728	log	plain
d-----	recipes-connectivity	850	log	plain
d-----	recipes-core	1307	log	plain
d-----	recipes-devtools	3174	log	plain
d-----	recipes-extended	2809	log	plain
d-----	recipes-gnome	77	log	plain
d-----	recipes-graphics	94	log	plain
d-----	recipes-kernel	675	log	plain
d-----	recipes-lsb4	64	log	plain
d-----	recipes-multimedia	745	log	plain
d-----	recipes-qt	248	log	plain
d-----	recipes-rt	102	log	plain
d-----	recipes-sato	948	log	plain
d-----	recipes-support	2161	log	plain
-rwxr-xr-x	recipes.txt	1407	log	plain
d-----	site	1506	log	plain

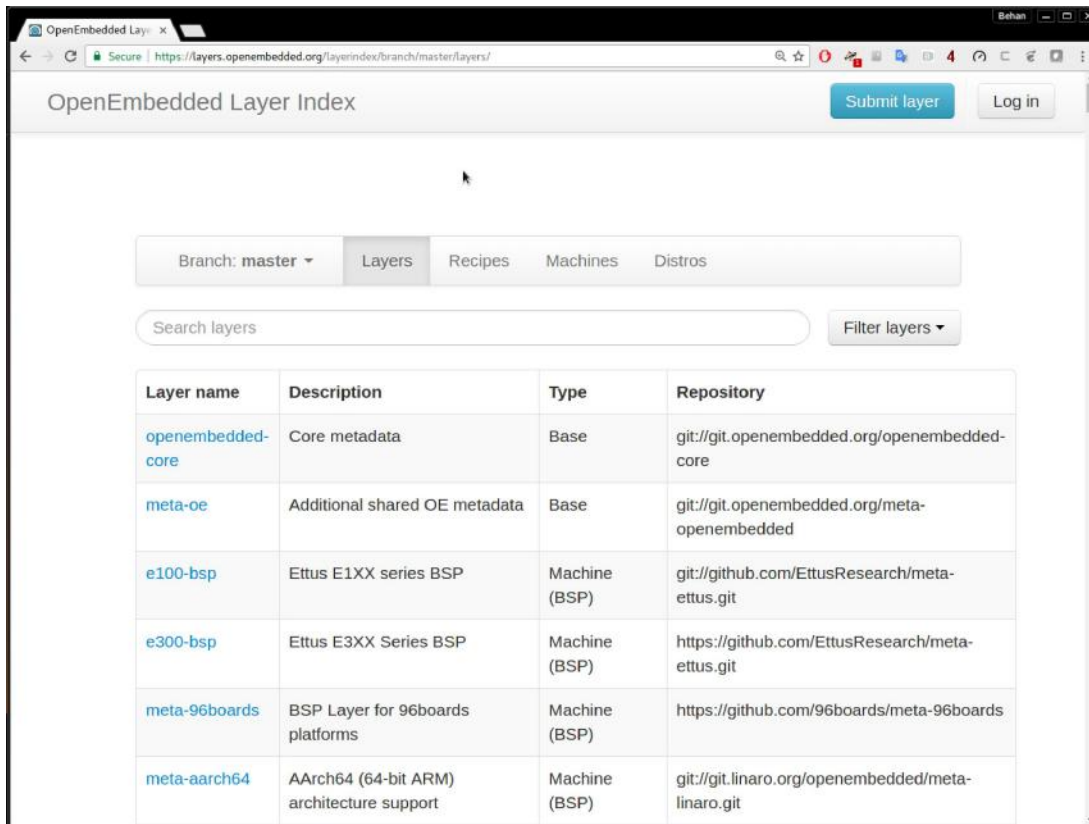
generated by `cglt v0.9.2-21-gd82e` at 2014-08-19 14:31:48 (GMT)

Yocto is based on OpenEmbedded-core

The Layer Index lists all known layers and allows you to search for:

