



Advanced Class

Henry Bruce, Marco Cavallini, Stephano Cetola, Sean Hudson, Joshua Lock,
Scott Murray, Tim Orling, Khem Raj, David Reyna, Marek Vasut

**Yocto Project Developer Day •
Portland • 26 October 2017**

Advanced Class

- **Class Content (download these slides!):**
 - https://wiki.yoctoproject.org/wiki/DevDay_Portland_2018
- **Requirements:**
 - Wireless
 - SSH (Windows: e.g. “putty”)
Wireless Registration:

Agenda – The Advanced Class

9:00- 9:15	Opening session
9:15- 9:30	Account setup , What's New
9:30-10:15	Kernel Modules with eSDKs
10:15-10:30	Morning Break
10:30-11:15	DT overlays
11:15-12:00	Devtool et. al. #1
12:00-12:45	Lunch
12:45- 1:45	Package Feeds
1:45- 2:15	Yocto Project - Rarely asked questions
2:30- 2:45	Afternoon Break
2:45- 3:15	Maintaining your Yocto Project Distribution
3:15- 4:00	Devtool et. al. #2
4:00- 4:30	A User's Experience
4:30- 5:00	Recipe specific sysroots
5:00- 5:30	Forum, Q and A



Class Account Setup

Yocto Project Dev Day Lab Setup

- **The virtual host's resources can be found here:**
 - Your Project: `"/scratch/poky/build-qemuarm"`
 - Extensible-SDK Install: `"/scratch/sdk/qemuarm"`
 - Sources: `"/scratch/src"`
 - Poky: `"/scratch/poky"`
 - Downloads: `"/scratch/downloads"`
 - Sstate-cache: `"/scratch/sstate-cache"`
- **You will be using SSH to communicate with your virtual server.**

FYI: How class project was prepared

```
$  
$ cd /scratch  
$ git clone -b rocko git://git.yoctoproject.org/poky.git  
$ cd poky  
$ bash # safe shell  
$ ./scratch/poky/oe-init-build-env build  
$ echo "MACHINE = \"qemuarm\"" >> conf/local.conf  
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\"" >> conf/local.conf  
$ echo "DL_DIR = \"/scratch/downloads\"" >> conf/local.conf  
$ echo "IMAGE_INSTALL_append = \" gdbserver openssh libstdc++ \  
    curl \"" >> conf/local.conf  
$ # Capture the build into a Bitbake/Toaster database  
$ # Build the project  
$ bitbake core-image-base  
$ exit # clean shell  
$  
$
```

NOTE: Clean Shells!

- **We are going to do a lot of different exercises in different build projects, each with their own environments.**
- **To keep things sane, you should have a new clean shell for each exercise.**
- **There are two simple ways to do it:**
 1. Close your existing SSH connection and open a new one
-- or --
 2. Do a “bash” before each exercise to get a new sub-shell, and “exit” at the end to remove it, in order to return to a pristine state.



Activity One

Yocto Project 2.5 (Sumo)

Yocto Project – What is new in 2.5 Sumo

- **Yocto Project 2.5 Theme: Stability**
 - Yocto Project Compliance 2.0
 - New website (yoctoproject.org)
 - Meltdown/Spectre patches (*IA so far*)
 - Automated Tooling
 - Automated Testing
 - Performance Improvements
 - Kernel 4.15
 - Multi-kernel support in same image
 - New machines (RiscV)
 - Icecream/icecc distributed compiler
 - Mason, Ninja



Activity Two

Kernel Modules with eSDKs

Marco Cavallini

Kernel modules with eSDKs – Overview

- **The Extensible SDK (eSDK) is a portable and standalone development environment , basically an SDK with an added bitbake executive via devtool.**
- **The “devtool” is a collection of tools to help development, in particular user space development.**
- **We can use devtool to manage a new kernel module:**
 - Like normal applications is possible to import and create a wrapper recipe to manage the kernel module with eSDKs.

Kernel modules with eSDKs – Compiling a kernel module

- **We have two choices**
- **Out of the kernel tree**
 - When the code is in a different directory outside of the kernel source tree
- **Inside the kernel tree**
 - When the code is managed by a KConfig and a Makefile into a kernel directory

Kernel modules with eSDKs – Pro and Cons of a module outside the kernel tree

- **When the code is outside of the kernel source tree in a different directory**
- **Advantages**
 - Might be easier to handle modifications than modify it into the kernel itself
- **Drawbacks**
 - Not integrated to the kernel configuration/compilation process
 - Needs to be built separately
 - The driver cannot be built statically

Kernel modules with eSDKs – Pro and Cons of a module inside the kernel tree

- **When the code is inside the same directory tree of the kernel sources**
- **Advantages**
 - Well integrated into the kernel configuration and compilation process
 - The driver can be built statically if needed
- **Drawbacks**
 - Bigger kernel size
 - Slower boot time

Kernel modules with eSDKs – The source code

```
#include <linux/module.h>
#include <linux/kernel.h>

static int __init hello_init(void)
{
    printk("When half way through the journey of our life\n");
    return 0;
}

static void __exit hello_exit(void)
{
    printk("I found that I was in a gloomy wood\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module from the Divine Comedy");
MODULE_AUTHOR("Dante Alighieri");
```

Kernel modules with eSDKs – The Makefile

```
obj-m += hellokernel.o
```

```
SRC := $(shell pwd)
```

```
all:
```

```
$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules
```

```
modules_install:
```

```
$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
```

- ***KERNEL_SRC*** is the location of the kernel sources.
- This variable is set to the value of the ***STAGING_KERNEL_DIR*** within the module class (*module.bbclass*)
- Sources available on <https://github.com/koansoftware/simplest-kernel-module.git>

Kernel modules with eSDKs – Devtool setup

- **Start a new Shell!** Otherwise, the existing bitbake environment can cause unexpected results
- Here is how the eSDK was prepared for this class account:

