

Creating a Custom Embedded Linux Distribution Using the Yocto Project

Part 2

Building an image, Layers, Images, Build an application

This section will introduce the concept of building an initial system image

BUILDING A FULL EMBEDDED IMAGE WITH YOCTO

Quick Start Guide in one Slide

1. Download Yocto Project sources:

- `$ mkdir yocto; cd yocto`
- `$ git clone -b kirkstone
git://git.yoctoproject.org/poky.git`

2. Build one of the reference Linux distributions:

- `$ source poky/oe-init-build-env mybuild`
- **Check/Edit 'conf/local.conf' for sanity (e.g., modify MACHINE = "qemux86-64" or MACHINE = "qemuarm64")**

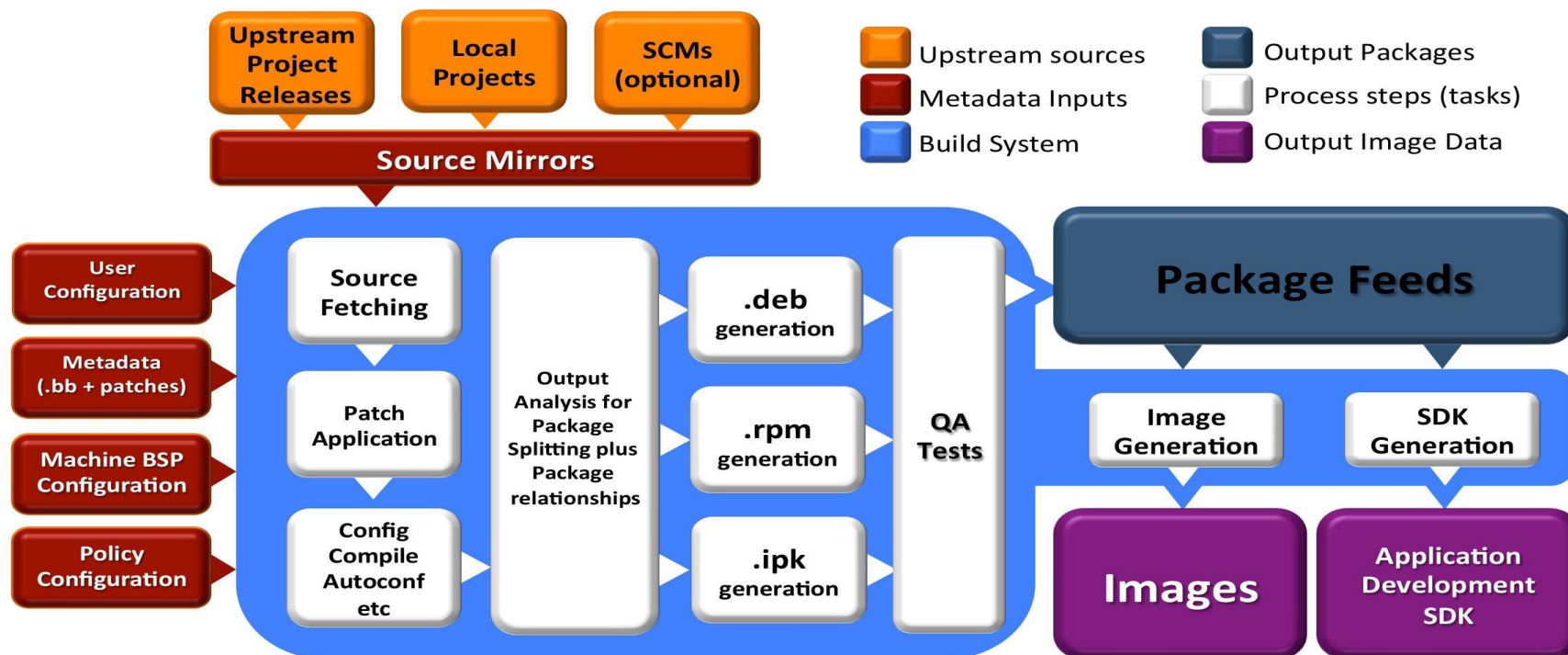
3. Build your image:

- `mybuild$ bitbake -k core-image-minimal`

4. Run the image under emulation:

- `mybuild$ runqemu qemux86-64`

Build System Workflow



Setting up a Build Directory

- Start by setting up a **build directory**

- Local configuration
- Temporary build artifacts

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

- Replace *mybuild* with whatever directory name you want to use for your project

- 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

*** We will cover how to use layers to make changes later**

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 configuration)

```
|--tmp/
```

(Build artifacts)