- Specific recipes
- Machine BSP
- Distros

Ties into other tools like Toaster and the upcoming setup tool



The screenshot shows the OpenEmbedded Layer Index website. The page title is "OpenEmbedded Layer Index" and the URL is "https://layers.openembedded.org/layerindex/branch/master/layers/". The page has a navigation bar with "Branch: master", "Layers", "Recipes", "Machines", and "Distros". Below the navigation bar is a search bar labeled "Search layers" and a "Filter layers" dropdown. The main content is a table with the following data:

Layer name	Description	Type	Repository
openembedded-core	Core metadata	Base	git://git.openembedded.org/openembedded-core
meta-oe	Additional shared OE metadata	Base	git://git.openembedded.org/meta-openembedded
e100-bsp	Ettus E1XX series BSP	Machine (BSP)	git://github.com/EttusResearch/meta-ettus.git
e300-bsp	Ettus E3XX Series BSP	Machine (BSP)	https://github.com/EttusResearch/meta-ettus.git
meta-96boards	BSP Layer for 96boards platforms	Machine (BSP)	https://github.com/96boards/meta-96boards
meta-aarch64	AArch64 (64-bit ARM) architecture support	Machine (BSP)	git://git.linaro.org/openembedded/meta-linaro.git

BUILDING A FULL EMBEDDED IMAGE WITH YOCTO

This section will introduce the concept of building an initial system image, which we will be using later...

Cheating a Little

- **Create a yocto directory**

```
$ mkdir $HOME/yocto/
```

```
$ cd $HOME/yocto/
```

- **Unpack the 2 tarballs which will speed up the build**

```
$ tar xvf downloads.tar
```

```
$ tar xvf sstate-cache.tar
```

- **This download mirror and SSTATE-CACHE have been prepared a head of class to make this faster...**

Getting a copy of Poky

Download the Yocto Project release “Morty”:

```
~/ $ cd yocto
```

```
~/yocto$ git clone -b morty git://git.yoctoproject.org/poky.git
```

Setting up a Build Directory

- Start by setting up a build directory

```
$ cd $HOME/yocto/
```

```
$ source ./poky/oe-init-build-env build
```

- You need to re-run this script in any new terminal you start (and don't forget the project directory)

Host System Layout

```
$HOME/yocto/  
|---build      (or whatever name you choose)  
    Project build directory  
|---downloads (DL_DIR)  
    Downloaded source cache  
|---poky      (Do Not Modify anything in here*)  
Poky, bitbake, scripts, oe-core, metadata  
|---sstate-cache (SSTATE_DIR)  
    Binary build cache
```


Build directory Layout

```
$HOME/yocto/build/  
|---bitbake.lock  
|---cache/           (bitbake cache files)  
|---conf/  
|   |--bblayers.conf (bitbake layers)  
|   |--local.conf    (local configuration)  
|   `--site.conf     (optional site conf)  
`---tmp/            (Build artifacts)
```

Note: A few files have been items omitted to facility the presentation on this slide

Using Layers

- Layers are added to your build by inserting them into the BBLAYERS variable within your bblayers file

`$HOME/yocto/build/conf/bblayers.conf`

```
BBLAYERS ?= "  
    ${HOME}/yocto/poky/meta  
    ${HOME}/yocto/poky/meta-poky  
    ${HOME}/yocto/poky/meta-yocto-bsp  
"
```

Update Build Configuration

- Configure build by editing local.conf

`$HOME/yocto/build/conf/local.conf`

- ◆ Add the following to the bottom of local.conf

```
MACHINE = "beaglebone"
```

```
DL_DIR = "${TOPDIR}/../downloads"
```

```
SSTATE_DIR = "${TOPDIR}/../sstate-cache"
```

- Notice how you can use variables in setting these values

Building an Embedded Image

- This builds an entire embedded Linux distribution
- The following builds a minimal embedded target
\$ `bitbake -k core-image-minimal`
- On a fast computer the first build may take the better part of an hour on a slow machine multiple ...
- The next time you build it (with no changes) it may take as little as 5 mins (due to the shared state cache)