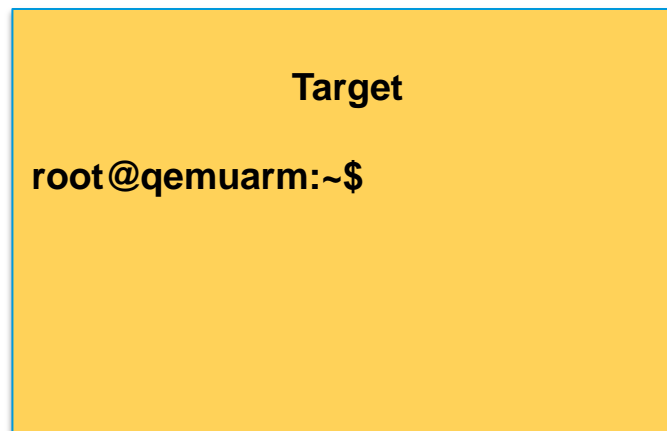
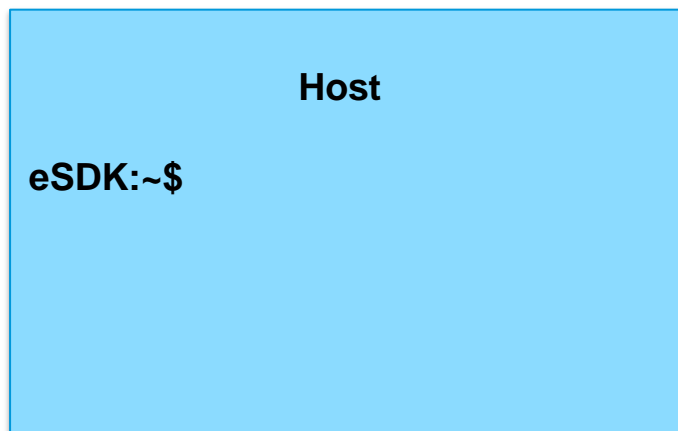
< DO NOT ENTER THE FOLLOWING COMMANDS : ALREADY EXECUTED >

```
$ bitbake core-image-base -c populate_sdk_ext
$ cd /scratch/working/build-qemuarm/tmp/deploy/sdk/
$ ./poky-glibc-x86_64-core-image-base-armv5e-toolchain-ext-2.4.sh \
    -d /scratch/sdk/qemuarm -y
$ cd /scratch/sdk/qemuarm
$ . environment-setup-armv5e-poky-linux-gnueabi
$ devtool modify virtual/kernel
$
```

- This installed the eSDK into:
/scratch/sdk/qemuarm

Kernel modules with eSDKs – Overview

- Starting from now we are using the **eSDK** and not the project
- During this exercise we using two different machines
 - The **HOST** containing the eSDK (providing devtool)
 - The **TARGET** running the final qemuarm image



Kernel modules with eSDKs – Globalsetup

- Open two terminal windows and setup the eSDK environment in each one

```
$ cd /scratch/sdk/qemuarm
$ bash # safe shell
$ source environment-setup-armv5e-poky-linux-gnueabi
...
SDK environment now set up;
additionally you may now run devtool to perform development tasks.
Run devtool --help for further details.
$
```

Kernel modules with eSDKs – build the target image

- After you have setup the eSDK environment, build an image

```
$ devtool build-image
```

- **This will create a new image into:**

```
/scratch/sdk/qemuarm/tmp/deploy/images/qemuarm
```

Kernel modules with eSDKs – build the target image

- Run the image to check if everything is OK
- This will run the QEMU machine in the TARGET shell you were using
- Login using user: **root** (no password required)

```
$ runqemu qemuarm nographic
```

Kernel modules with eSDKs – Hooking a new module into the build

- Run the devtool to add a new recipe (on the HOST side)

```
$ devtool add --version 1.0 simplestmodule \  
  /scratch/src/kmod/simplest-kernel-module/
```

- This generates a minimal recipe in the workspace layer
- This adds EXTERNALSRC in an workspace/appends/simplestmodule_git.bbappend file that points to the sources
- In other words, the source tree stays where it is, devtool just creates a wrapper recipe that points to it
- ***Note: this does not add your image to the original build engineer's image, which requires changing the platform project's conf/local.conf***

After the add

Workspace layer layout

```
$ tree /scratch/sdk/qemuarm/workspace/  
  
/scratch/sdk/qemuarm/workspace/  
├── appends  
│   └── simplestmodule_git.bbappend  
├── conf  
│   └── layer.conf  
├── README  
└── recipes  
    └── simplestmodule  
        └── simplestmodule_git.bb
```

Kernel modules with eSDKs – Build the Module

- Build the new recipe (on the HOST side)

```
$ devtool build simplestmodule
```

*This will create the **simplestmodule.ko** kernel module*

*This downloads the kernel sources (already downloaded for you):
linux-yocto-4.12.12+gitAUTOINC+eda4d18ce4_67b62d8d7b-r0 do_fetch*

Kernel modules with eSDKs – Deploy the Module

- *Get the target's IP address from the target serial console*

```
root@qemuarm:~# ifconfig
```

- **In the eSDK (HOST) shell, deploy the output**
(the target's ip address may change)

```
$ devtool deploy-target -s simplestmodule root@192.168.7.2
```

- *NOTE: the '-s' option will note any ssh keygen issues, allowing you to (for example) remove/add this IP address to the known hosts table*

Kernel modules with eSDKs – Deploy Details

- In the target (qemuarm), observe the result of deployment

```
devtool_deploy.list          100%   108          0.1KB/s  00:00
devtool_deploy.sh           100%  1017          1.0KB/s  00:00
./
./lib/
./lib/modules/
./lib/modules/4.12.12-yocto-standard/
./lib/modules/4.12.12-yocto-standard/extra/
./lib/modules/4.12.12-yocto-standard/extra/hellokernel.ko
./usr/
./usr/include/
./usr/include/simplestmodule/
./usr/include/simplestmodule/Module.symvers
./etc/
./etc/modprobe.d/
./etc/modules-load.d/
```

