yocto PROJECT™ DEV DAY

It's not an embedded Linux distribution –
It creates a custom one for you.

Create a Custom Embedded Linux OS for Any Embedded Device Using the Yocto Project

Scott Garman
Intel Corporation
November 8, 2012

yocto PROJECT · THE LINUX FOUNDATION

Agenda

- **Introduction to the Yocto Project**
- **Key Concepts**
  - Build System Overview & Workflow
  - Exercise 1: Poky Directory Tree Map
- **Recipes In-Depth**
  - Standard Recipe Build Steps
  - Exercise 2: Examining Recipes
- **Building and Booting an Image**
  - Exercise 3: Building Your First Linux Image
  - Exercise 4: Booting Your Linux Image Using QEMU
- **Layers and BSPs**
  - Exercise 5: Creating a Custom Layer
  - Exercise 6-7: Adding a graphical boot logo and SSH server
  - Exercise 8-9: Booting an embedded hardware board

Meet the Yocto Project

- **Embedded tools and a Linux distribution build environment**
  - Eglibc, prelink, pseudo, swabber, along with other tools
- **Support x86 (32 & 64 bit), ARM, MIPS, PPC**
- **Shares build system and core metadata (oe-core) with the OpenEmbedded community**
- **Layer architecture allows for easy re-use of code**
- **Supports use of rpm/deb/ipk binary package formats (or none at all) in your final image**
- **Releases on a 6-month cadence**
  - Latest (stable) kernel, toolchain and packages, documentation
  - App Development Tools including Eclipse plugin, ADT, hob
- **BSPs are available from numerous vendors**

It's not an embedded Linux distribution – it creates a custom one for you

Introducing the Yocto Project
→ Governance

- Open source umbrella project
- Organized under the Linux Foundation
- Split governance model
- Technical Leadership Team
- Advisory Board made up of participating organizations
-

Texas Instruments
SAKOMAN INC
LONG TERM SUPPORT INITIATIVE
WIND RIVER
HUAWEI
ENEA software
JUNIPER NETWORKS
Mentor Graphics
intel
montavista
openembedded

THE LINUX FOUNDATION

Yocto Project presentation @ OSCON 2012
© 2012 The Linux Foundation. All rights reserved
All logos are trademarks of their respective owners.

Yocto Project Build System Overview

- **OpenEmbedded (OE)** – The overall build architecture used by the Yocto Project
- **BitBake** – Task executor and scheduler
- **Metadata** – Task definitions
  - **Configuration (*.conf)** – global definitions of variables
  - **Classes (*.bbclass)** – encapsulation and inheritance of build logic, packaging, etc.
  - **Recipes (*.bb)** – the logical units of software/images to build