SD-card installation

- Insert your uSD card into your card reader
 - If you don't know the device which represents your card reader, ask an instructor
 - likely `/dev/mmcblk0` or `/dev/sdb`

```
$ cd tmp/deploy/images/beaglebone/
```

```
$ sudo dd if=core-image-minimal-beaglebone.wic of=/dev/<YourSDCard> bs=1M
```

```
$ eject /dev/<YourSDCard>
```

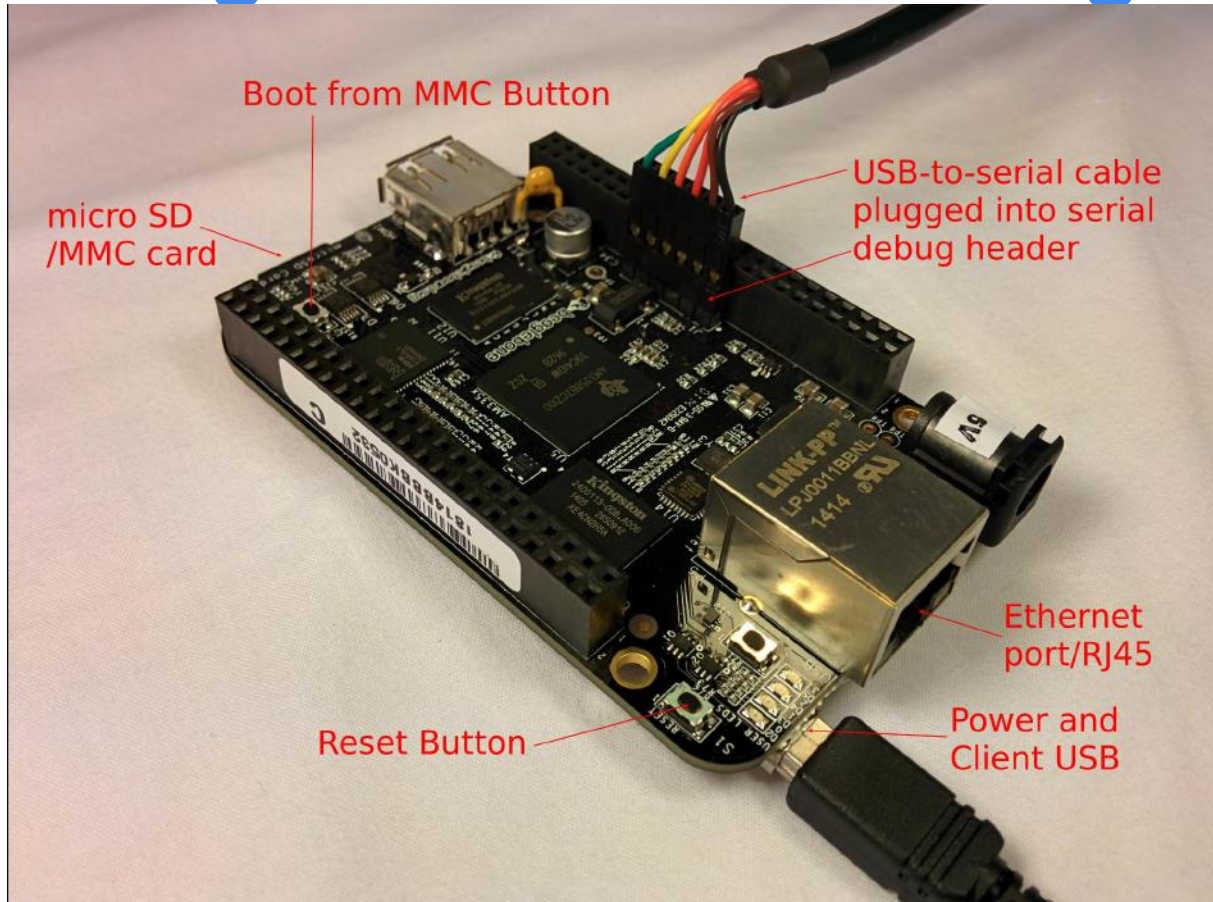
(YourSDCard will be something like `/dev/sdc` or `/dev/mmcblk0`)

- Connect to the serial and boot.