NOTE: Successfully deployed

/scratch/sdk/qemuarm/tmp/work/qemuarm-poky-linux-gnueabi/simplestmodule/

Kernel modules with eSDKs – Load the Module

- **In the target (qemuarm), load the module and observe the results**

```
root@qemuarm:~# depmod -a
```

```
root@qemuarm:~# modprobe hellokernel
```

```
[ 874.941880] hellokernel: loading out-of-tree module taints kernel.
```

```
[ 874.960165] When half way through the journey of our life
```

```
root@qemuarm:~# lsmod
```

Module	Size	Used by
hellokernel	929	0
nfsd	271348	11

Kernel modules with eSDKs – Unload the Module

- In the target (qemuarm), unload the module

```
root@qemuarm:~# modprobe -r hellokernel
[ 36.005902] I found that I was in a gloomy wood

root@qemuarm:~# lsmod
Module                Size  Used by
nfsd                   271348  11
```

Kernel modules with eSDKs – automatic load of the module at boot

- In the target (qemuarm), edit the file below and add a new line containing the module name 'hellokernel'

```
root@qemuarm:~# vi /etc/modules-load.d/hello.conf  
  
< insert the following line and save >  
  
hellokernel
```

- Then reboot the Qemu machine and verify

```
root@qemuarm:~# reboot
```

Questions



Activity Three

DT overlays
Marek Vasut

Device Tree

- **Data structure describing hardware**
- **Usually passed to OS to provide information about HW topology where it cannot be detected/probed**
- **Tree, made of named nodes and properties**
 - **Nodes can contain other nodes and properties**
 - **Properties are a name-value pair**
 - **See https://en.wikipedia.org/wiki/Device_tree**
- **DT can contain cycles by means of phandles**
 - **phandles provide simple references to device definitions (e.g. “<&L2>” = level 2 cache definition)**
 - **phandles can be used to reference objects in different trees (e.g. use that predefined cache type)**

Device Tree Example

- **arch/arm/boot/dts/arm-realview-eb-a9mp.dts**

```
/dts-v1/;
#include "arm-realview-eb-mp.dtsi"
/ {
    model = "ARM RealView EB Cortex A9 MPCore";
    [...]
    cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        enable-method = "arm,realview-smp";
        A9_0: cpu@0 {
            device_type = "cpu";
            compatible = "arm,cortex-a9";
            reg = <0>;
            next-level-cache = <&L2>;
        };
    };
    [...]
    &pmu {
        interrupt-affinity = <&A9_0>, <&A9_1>, <&A9_2>, <&A9_3>;
    };
};
```

Problem – Variable hardware

- **DT started on machines the size of a little fridge**
 - **HW was mostly static**
 - **DT was baked into ROM, optionally modified by bootloader**
- **DT was good, so it spread**
 - **First PPC, embedded PPC, then ARM ...**
- **There always was slightly variable hardware**
 - **Solved by patching DT in bootloader**
 - **Solved by carrying multiple DTs**
 - **Solved by co-operation of board files and DT**
 - **^ all that does not scale**

Problem – Variable hardware – 201x edition

- **Come 201x, variable HW became easy to make:**
 - Cheap devkits with hats, lures, capes, ...
 - FPGAs and SoC+FPGAs became commonplace ...
 - => Combinatorial explosion of possible HW configurations
- **Solution retaining developers' sanity**
 - Describe only the piece of HW that is being added
 - Combine these descriptions to create a DT for the system
 - Enter DT overlays

Device Tree Overlays

- DT: Data structure describing hardware
- DTO: necessary change(s) to the DT to support particular feature
 - Example: an expansion board, a hardware quirk,...
- Example DTO: vendor='hello', devicetype='dto' (no magic)

```
/dts-v1/;
/plugin/;
/ {
    #address-cells = <1>;
    #size-cells = <0>;
    fragment@0 {
        reg = <0>;
        target-path = "/";
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;
            hello@0 {
                compatible = "hello,dto";
                reg = <0>;
            };
        };
    };
};
```

Advanced DTO example

- Enable USB port, ETH port (over “gmii” channel)

```
/dts-v1/;
/plugin/;
[...]
    fragment@2 {
        reg = <2>;
        target-path = "/soc/usb@ffb40000";
        __overlay__ {
[...]
                status = "okay";
        };
    };

    fragment@3 {
        reg = <3>;
        target-path = "/soc/ethernet@ff700000";
        __overlay__ {
[...]
                status = "okay";
                phy-mode = "gmii";
        };
    };
};
```

DTO Hands-on

- Use pre-prepared meta-dto-microdemo layer
- meta-dto-demo contains:
 - Kernel patch with DTO loader with ConfigFS interface
 - Kernel config fragment to enable the DTO and loader
 - Demo module
 - Demo DTO source (hello-dto.dts)
 - core-image-dto-microdemo derivative from core-image-minimal with added DTO examples and DTC

DTO Example Layer Tree

```
\-- meta-dto-microdemo
  |-- conf
  |   |-- layer.conf
  |-- recipes-core
  |   |-- images
  |       |-- core-image-dto-microdemo.bb
  |-- recipes-kernel
  |   |-- hello-dto-dto
  |       |-- files
  |           |-- hello-dto.dts
  |           |-- hello-dto-dto_0.1.bb
  |-- hello-dto-mod
  |   |-- files
  |       |-- COPYING
  |       |-- hello-dto.c
  |       |-- Makefile
  |       |-- hello-dto-mod_0.1.bb
  |-- linux
  |   |-- files
  |       |-- 0001-ARM-dts-Compile-the-DTS-with-symbols-enabled.patch
  |       |-- 0002-OF-DT-Overlay-configfs-interface-v7.patch
  |       |-- enable-dtos.cfg
  |-- linux-yocto_4.12.bbappend
```