Poky Layout

```
$HOME/yocto/poky/  
| LICENSE  
| README  
| README.hardware  
|--bitbake/  
|--documentation/  
|--meta/  
|--meta-poky/  
|--meta-yocto-bsp/  
|--oe-init-build-env  
|--scripts/
```



(The build tool)

(oe-core)

(Poky metadata)

(Yocto reference BSPs)

(Project setup script)

(Scripts and utilities)

Building a Linux Image

■ Create a project directory using:

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

■ Configure build by editing 'local.conf'

- `$ nano $HOME/yocto/build/conf/local.conf`
- Select appropriate MACHINE type
- Set shared downloads directory (DL_DIR)
- Set shared state directory (SSTATE_DIR)

■ Build your selected image

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

■ (Detailed steps follow...)

Update Build Configuration

- Set appropriate MACHINE, DL_DIR and SSTATE_DIR
- Add the following to the bottom of 'conf/local.conf'

```
MACHINE = "qemuarm64"
```

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

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

- Notice how you can use variables in setting these values

Building an Embedded Image

- **Choose from one of the available images**
 - 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 one or two hours, 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

Booting Your Image with QEMU

- The *runqemu* script is used to boot the image with QEMU
- It auto-detects settings as much as possible, enabling the following command to boot our reference images:
 - `$ runqemu qemuarm64 [nographic]`
 - (replace `qemuarm64` with your value of `MACHINE`)
- Using *nographic* disables the video console
 - E.g., if using a non-graphical session (`ssh`)
- Your QEMU instance should boot
- Quit by closing the QEMU window
 - Alternatively, you can kill all the processes if something goes wrong
 - `$ killall qemu-system-aarch64`

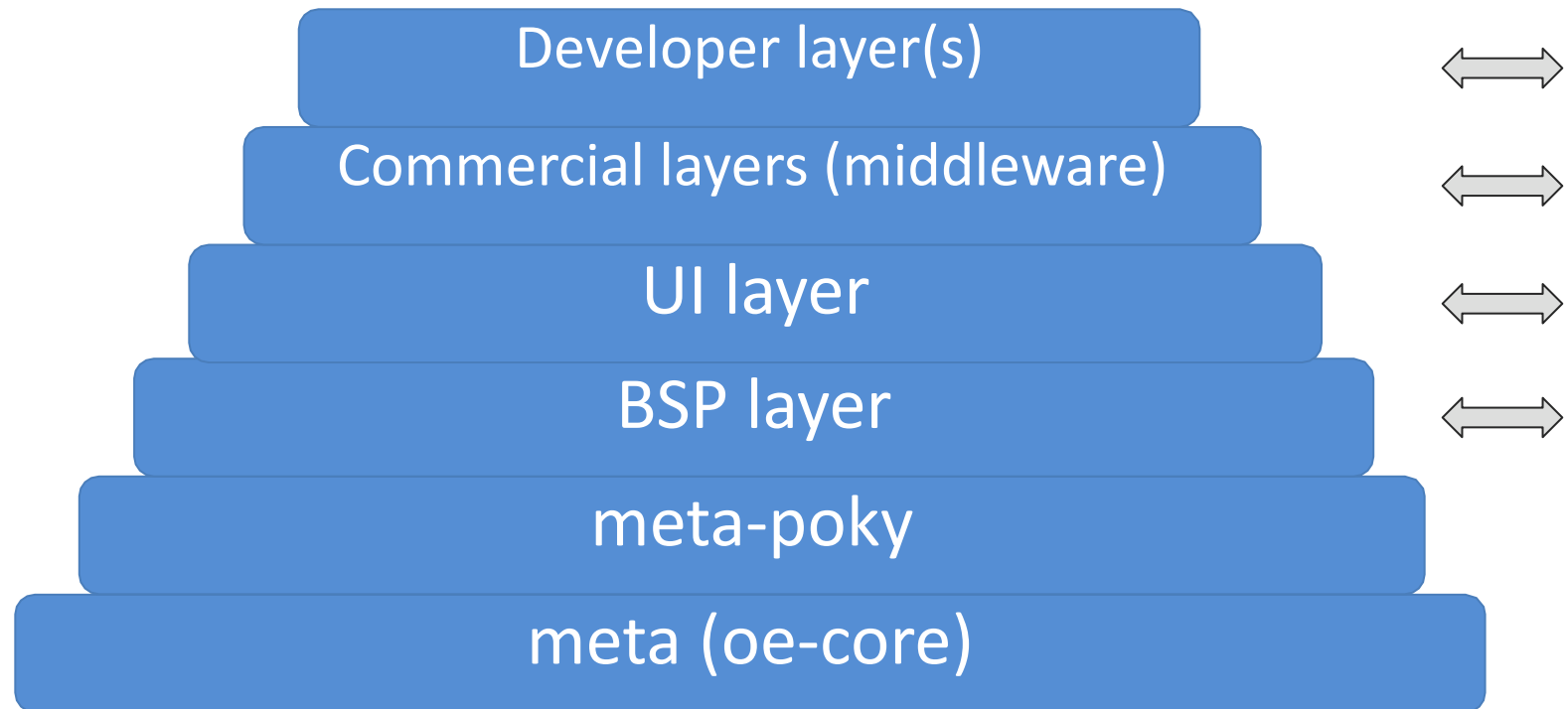
This section will introduce the concept of layers and how important they are in the overall build architecture