Yocto Project Build System Overview

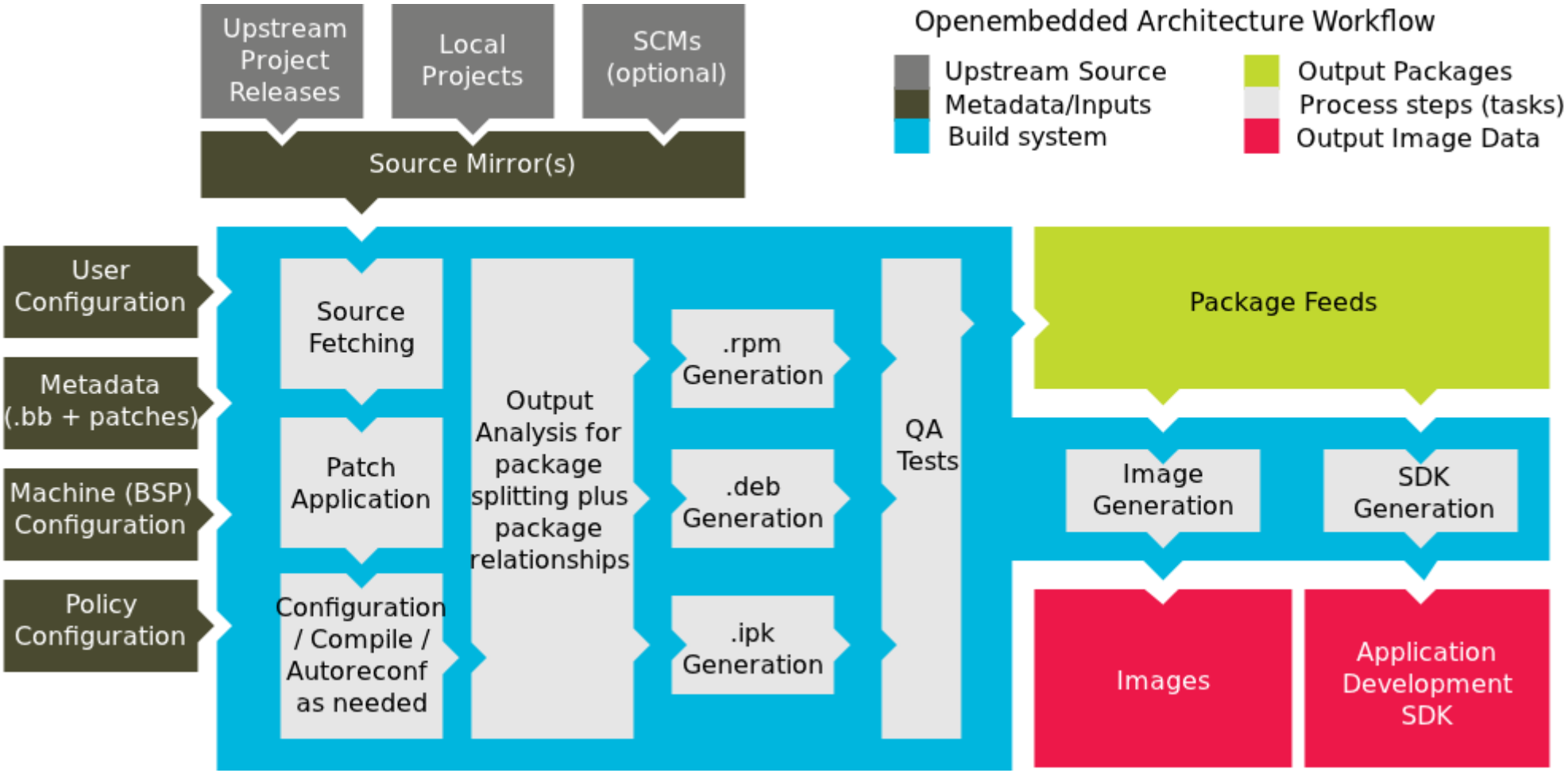
- **OpenEmbedded Core (oe-core)** – A core set of metadata shared by the OpenEmbedded and the Yocto Project
- **meta-yocto** – Reference policy/distro configuration and reference hardware support layer
- **Poky** – A pre-prepared combination of the build system components needed; also the name of our reference distro in meta-yocto

Poky = BitBake + OE-core + meta-yocto + docs

Key Concepts

- **The Yocto Project provides tools and metadata for creating custom Linux images**
- **These images are created from a repository of 'baked' recipes**
- **A recipe is a set of instructions for building packages, including:**
  - Where to obtain the upstream sources and which patches to apply
  - Dependencies (on libraries or other recipes)
  - Configuration/compilation options
  - Define which files go into what output packages

Build System Workflow



Quick Start Guide in a Slide

Obtain our sources:

- Download poky-denzil-7.0.1.tar.bz2
- tar xvjf poky-denzil-7.0.1.tar.bz2
- cd poky-denzil-7.0.1

Build one of our reference Linux images:

- source oe-init-build-env
- MACHINE=qemux86 bitbake core-image-minimal

Run the image under emulation:

- runqemu qemux86

Exercise 1: Poky Directory Tree Layout

- **Objective:** Familiarize yourself with how the Poky metadata sources are organized

- Learn where you can find **conf files**, **BitBake class files**, and **recipe files**