`" dtc hello-dto-mod hello-dto-dto "`

`"install -m 0644 *.dts ${D}/lib/firmware/dto/"`

`Debug messages for module load, remove`

`"+DTC_FLAGS := -@"`

`Patch in "overlay-configfs" (*)`

`CONFIG_OF_OVERLAY=y`
`CONFIG_OF_CONFIGFS=y`

DTO Hands-on 1/2

- **Add meta-dto-demo to bblayers.conf BBLAYERS:**

```
$ echo "BBLAYERS += \"/scratch/src/dto/meta-dto-microdemo\" \" \" \
  >> conf/bb_layers.conf
$ echo "MACHINE = \"qemuarm\" \" \" >> conf/local.conf
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\" \" \" >> conf/local.conf
$ echo "DL_DIR = \"/scratch/downloads\" \" \" >> conf/local.conf
```

- **Rebuild virtual/kernel and core-image-dto-microdemo**

```
$ bitbake -c cleansstate virtual/kernel
$ bitbake core-image-dto-microdemo
```

- **Start the new image in QEMU (*login: root, no password*)**

```
$ runqemu qemuarm nographic
```

- **(CTRL-A x to quit QEMU)**

DTO Hands-on 2/2

- **Compile DTO**

```
$ dtc -I dts -O dtb /lib/firmware/dto/hello-dto.dts > \  
    /tmp/hello-dto.dtb
```

- **Load DTO**

```
$ mkdir /sys/kernel/config/device-tree/overlays/mydto  
$ cat /tmp/hello-dto.dtb > \  
    /sys/kernel/config/device-tree/overlays/mydto/dtbo
```

- **Confirm DTO was loaded**

```
# ls /proc/device-tree  
... hello@0 ...  
# ls /sys/kernel/config/device-tree/overlays/mydto  
dtbo      path      status  
# cat /sys/kernel/config/device-tree/overlays/mydto/status  
Applied  
#
```

- **Unload DTO**

```
# rmdir /sys/kernel/config/device-tree/overlays/mydto
```

DTO encore

- DTOs can be used to operate SoC+FPGA hardware
- Done using FPGA manager in Linux (load firmware into ASIC)

```
fragment@0 {
    reg = <0>;
    /* controlling bridge */
    target-path = "/soc/fpgamgr@ff706000/bridge@0";
    __overlay__ {
        #address-cells = <1>;
        #size-cells = <1>;
        region@0 {
            compatible = "fpga-region";
            #address-cells = <2>;
            #size-cells = <1>;
            ranges = <0 0x00000000 0xff200000 0x00080000>;
            firmware-name = "fpga/bitstream.rbf";
            fpga_version@0 {
                compatible = "vendor,fpgablock-1.0";
                reg = <0 0x0 0x04>;
            };
        };
    };
};
```

DTO Overlay Patch, DTO Workflows

- **Why is the configs overlay support in a patch?**
 - It is not being accepted into mainline kernel because of the potential security risk (i.e. manufactures accidentally ship it in a production device and do not lock it down)
- **Example using Uboot**
 - Create DT baseline, point uboot to it
 - Create DTO's for each board variation to customize via uboot
- **Example for newbie using DTO's to prepare and debug DT's**
 - Debug the DTO with the manual configs overlay
 - Add the DTO to uboot and then debug the hardware and kernel modules
 - Turn the DTO into a pure DT for production

DTO Extra

- **Recommended way to load custom DTO at boot**
 - There are sysvinit, systemd, custom scripts, or add to uboot, however there is no standard for that currently
- **Top debugging techniques, tricks and tips:**
 - The “/proc/device-tree” is the image of the live DT, to check if your overlay was applied properly
 - The configs interface provides you a status information for each overlay
- **Top common user errors and gothchas**
 - Usually typos in the DT (no verification in DT compiler)
 - Not exact “compatible” match between DTO/kernel module
- **Anything special about DTO's vis-à-vis Yocto Project**
 - Not really, they are orthogonal
 - The “dtc” compiler is part of openembedded-core layer

DTO Extra

- Examples of DTO's in production systems
 - RaspberryPi (hats)
 - Beaglebone (cape manager)
 - Some Intel boards (e.g. via ACPI)
- **More on DT at:**
 - <https://www.devicetree.org/>
- **ePAPR specification of DT:**
 - https://elinux.org/images/c/cf/Power_ePAPR_APPROVED_v1.1.pdf
- **Contact:**
 - Contact: Marek Vasut marek.vasut@gmail.com
 - https://sched.ws/hosted_files/elciotna18/c1/elc-2018.pdf



Activity Four

Devtool – Part 1

Tim Orling, Sean Hudson, David Reyna

devtool – Overview

- **devtool is a collection of tools to aid developer workflow:**
 - Create, update, modify recipes in the build environment
 - Streamlines development by performing repetitive tasks via tinfoil (wrapper around bitbake) and `recipetool`.
 - Application development in user space (with eSDK)
- **The extensible SDK (eSDK) is a portable and standalone development environment , basically an SDK with an added bitbake executive via devtool.**
- **The eSDK runs in a Linux environment, but we it can be run in a Mac OS X (or Windows) environment using CROPS (Docker containers).**
 - <https://github.com/crops/extsdk-container>
 - <https://github.com/crops/docker-win-mac-docs/wiki>
 - <https://hub.docker.com/r/crops/extsdk-container/>
- ***NOTE: this session will focus on the layer maintainer/system integrator's workflow (build environment), the second session this afternoon will focus on eSDK application developer workflow***