LAYERS

Layers

- Metadata is provided in a series of **layers** which allow you to override any value without editing the originally provided files
- A layer is a **logical collection of metadata** in the form of recipes
 - A layer is used to represent oe-core, a Board Support Package (BSP), an application stack, and your new code
- All layers have a priority and can override policy, metadata and config settings of layers with a lesser priority

Layer Hierarchy



Board Support Packages

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

Notes on using Layers

- When doing development with Yocto, do not edit files within the Poky source tree
- Use a new custom layer for modularity and maintainability
- Layers also allow you to easily port from one version of Yocto/Poky to the next version

Check Existing Layers

- Before creating a new layer, you should be sure someone has not already created a layer containing the metadata you need
- You can see the OpenEmbedded Metadata Index for a list of layers that can be used in the Yocto Project
 - `https://layers.openembedded.org/`
- You could find a layer that is identical or close to what you need

Creating a Custom Layer

- **Layers can be created manually**
 - They all start with “meta-” by convention
- **Be sure to create the directory in an area not associated with the cloned poky repository**
- **Inside your new layer folder, you need to create a ‘conf/layer.conf’ file**
 - It is easiest to take an existing layer configuration file and copy that to your layer’s conf directory and then modify the file as needed.
 - The ‘meta-yocto-bsp/conf/layer.conf’ file demonstrates the required syntax

Creating a Custom Layer

- However, using the **bitbake-layers** tool is easier

```
$ cd ~/yocto/
```

```
$ yocto$ bitbake-layers create-layer meta-mylayer
```

- This will create 'meta-mylayer' in the current directory

The New Custom Layer Layout

```
$HOME/yocto/meta-mylayer/
```

```
| COPYING.MIT
```

(The license file)

```
| README
```

(Starting point for README)

```
| --conf/
```

```
|   |--layer.conf
```

(Layer configuration file)

```
| --recipes-example/
```

(A group of recipes)

```
|   |--example/
```

(The example package)

```
|     |example_0.1.bb
```

(The example recipe)

The layer.conf File

```
# 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 += "mylayer"
BBFILE_PATTERN_mylayer = "^${LAYERDIR}/"
BBFILE_PRIORITY_mylayer = "6"
```

```
LAYERDEPENDS_mylayer = "core"
LAYERSERIES_COMPAT_mylayer = "kirkstone"
```

Adding the Layer to your Build

- Layers are added to your build by inserting them into the **BBLAYERS** variable within your 'bblayers.conf' file
 - `$HOME/yocto/build/conf/bblayers.conf`

```
BBLAYERS  += "  
    ${HOME}/yocto/poky/meta           \  
    ${HOME}/yocto/poky/meta-poky      \  
    ${HOME}/yocto/poky/meta-yocto-bsp \  
    ${HOME}/yocto/meta-mylayer       \  
"
```

Adding the Layer to your Build

- Once again, using the **bitbake-layers** tool is easier

- `build$ bitbake-layers add-layer $HOME/yocto/meta-mylayer/`

- Check the resulting layer configuration:

- `$ bitbake-layers show-layers`

- To remove a layer, you can either manually edit the 'bblayers.conf' file or use the bitbake-layers tool

- `build$ bitbake-layers remove-layer mylayer`

- Explore the bitbake-layers tool with:

- `$ bitbake-layers --help`

Build Your New Recipe

- **You can now build the new recipe**

- `$ bitbake example`

- **This will now build the 'example_0.1.bb' recipe**

- Which is found in `meta-mylayer/recipes-example/example/example_0.1.bb`

This section will introduce the concept of images – recipes which build embedded system images

IMAGES

What is an Image?