BEAGLEBONE SETUP

This section will show you how to setup your board

BeagleBone Walkthrough



Cable installation

- Plug in your USB-2-Serial with the Black wire on pin one (marked by a dot closest to 5V)
- Plug mini-USB cable into board for power
- Plug both USB-A plugs into your computer
- Alternately plug the power USB into a phone charger or power bank

Boot from external uSD

- Hold down the boot button
 - The button at the USB end of the board
- Unplug and replug the mini-USB power
- Release the boot button
- Until the board is powered down, it will boot from the external uSD

Connect to serial port

- There are many serial terminals
- We will use screen

```
$ sudo screen /dev/ttyUSB0 115200
```

- Login to the beaglebone as “root”

Board Support packages

This section will introduce the concept of board support packages

Board Support Packages

- **Documentation**
 - **Hardware Features**
 - **Configuration Data**
 - **Source Patches**
 - **Binary files**
- <http://www.yoctoproject.org/docs/current/bsp-guide/bsp-guide.html>

Board Support Packages (2)

- **Built as a meta layer**
 - ◆ **meta-<BSP>**
- **Alternately several BSPs can be in a container layer**
 - ◆ **For instance meta-intel**

Board Support Packages (3)

- **BSPs can be found in the layer index**
- **BSP can have dependencies**
- **Typically employ bbappend files**
- **The configuration for a BSP is in**
meta-mybsp/conf/machine/mybsp.conf
- **Set MACHINE=mybsp in local.conf**

Board Support Packages (4)

- **Yocto project has the yocto-bsp script for templating new BSPs**

\$ yocto-bsp list karch

\$ yocto-bsp create boardfoo arm

The yocto-kernel tool

- **yocto-kernel <command> [args]**
 - ◆ **config {add,rm,list}**
 - ◆ **patch {add,rm,list}**
 - ◆ **feature {add, rm, list}**
 - ◆ **features list**
 - ◆ **feature {describe, create, destroy}**