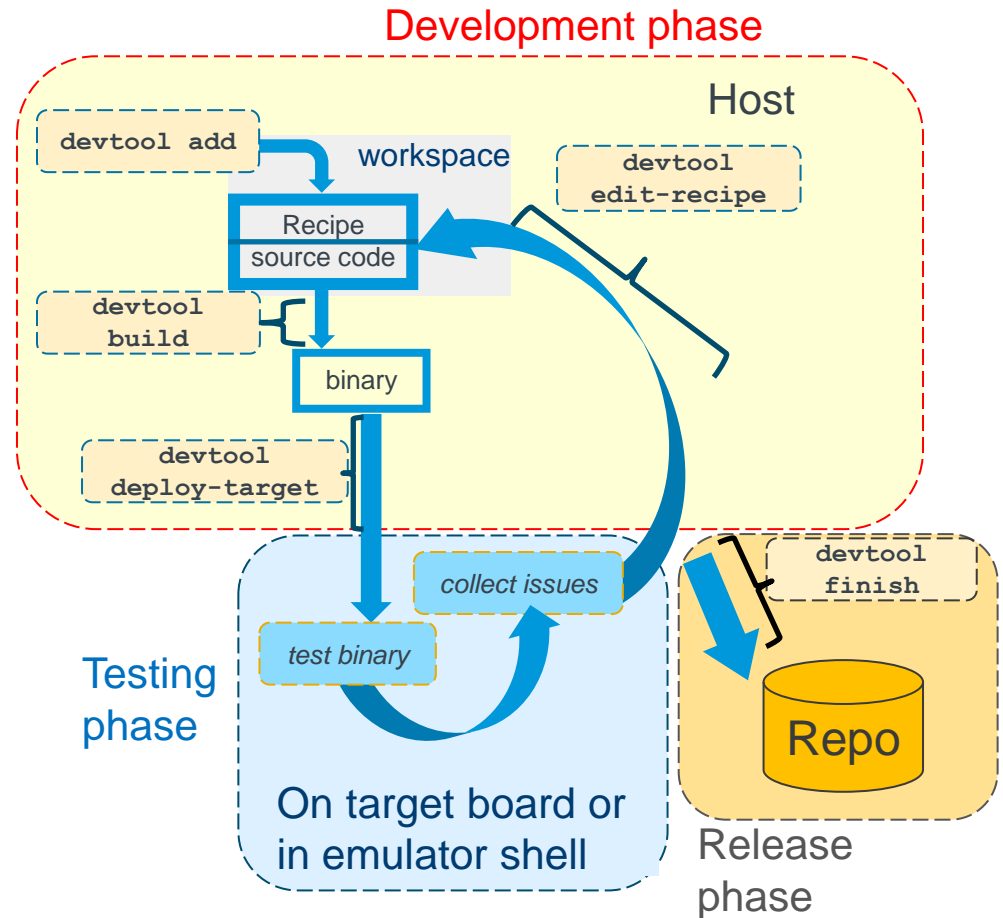
`devtool` – Types of projects currently supported

- Autotools (`autoconf` and `automake`)
- Cmake
- `qmake`
- Plain `Makefile`
- Out-of-tree kernel module
- Binary package (i.e. “-b” option)
- Node.js module
- Python modules that use `setuptools` or `distutils`
- ROS (Robot Operating System v1)

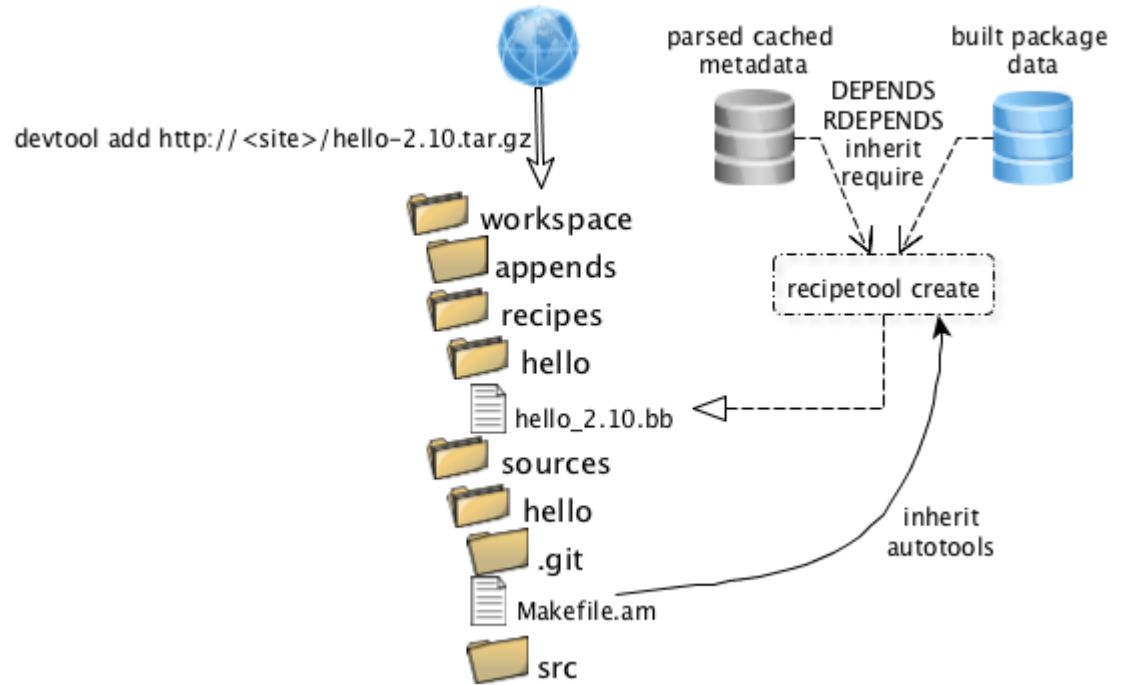
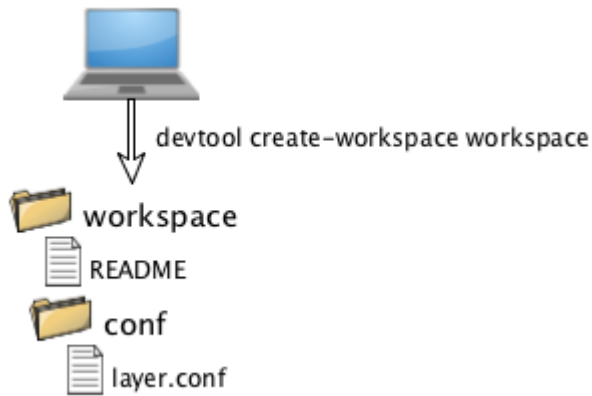
devtool – Overview