- **Building an image creates an entire Linux distribution from source**
 - Compiler, tools, libraries
 - BSP: Bootloader, Kernel
 - Root filesystem:
 - Base OS
 - Services
 - Applications
 - etc.

Available Images

- The OpenEmbedded build system provides several example images to satisfy different needs
- When you issue the *bitbake* command you provide a “top-level” recipe
 - That essentially begins the build for the type of image you want
- Display the list of directories within the source directory that contain image recipe files:
 - `$ ls meta*/recipes*/images/*.bb`

Extending an Image

- You often need to create your own image recipe in order to add new packages or functionality
- With Yocto/OpenEmbedded it is always preferable to extend an existing recipe or inherit a class
- The simplest way is to inherit the 'core-image' bbclass
- You add packages to the image by adding them to `IMAGE_INSTALL`

A Simple Image Recipe

■ Create an images directory

- `$ mkdir -p ${HOME}/yocto/build/meta-mylayer/recipes-core/images`

■ Create the image recipe

- `$ nano ${HOME}/yocto/build/meta-mylayer/recipes-core/images/mylayer-image.bb`

A Simple Image Recipe

```
DESCRIPTION = "A core image for YPDD"
LICENSE = "MIT"

# Core files for basic console boot
IMAGE_INSTALL = "packagegroup-core-boot"

# Add our desired packages
IMAGE_INSTALL += "psplash dropbear"

inherit core-image

IMAGE_ROOTFS_SIZE ?= "8192"
```

Build and Boot Your Custom Image

- **Make sure your layer is added to BBLAYERS in 'bblayers.conf'**
 - (We already did this step in a previous section manually and with `bitbake-layers add-layer`)
- **Build your custom image:**
 - `$ bitbake mylayer-image`
 - (If your `SSTATE_DIR` is configured correctly from a previous build this should be quick)
- **Boot the image with QEMU:**
 - `$ runqemu qemuarm64
tmp/deploy/images/qemuarm64/mylayer-image-
qemuarm64.ext4 [nographic]`

Build and Boot Your Custom Image

- **Verify that dropbear ssh server is present**
 - `$ which dropbear`
- **If you used the graphical invocation of QEMU, you will see the splash screen on boot**

Adding a "hello world" application to our custom image

BUILD AN APPLICATION

Building an Application

■ General procedure:

- Write the hello world application (hello.c)
- Create a recipe for the hello world application
- Modify the image recipe to add the hello world application to your image

■ What follows is the example of a simple one C file application

- Building a more complicated recipe from a tarball would specify how to find the upstream source with the SRC_URI

Add Application Code

- **For a simple one C file package, you can add the hello world application source to a directory called *files* in the *hello* package directory**

- `$ mkdir -p ${HOME}/yocto/meta-mylayer/recipes-core/hello/files`

- **Create the 'hello.c' file**

- `$ nano ${HOME}/yocto/meta-mylayer/recipes-core/hello/files/hello.c`

```
#include <stdio.h>
```

```
int main(int argc, char **argv) {  
    printf("Hello World\n");  return 0;  
}
```

Add Application Recipe

■ Write the hello world recipe

- `$ nano ${HOME}/yocto/meta-mylayer/recipes-core/hello/files/hello_0.1.bb`

```
DESCRIPTION = "Hello World example"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COREBASE}/meta/COPYING.MIT;md5=3da9cfbcb788c80a0
384361b4de20420"
S = "${WORKDIR}"
SRC_URI = "file://hello.c"

do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} hello.c -o hello
}
```

(con't next page)

Add Application Recipe

■ Write the hello world recipe

- `$ nano ${HOME}/yocto/meta-mylayer/recipes-core/hello/files/hello_0.1.bb`

(con't from previous page)

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

Add Application to the Image

- **Modify the image recipe to add the hello world application to your image**

```
DESCRIPTION = "A core image for YPDD"
LICENSE = "MIT"
# Core files for basic console boot
IMAGE_INSTALL = "packagegroup-core-boot"

# Add our desired packages
IMAGE_INSTALL += "psplash dropbear hello"

inherit core-image

IMAGE_ROOTFS_SIZE ?= "8192"
```

Build and Test Application

■ Now (re)build your image recipe

- `$ bitbake mylayer-image`
- `'hello_1.0.bb'` will be processed because it is in your custom layer and referenced in your image recipe

■ Boot your image using *runqemu*, as before:

- ```
$ runqemu qemuarm64
tmp/deploy/images/qemuarm64/mylayer-image-
qemuarm64.ext4 [nographic]
```

## ■ You should be able to type “hello” at the command line and see “Hello World”



# 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`, its 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