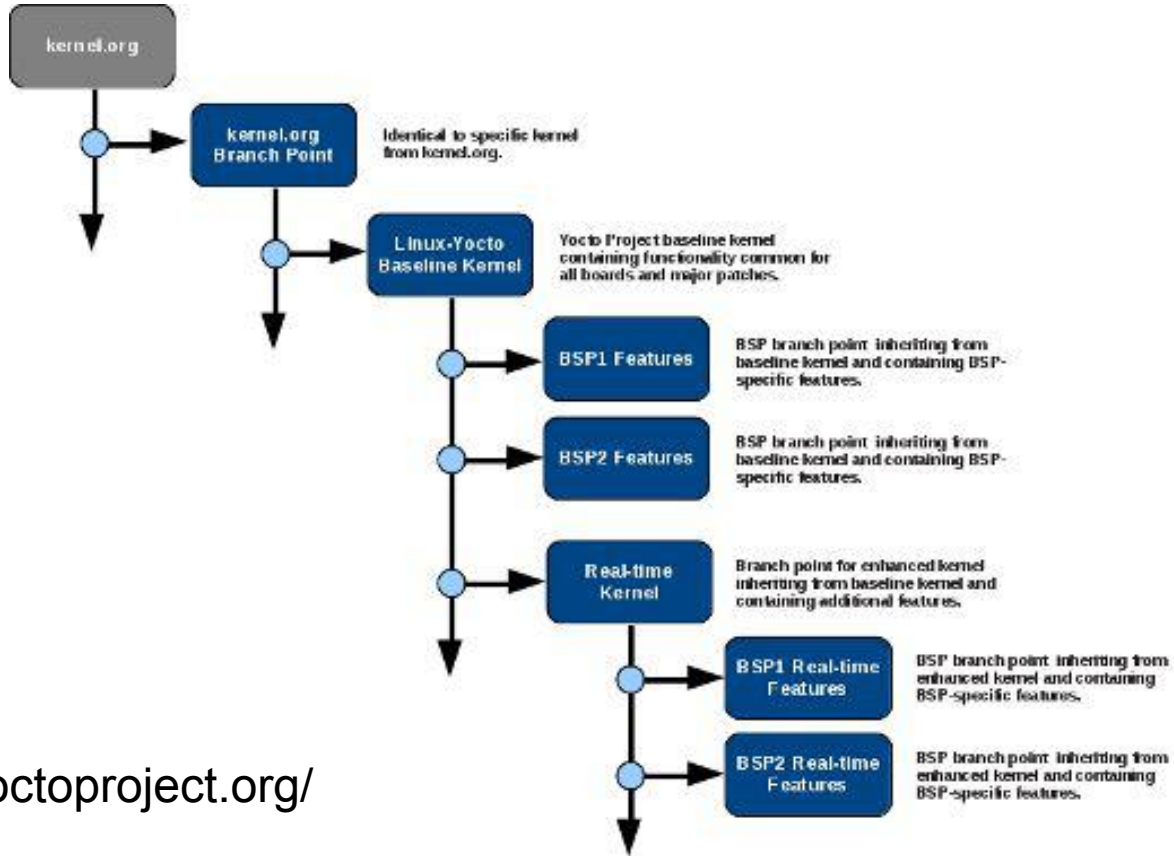
Yocto Kernels

This section will introduce the concept of kernel recipes

Booting Your Image with QEMU

Yocto Project Release	Kernel
2.0	linux-yocto-4.1
2.1	linux-yocto-4.4
2.2	linux-yocto-4.8

Yocto Kernel branches



<http://git.yoctoproject.org/>

Config fragments

➤ SRC_URI += smp.cfg

```
$ cat smp.cfg
```

```
CONFIG_SMP=y
```

Default configuration

- The default kernel config is called “defconfig”

`SRC_URI += defconfig`

- Search path for relative files in `SRC_URI`

`FILESPATH = "${FILE_DIRNAME}/${PF}:\`

`${FILE_DIRNAME}/${P}:\`

`${FILE_DIRNAME}/${PN}: \`

`${FILE_DIRNAME}/files:${FILE_DIRNAME}"`

Config fragments

➤ **SRC_URI += "file://smp.cfg"**

\$ cat smp.cfg

CONFIG_SMP=y

menuconfig

- **Used to temporarily change configurations with devshell and “make menuconfig”**
- **You can now do it with:
“bitbake -c menuconfig linux-yocto”**
- **You must manually copy back the resulting .config**

Kernel exercise with menuconfig

- Change `CONFIG_LOCALVERSION` to “-SCaLE15x”
\$ `cd $HOME/yocto`
\$ `source poky/oe-init-build-env build`
\$ `bitbake -c menuconfig linux-yocto`
\$ `bitbake -C compile linux-yocto`
\$ `bitbake core-image-minimal`
- Test by reburning your uSD card and trying it in your BeagleBone with “`uname -r`”

Kernel exercise config fragments

- **Change CONFIG_LOCALVERSION to “-SCaLE15x”**
- **\$ cd \$HOME/yocto**
- **\$ yocto-layer create mylayer**

Please enter the layer priority you'd like to use for the layer: [default: 6]

Would you like to have an example recipe created? (y/n) [default: n]

Would you like to have an example bbappend file created? (y/n) [default: n]

New layer created in meta-mylayer.

Don't forget to add it to your BBLAYERS (for details see meta-mylayer/README).

Kernel exercise config fragments (2)

Add layer to `${TOPDIR}/conf/bblayers.conf`

```
/home/<USER>/yocto/meta-mylayer \
```

```
yocto$ mkdir -p \  
    meta-mylayer/recipes-kernel/linux
```

Kernel exercise config fragments (2)

```
$ cd meta-mylayer/recipes-kernel/linux
```

```
$ echo 'CONFIG_LOCALVERSION="-SCaLE15x"' \  
> localversion.cfg
```

Create "linux-yocto_4.8.bbappend" containing:

```
FILESEXTRAPATHS_prepend := "${THISDIR}:"
```

```
SRC_URI += "file:///localversion.cfg"
```

Kernel exercise with menuconfig

```
$ bitbake -c clean linux-yocto
```

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

- Test by reburning your uSD card and trying it in your BeagleBone

```
# uname -r
```

```
4.8.12-SCaLE15x
```

wic

This section will introduce the concept of kernel recipes

What is wic?