Example Workflow

- Create a new recipe
- Create workspace layer
- Build it
- Deploy to target
- Testing testing testing
- Correct any findings in the recipe
- Merge new recipe into layer



devtool – Overview



`devtool` - Baking in a sandbox

Class will cover these use cases for devtool

- **Development cycle with a new recipe**
 - Create a recipe from a source tree, then we will build, deploy, edit, build, and deploy
- **Development cycle to modify the source of existing recipe**
 - Extract recipe and source, then edit, build, and deploy
- **Development cycle to upgrade an existing recipe**
 - Extract recipe and source, then edit, build, and deploy

devtool - subcommands

Beginning work on a recipe:

<code>add</code>	Add a new recipe
<code>modify</code>	Modify the source for an existing recipe
<code>upgrade</code>	Upgrade an existing recipe

Getting information:

<code>status</code>	Show workspace status
<code>search</code>	Search available recipes

Working on a recipe in the workspace:

<code>build</code>	Build a recipe
<code>edit-recipe</code>	Edit a recipe file in your workspace
<code>configure-help</code>	Get help on configure script options
<code>update-recipe</code>	Apply changes from external source tree to recipe
<code>reset</code>	Remove a recipe from your workspace

Testing changes on target:

<code>deploy-target</code>	Deploy recipe output files to live target machine
<code>undeploy-target</code>	Undeploy recipe output files in live target
<code>build-image</code>	Build image including workspace recipe packages

Advanced:

<code>create-workspace</code>	Set up workspace in an alternative location
<code>extract</code>	Extract the source for an existing recipe
<code>sync</code>	Synchronize the source tree for an existing recipe

Activity 0 – Setup our build environment

- **Start a new Shell!** Otherwise, the existing bitbake environment can cause unexpected results

```
<open new clean shell>  
$ cd /scratch
```

- **Source the build environment**

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

- ***Use the pre-populated downloads and sstate-cache***

```
$ sed -i -e 's:#DL_DIR ?= "${TOPDIR}/downloads":DL_DIR ?=  
"/scratch/downloads":g' conf/local.conf
```

```
$ sed -i -e 's:#SSTATE_DIR ?= "${TOPDIR}/sstate-cache":SSTATE_DIR ?=  
"/scratch/sstate-cache":g' conf/local.conf
```

- ***Set machine to qemuarm***

```
$ sed -i -e 's:#MACHINE ?= "qemuarm":MACHINE ?= "qemuarm":g'  
conf/local.conf
```

- ***(Optional) On limited compute machines (laptop), use rm_work***

```
$ echo 'INHERIT += "rm_work"' >> conf/local.conf
```

Activity 0 – Setup a new layer to receive our work

- **Best practice** is to use a function/application layer, so let's create one

```
$ pushd ..  
$ bitbake-layers create-layer meta-foo  
$ popd
```

- Add our new layer to our configuration

```
$ bitbake-layers add-layer ../meta-foo
```

- *Setup complete! Time to create a new recipe...*

Activity 1: Add a new recipe

- **Optional: build core-image-minimal first (this has already been done in the class VMs)**

```
$ pwd
(should be in /scratch/poky/build-gemu)
$ devtool build-image core-image-minimal
```

- Add our new recipe

```
$ devtool add nano \
    https://www.nano-editor.org/dist/v2.7/nano-2.7.4.tar.xz
```

- *Examine what devtool created:*

```
$ ls workspace
$ find workspace/recipes
$ pushd workspace/sources/nano/
$ git log
$ popd
```

- *Now we are ready to build it:*

```
$ devtool build nano
$ devtool build-image core-image-minimal
```

Activity 1: Add a new recipe (continued)

- Run our image in QEMU

```
$ runqemu slirp nographic qemuarm  
(login as root, no password)
```

- *Run our application*

```
$ nano  
(CTRL-x to exit nano)
```

- *Examine where it was installed*

```
$ ls /usr/bin/nano  
$ exit  
(CTRL-a x to exit qemu)
```


Activity 1: Add a new recipe (continued)

- “Publish” our recipe

```
$ devtool finish nano ../meta-foo
```

- *Clean up*

```
$ rm -rf workspace/sources/nano
```

- *Add our recipe to our image*

```
$ echo 'CORE_IMAGE_EXTRA_INSTALL += "nano"' >> conf/local.conf
```

- *Profit!*

Activity 2: Modify a recipe

- **Sanity check**

```
$ pwd
```

(should be in /scratch/build-devtool)

- ***Re-inforce what we just learned***

```
$ devtool add hello \
```

```
    https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
```

```
$ devtool build hello
```

```
$ devtool build-image core-image-minimal
```

```
$ runqemu slirp nographic qemuarm
```

(login as root, no password)

- ***Run our new application***

```
$ hello
```

Hello, world!