Log into your lab computer using the "Intro Lab" account. Password: *yoctointro*

Poky Directory Tree Map

- **bitbake:** the BitBake utility itself
- **documentation:** documentation sources
- **scripts:** various support scripts (e.g, runqemu)
- **meta/conf:** important configuration files, bitbake.conf, reference distro config, machine configs for QEMU architectures
- **meta/classes:** BitBake classes
- **meta/recipes-<xyz>:** recipes

Recipes In-Depth Agenda

- **Example Recipe: ethtool**
- **Standard Recipe Build Steps**
- **Exercise 2: Examining Recipes**

Example Recipe – ethtool_2.6.36.bb

SUMMARY = "Display or change ethernet card settings"

DESCRIPTION = "A small utility for examining and tuning the settings of your ethernet-based network interfaces."

HOMEPAGE = "http://sourceforge.net/projects/gkernel/"

LICENSE = "GPLv2+"

SRC_URI = "${SOURCEFORGE_MIRROR}/gkernel/ethtool-${PV}.tar.gz"

inherit autotools

Standard Recipe Build Steps

- Building recipes involves executing the following functions, which can be overridden when needed for customizations
  - **do_fetch**
  - **do_unpack**
  - **do_patch**
  - **do_configure**
  - **do_compile**
  - **do_install**
  - **do_package**
  - **do_build**

Exercise 2: Examining Recipes

- **meta/recipes-extended/bc/**
  - Uses LIC_FILES_CHKSUM and SRC_URI checksums
  - Note the DEPENDS build dependency declaration
- **meta/recipes-multimedia/flac/**
  - Includes custom source patches to apply to the sources
  - Customizes autoconf configure options (**EXTRA_OECONF**)
  - Overrides the **do_configure()** build step
  - Breaks up output into multiple binary packages
- **Meta/recipes-connectivity/ofono/**
  - Splits recipe into common .inc file to share metadata between multiple recipes
  - Sets a conditional build **DEPENDS** based on a distro feature
  - Sets up an init service

Exercise 3: Building a Linux Image

- **cd ~/poky-denzil-7.0.1**
- **source oe-init-build-env**
  - Sets up important environment variables
- **Set MACHINE = “qemux86” in conf/local.conf**
  - Specifies that we're building for the qemux86 target
- **bitbake core-image-minimal**
  - Builds a reference image for the qemux86 target

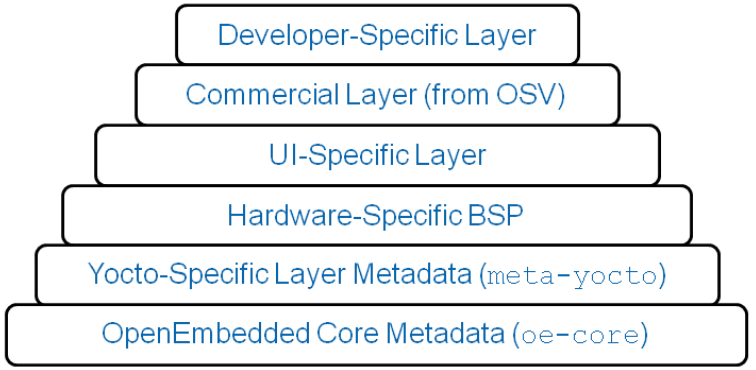
Exercise 4: Booting Your Image with QEMU

- Yocto uses QEMU, which supports all major architectures: x86(-64), arm, mips, ppc
- Simply set **MACHINE** to one of these **qemu[arch]** types in local.conf and build your image
- The **runqemu** script is used to boot the image with QEMU – it auto-detects settings as much as possible, allowing the following to boot our reference images:

runqemu qemux86

Layers Agenda

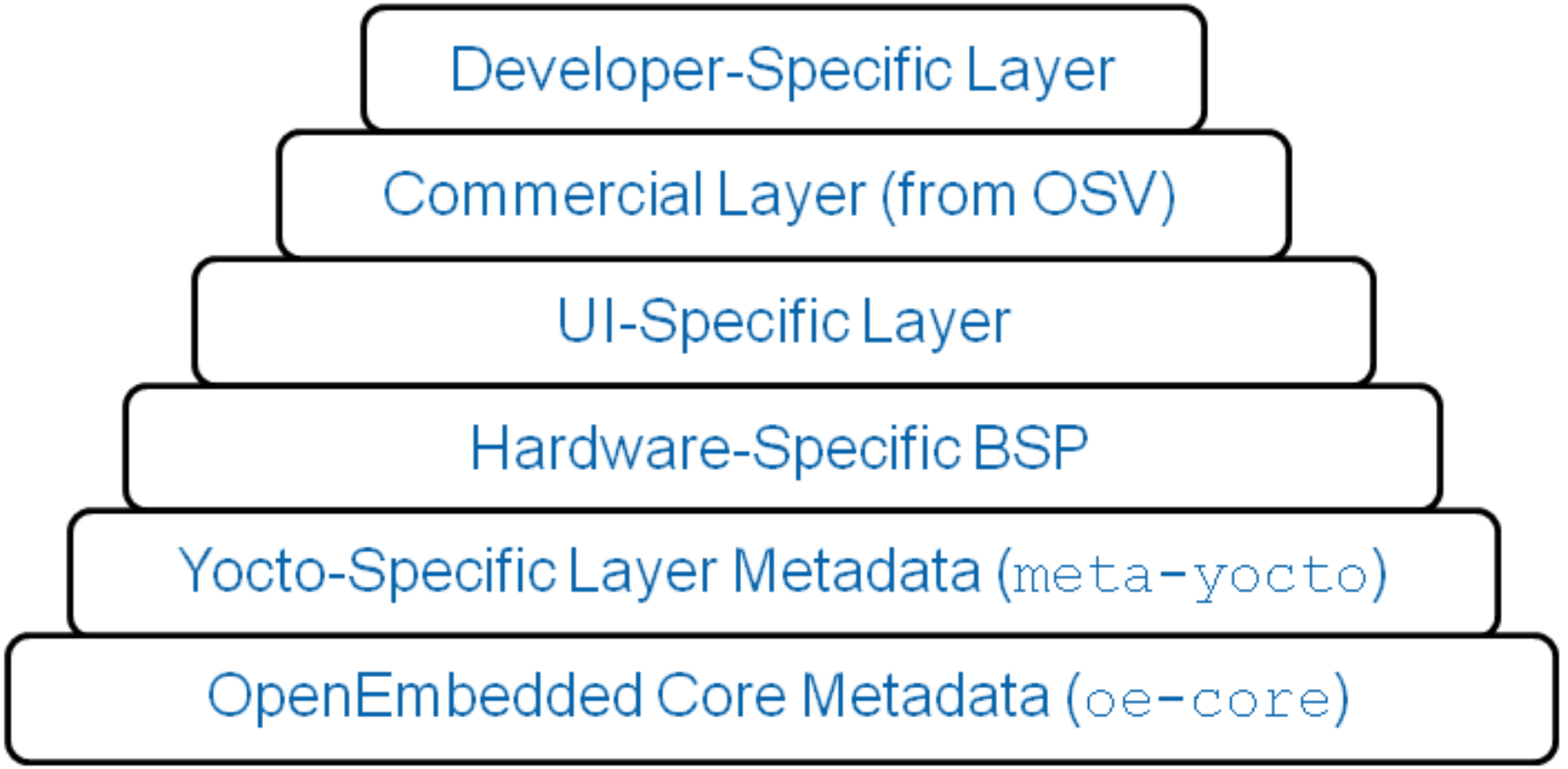
- **Introduction to Layers**
- **Stacking Customizations**
- **Adding Layers**
- **Board Support Packages**



Layers

- The Yocto Project build system is composed of layers
- A **layer** is a logical collection of recipes representing the core, a Board Support Package (BSP), or an application stack
- All layers have a priority and can override policy and config settings of the layers beneath it

Stacking Customizations



Using Layers

- Layers are added to your build by inserting them into the BBLAYERS variable within your **build/conf/bblayers.conf** file:

```
BBLAYERS = "\
    /data/poky/meta \
    /data/poky/meta-yocto \
    /data/poky/meta-my-custom-layer \
    "
```