- wic uses *kickstart* files to describe the partition layout and the filesystems used
- Much simpler than defining your own `IMAGE_FSTYPES`
- This is an example `.wks` file:

And Example .wks file

```
# short-description: Create SD card image with a boot partition  
# long-description: Creates a partitioned SD card image. Boot files  
# are located in the first vfat partition.
```

```
part /boot --source bootimg-partition --ondisk mmcblk --fstype=vfat \  
    --label boot --active --align 4 --size 16  
part / --source rootfs --ondisk mmcblk --fstype=ext4 --label root --align 4
```

Running wic

```
$ wic create -o OUTPUTFILE \  
-e core-image-minimal sdimage-bootpart
```

- Wic can also be run in “raw mode” outside of OpenEmbedded

devtool

This section will introduce devtool

devtool

➤ Devtool automates your work flow

```
$ devtool add https://github.com/msgpack/msgpack-c.git;tag=cpp-2.1.1
```

```
$ devtool edit-recipe msgpack-c
```

```
$ devtool build msgpack-c
```

```
$ devtool build-image -p msgpack-c core-image-minimal
```

```
$ devtool deploy-target msgpack-c beaglebone
```

```
$ devtool undeploy-target msgpack-c beaglebone
```

```
$ devtool finish msgpack-c meta-mylayer
```

You can learn more from the yocto documentation:

4.3.1.1. Use devtool add to Add an Application

Alternate devtool workflow

\$ devtool search <pkgname> or

<http://layers.openembedded.org>

\$ devtool modify -x <pkgname> <sourcepath-to-extract-to>

Apply patches to the sources

Commit changes to VCS (git)

\$ devtool build <pkgname>

\$ devtool build-image <imagenamename>

\$ devtool update-recipe <pkgname>

\$ devtool finish <pkgname> <layername>

eSDK

This section will introduce the extensible Software Development Kit

Building the eSDK

➤ **You can build the eSDK like this:**

```
$ bitbake -c populate_sdk_ext core-image-minimal
```

➤ **You can build the old SDK like this:**

```
$ bitbake -c populate_sdk core-image-minimal
```

Building the eSDK

➤ **You can build the eSDK like this:**

```
$ bitbake -c populate_sdk_ext core-image-minimal
```

➤ **You can build the old SDK like this:**

```
$ bitbake -c populate_sdk core-image-minimal
```

Installing the eSDK

```
$ cd tmp/deploy/sdk
```

```
$ source ./poky-glibc-x86_64-core-image-minimal-\  
cortexa8hf-neon-toolchain-ext-2.2.1.sh
```

➤ **Which installs the eSDK in `$HOME/poky_sdk/`**

Licensing and SPDX

This section will show off the Licensing mechanism in OpenEmbedded

Simple licensing

```
LICENSE = "licname1 licname2"
```

```
LIC_FILES_CHECKSUM = "file://COPYING;md5=xxxx \  
file://licfile.txt;beginline=5;endline=29;md5=yyyy \  
file://src/module.h;endline=45;md5=zzzz \  
file://../license.html;md5=aaaa \  
..."
```

- tmp/deploy/images/beaglebone/
core-image-minimal-beaglebone.manifest

Simple licensing

```
LICENSE_FLAGS = "commercial"
```

```
LICENSE_FLAGS = "license_${PN}_${PV}"
```

```
LICENSE_FILE_WHITELIST = "commercial_gst-plugins-ugly"
```

```
LICENSE_FILE_WHITELIST = "commercial"
```

SPDX

- Can auto detect the license
- Defines a comprehensive XML format for encoding licenses in a machine readable way
- The ability to verify that licensing is compatible
- Interim step of simple common headers in all files.
- <https://spdx.org/>

*It's **not** an Embedded
Linux Distribution*

*It Creates a 
Custom One For You*



Embedded Linux Development with Yocto Project Training from The Linux Foundation

Want to learn how to use Yocto Project like a Pro?