Activity 2: Modify a recipe (continued)

- **Sanity check**

```
$ pwd  
(should be in /scratch/build-devtool)
```

- ***Re-inforce what we just learned***

```
$ devtool add hello \  
    https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz  
$ devtool build hello  
$ devtool build-image core-image-minimal  
$ runqemu slirp nographic qemuarm  
(login as root, no password)
```

- ***Run our new application***

```
$ hello  
Hello, world!  
$ exit  
(Ctrl-a x to exit qemu)
```

- ***Publish our new recipe and cleanup***

```
$ devtool finish hello ../meta-foo  
$ rm -rf workspace/sources/hello
```

Activity 2: Modify a recipe (continued)

- Might need to let git know who you are

```
$ git config --global user.email you@example.com
$ git config --global user.name "Your Name"
```
- **Modify our application's source code**

```
$ devtool modify hello
$ pushd workspace/sources/hello
$ sed -i -e 's:"Hello, world!":"Hello, Portland!":g'
src/hello.c
$ git log
$ git add src/hello.c
$ git commit -m "Change world to Portland"
```
- ***Build and run our modified application***

```
$ devtool build-image core-image-minimal
$ runqemu slirp nographic qemuarm
(login as root, no password)
$ hello
Hello, Portland!
$ exit
(CTRL-a x to exit qemu)
```

Activity 2: Modify a recipe (continued)

- *Publish our modified source and recipe and cleanup*

```
$ popd
$ devtool finish hello ../meta-foo
$ rm -rf workspace/sources/hello
```

- **Review what changed**

```
$ pushd ../meta-foo/recipes-hello/hello
$ ls
$ cat hello_2.10.bb
$ cat hello_%.bbappend
$ cat hello/0001-Change-world-to-Portland.patch
$ popd
```

- **Cleanup**

```
$ rm -rf workspace/sources/hello
```

- ***Profit!***

Activity 3: Upgrade a recipe

- *(Optional) Add meta-openembedded/meta-oe layer*

```
$ pushd /scratch
$ git clone https://github.com/openembedded/meta-openembedded
$ popd
$ bitbake-layers add-layer ../meta-openembedded/meta-oe
```

- *Upgrade to a specific version*

```
$ devtool upgrade nano --version 2.9.4
```

- *Review what changed*

```
$ tree workspace/recipes/nano
  (meta-oe version)
    workspace/recipes/nano
    └─ nano_2.9.4.bb
    └─ nano.inc
  (scratch version)
    workspace/recipes/nano
    └─ nano_2.9.4.bb
$ cat workspace/recipes/nano/nano_2.9.4.bb
```

Activity 3: Upgrade a recipe (continued)

- Test our upgraded application

```
$ devtool build-image core-image-minimal
```

```
$ runqemu slirp nographic qemuarm
```

```
(login as root, no password)
```

```
$ nano
```

```
(Ctrl-x to exit nano)
```

```
$ exit
```

```
(Ctrl-a x to exit qemu)
```

- Publish our work and cleanup

```
$ devtool finish nano ../meta-foo
```

```
$ rm -rf workspace/sources/nano
```

- *Profit!*

devtool - References

1. Yocto devtool documentation

<http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#using-devtool-in-your-workflow>

2. Tool Author Paul Eggleton's ELC Presentation:

http://events.linuxfoundation.org/sites/events/files/slides/yocto_project_dev_workflow_elc_2015_0.pdf

3. Trevor Woerner's Tutorial

<https://drive.google.com/file/d/0B3KGzY5fW7laQmgxVXVTSDJHeFU/view?usp=sharing>

4. Sean Hudson's YP Dev Day Presentation (more focused on eSDK workflow):

https://wiki.yoctoproject.org/wiki/images/f/f6/Yocto_DevDay_Advanced_Class_Portland.pdf

5. Instructor's ELC Presentation:

https://elinux.org/images/e/e2/2017_ELC_-_Using_devtool_to_Streamline_your_Yocto_Project_Workflow.pdf

<https://www.youtube.com/watch?v=CiD7rB35CRE>



Activity Five

On Target Development using Package Feeds

Stephano Cetola

Package Feed Overview

- **Tested package types: rpm and ipk**
- **For rpm packages, we now use DNF instead of smart**
- **Setting up a package feed is EASY**
 - stephano.cetola@linux.intel.com
 - @stephano approves this message
- **Signing your packages and package feed is doable**
- **Two major use cases:**
 - On target development (faster and smarter)
 - In the field updates (YMMV)

On Target Development – Better, Faster, Stronger

Topics

- **Setting up a package feed**
- On target example – AWS + Beaglebone Black
- Signing package feeds
- Keeping your code secure
- The future of package feeds

Setting up a package feed - Target Setup

- **Install Package Management on the target**
 - EXTRA_IMAGE_FEATURES += " package-management "
- **Set the correct package class**
 - PACKAGE_CLASSES = "package_rpm"
- **Customize the feed (optional)**
 - PACKAGE_FEED_URI = <http://my-server.com/repo>
 - PACKAGE_FEED_BASE_PATHS = "rpm"
 - PACKAGE_FEED_ARCHS = "all armv7at2hf-neon beaglebone"
- **Edit /etc/yum.repos.d/oe-remote-repo.repo (optional)**
 - enabled=1
 - metadata_expire=0
 - gpgcheck=0

Setting up a package feed

- Publish a repo, index the repo, and...

```
$ bitbake core-image-minimal
...
$ bitbake package-index
...
$ twistd -n web --path tmp/deploy/rpm -p 5678
[-] Log opened
[-] twistd 16.0.0 (/usr/bin/python 2.7.12) starting up.
[-] reactor class: twisted.internet.epollreactor.EPollReactor.
[-] Site starting on 5678
```

- You are now running a web server on port 5678

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- **On target example – AWS + Beaglebone Black**
- Signing package feeds
- Keeping your code secure
- The future of package feeds

Caveats

- **Running bitbake world can take some time**
 - You may want to update your repo as needed
- **Serve the repo from a build machine**
 - Or simply rsync to a webserver
- **Do not forget to run `bitbake packge-index`**
 - Package index will not auto-update
- **Good practice is to dogfood your repo**

Understanding RPM Packages and repomd.xml

- **repomd == Repo Metadata**
 - This is the “package index”
- **Repository Tools**
 - createrepo
 - rpm2cpio
 - dnf (replaces yum)
 - yum-utils (historical)
- **Important Commands**
 - rpm -qip (general info)
 - rpm -qpR (depends)
 - <https://wiki.yoctoproject.org/wiki/TipsAndTricks/UsingRPM>