Board Support Packages

- BSPs are layers to enable support for specific hardware platforms
- Defines machine configuration for the “board”
- Adds machine-specific recipes and customizations
  - Kernel config
  - Graphics drivers (e.g, Xorg)
  - Additional recipes to support hardware features

Exercise 5: Creating a Custom Layer

- When doing development with Yocto, do **not** simply edit files within the Poky source tree – use a custom layer for modularity and maintainability
- We will create a custom layer to hold a custom image recipe
- Let's call this layer **meta-ypdd**
- This layer must include:
  - **meta-ypdd/conf/layer.conf** file
  - Recipes directory (**meta-ypdd/recipes-ypdd/**)
  - A **meta-ypdd/README** file (basic documentation for the layer, including maintainer info)

meta-ypdd/conf/layer.conf

```
BBPATH := “${BBPATH}:${LAYERDIR}”

BBFILES := “${BBFILES} ${LAYERDIR}/recipes-*/*/*.bb \
           ${LAYERDIR}/recipes-*/*/*.bbappend”

BBFILE_COLLECTIONS += “ypdd”
BBFILE_PATTERN_ypdd := “^${LAYERDIR}/”
BBFILE_PRIORITY_ypdd = “6”
```

Exercise 6: Creating a Custom Image Recipe

- We'll derive this from **core-image-minimal**, but add support for a graphical boot logo (via **psplash**) and an SSH server (**dropbear**)
- We'll name our custom image **ypdd-image**, so the recipe will be **meta-ypdd/recipes-ypdd/images/ypdd-image.bb**
- The simplest way to add packages to a predefined image is to append them to **IMAGE_INSTALL** within the image recipe

Exercise 6: Creating a Custom Image Recipe

```
IMAGE_INSTALL = "task-core-boot ${ROOTFS_PKGMANAGE_BOOTSTRAP} \
                    ${POKY_EXTRA_INSTALL} psplash dropbear"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image

IMAGE_ROOTFS_SIZE = "8192"

# remove not needed ipkg information
ROOTFS_POSTPROCESS_COMMAND += "remove_packaging_data_files ; "
```

Exercise 7: Build and Boot Your Custom Image

- **Enable the meta-ypdd layer**
  - Edit conf/bblayers.conf and add the path to meta-ypdd to the BBLAYERS variable declaration
- **Build your custom image:**
  - bitbake ypdd-image
- **Boot the image with QEMU:**
  - runqemu qemux86 tmp/deploy/images/ypdd-image-qemux86.ext3

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**, so is not idempotent and can cause unpredictable build errors
- **Do not try to share sstate-cache between hosts running different Linux distros (now fixed in Danny) :)**

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.net
- Bug reports and feature requests
  - http://bugzilla.yoctoproject.org

Feabhas

TRAINING IN REAL-TIME

EMBEDDED DEVELOPMENT

Embedded Linux for Intel® Atom™ Training Class (Course EL-505) 10 – 14 December 2012, UK

- **Yocto – providing templates, tools and methods to help create custom Linux-based systems for the Intel® Atom™ development board**
- GStreamer – framework for creating for multimedia components
- How to build and configure a customised Linux 3 kernel
- How to construct a compact root filesystem from scratch
- How to develop and debug code for the target system using the Eclipse IDE
- How to write single and multi-threaded programs using POSIX functions

http://www.feabhas.com/

Feabhas
TRAINING IN REAL-TIME
EMBEDDED DEVELOPMENT

You should attend this course if you ...

- Are evaluating the Intel® Atom™ Processor for a future embedded Linux project
- Are porting an application to the Intel® Atom™ Processor
- Want to extend your application to use more sophisticated graphics capabilities
- Want to benefit from the wealth of tools/applications available in the traditional PC market

- Feabhas able to deliver course in any location in EMEA
- Please contact Feabhas to discuss

Feabhas
TRAINING IN REAL-TIME
EMBEDDED DEVELOPMENT

It's not an embedded Linux distribution

It creates a custom one for you.