<https://training.linuxfoundation.org/>

Embedded Linux Platform Development with Yocto Project

<http://bit.ly/elddyocto>

TIPS HINTS AND OTHER RESOURCES

The following slides contain reference material that will help you climb the Yocto Project learning curve

Common Gotchas When Getting Started

- Working behind a network proxy? Please follow this guide:
 - https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy
- Do not try to re-use the same shell environment when moving between copies of the build system
- `oe-init-build-env` script appends to your `$PATH`, it's results are cumulative and can cause unpredictable build errors
- Do not try to share `sstate-cache` between hosts running different Linux distros even if they say it works

Project Resources

- The Yocto Project is an open source project, and aims to deliver an open standard for the embedded Linux community and industry
- Development is done in the open through public mailing lists: `openembedded-core@lists.openembedded.org`, `poky@yoctoproject.org`, and yocto@yoctoproject.org
- And public code repositories:
- <http://git.yoctoproject.org> and
- <http://git.openembedded.org>
- Bug reports and feature requests
- <http://bugzilla.yoctoproject.org>

Tip: ack-grep

- **Much faster than grep for the relevant use cases**
- **Designed for code search**
- **Searches only relevant files**
 - ◆ Knows about many types: C, asm, perl
 - ◆ By default, skips .git, .svn, etc.
 - ◆ Can be taught arbitrary types
- **Perfect for searching metadata**

Tip: ack-grep

```
chris@speedy 11:34 AM /build/intro-lab/poky-dylan-9.0.2
$ bback "SRC_URI ="
documentation/ref-manual/examples/hello-single/hello.bb
6:SRC_URI = "${GNU_MIRROR}/hello/hello-single/hello.bb

documentation/ref-manual/examples/mtd-makefile/mtd-utils_1.0.0.bb
9:SRC_URI = "ftp://ftp.infradead.org/pub/mtd-utils/mtd-utils-${PV}.tar.gz"

meta/classes/bin_package.bbclass
15:# SRC_URI = "http://foo.com/foo-1.0-r1.i586.rpm;subdir=foo-1.0"

meta/classes/externalsrc.bbclass
20:SRC_URI = ""

meta/classes/gnomebase.bbclass
8:SRC_URI = "${GNOME_MIRROR}/${BPN}/${@gnome_verdir("${PV}")}/${BPN}-
```

alias bback='ack-grep --type bitbake'

TIP: VIM Syntax Highlighting

- <https://github.com/openembedded/bitbake/tree/master/contrib/vim>
- Install files from the above repo in ~/.vim/
- Add "syntax on" in ~/.vimrc

```
$ tree ~/.vim/  
/Users/chris/.vim/  
├── ftdetect  
│   └── bitbake.vim  
├── ftplugin  
│   └── bitbake.vim  
├── plugin  
│   └── newbb.vim  
└── syntax  
    └── bitbake.vim
```

TIP: VIM Syntax Highlighting

```
chris@speedy: ~ — ssh — 80x24
SUMMARY = "The basic file, shell and text manipulation utilities."
DESCRIPTION = "The GNU Core Utilities provide the basic file, shell and
text \
manipulation utilities. These are the core utilities which are expectedd
to exist on \
every system."
HOMEPAGE = "http://www.gnu.org/software/coreutils/"
BUGTRACKER = "http://debbugs.gnu.org/coreutils"
LICENSE = "GPLv3+"
LIC_FILES_CHKSUM = "file://COPYING;md5=d32239bcb673463ab874e80d47fae5044
\
file://src/ls.c;beginline=5;endline=16;md5=38b797855
ca88537b75871782a2a3c6b8"
PR = "r0"
DEPENDS = "gmp libcap"
DEPENDS_class-native = ""

inherit autotools gettext

SRC_URI = "${GNU_MIRROR}/coreutils/${BP}.tar.xz \
file://remove-usr-local-lib-from-m4.patch \
file://coreutils-build-with-acl.patch \
file://dummy_help2man.patch \
1,1 Top
```