On Target Demo

Beaglebone Repo on AWS

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- **Signing package feeds**
- Keeping your code secure
- The future of package feeds

Signing The Packages

- **Inherit bbclass to enable signing functionality**
 - INHERIT += "sign_rpm"
- **Define the GPG key that will be used for signing.**
 - RPM_GPG_NAME = "*key_name*"
- **Provide passphrase for the key**
 - RPM_GPG_PASSPHRASE = "*passphrase*"

Signing The Package Feed

- **Inherit bbclass to enable signing functionality**
 - INHERIT += "sign_package_feed"
- **Define the GPG key that will be used for signing.**
 - PACKAGE_FEED_GPG_NAME = "*key_name*"
- **Provide passphrase for the key**
 - PACKAGE_FEED_GPG_PASSPHRASE_FILE = "*passphrase*"

Signing The Package Feed (optional)

- ***GPG_BIN***
 - GPG binary executed when the package is signed
- ***GPG_PATH***
 - GPG home directory used when the package is signed.
- ***PACKAGE_FEED_GPG_SIGNATURE_TYPE***
 - Specifies the type of gpg signature. This variable applies only to RPM and IPK package feeds. Allowable values for the `PACKAGE_FEED_GPG_SIGNATURE_TYPE` are "ASC", which is the default and specifies ascii armored, and "BIN", which specifies binary.

Testing Packages with ptest (Optional? Not really!)

- **Package Test (ptest)**

- Runs tests against packages
- Contains at least two items:
 - 1 the actual test (can be a script or an elaborate system)
 - 2 shell script (run-ptest) that starts the test (not the actual test)

- **Simple Setup**

- `DISTRO_FEATURES_append = " ptest"`
- `EXTRA_IMAGE_FEATURES += "ptest-pkgs"`
- Installed to: `/usr/lib/package/ptest`

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- Signing package feeds
- **Keeping your code secure**
- The future of package feeds

Keeping feeds secure

- **PACKAGE_FEED_GPG_PASSPHRASE_FILE**
 - This should NOT go in your configuration as plain text.
- **Is your code proprietary?**
 - You should probably be shipping a binary in Yocto
 - `bin_package.bbclass`: binary can be `.rpm`, `.deb`, `.ipk`
- **Have you thought about DEBUG_FLAGS?**
 - See `bitbake.conf` for more details
 - The flags can be filtered or set in the recipe

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- Signing package feeds
- Keeping your code secure
- **The future of package feeds**

The Future of Package Feeds – Can We Upgrade?

- Repository
 - Switch to new source entries
 - Remove unknown (3rd party) repositories
- Package
 - Check there are no broken or renamed packages
 - Versioning: what happens when they go backwards
 - Remove and install specific packages (release dependent)
 - Remove blacklisted / obsolete and add whitelisted
- Dreaming Even Bigger...
 - Kernels, Desktops (UI), Permissions, Users, Groups



Activity Six

Yocto Project - Rarely asked questions

Khem Raj

How to add layers to Workspace

- **bitbake-layers**
 - `add-layer/remove-layer` – Add/Remove a layer to workspace
 - `show-layer` – Show current list of used layers
 - `show-recipes` – List available recipes
 - `show-appends` – List appends and corresponding recipe
 - `show-overlayed` – List overlayed recipes

Are there some Workspace helper Tools

- **bitbake-whatchanged**

- print what will be done between the current and last builds

```
$ bitbake core-image-sato
```

```
# Edit the recipes
```

```
$ bitbake-whatchanged core-image-sato
```

How to make changes in workspace

- **Prepare a package to make changes**

```
# devtool modify <recipe>
```

- **Change sources**

- Change into workspace/sources/<recipe>
- Edit

- **Build Changes**

```
$ devtool build <recipe>
```

- **Test changes**

```
$ devtool deploy-target <recipe> <target-IP>
```

- **Make changes final**

```
$ devtool finish <recipe> <layer>
```

How to enquire package information ?

- **oe-pkgdata-util** - queries the pkgdata files written out during do_package

subcommands :

lookup-pkg and	Translate between recipe-space package names runtime package names
list-pkgs	List packages
list-pkg-files	List files within a package
lookup-recipe	Find recipe producing one or more packages
package-info one or	Show version, recipe and size information for more packages
find-path	Find package providing a target path
read-value packages	Read any pkgdata value for one or more
glob	Expand package name glob expression

Use `oe-pkgdata-util <subcommand> --help` to get help on a specific command

How to run meta-data self tests (unit tests)

- **oe-selftest**

- # Script that runs unit tests against bitbake and other Yocto related tools. The goal is to validate tools functionality and metadata integrity

- List available tests

- \$ `oe-selftest -l`

- Run all tests

- \$ `oe-selftest --run-all-tests`

- Run Selective Unit Test

- \$ `oe-selftest -r devtool
devtool.DevtoolTests.test_devtool_add_fetch_simpl
e`

- <https://wiki.yoctoproject.org/wiki/Oe-selftest>

How to run image auto-test

- **Can test image (-c testimage) (-c testimage_auto)**

```
INHERIT += "testimage"  
DISTRO_FEATURES_append = " ptest"  
EXTRA_IMAGE_FEATURES_append = " ptest-pkgs"  
##TEST_SUITES = "auto"  
TEST_IMAGE_qemuall = "1"  
TEST_TARGET_qemuall = "qemu"  
TEST_TARGET ?= "simpleremote"  
TEST_SERVER_IP = "10.0.0.10"  
TEST_TARGET_IP ?= "192.168.7.2"
```

- **Testing SDK (-c testsdk and -c testsdkext)**

```
INHERIT += "testsdk"  
SDK_EXT_TYPE = "minimal"
```

- <https://www.yoctoproject.org/docs/latest/dev-manual/dev-manual.html#performing-automated-runtime-testing>

How to send Code upstream

- **create-pull-request**

Examples:

```
create-pull-request -u contrib -b joe/topic
```

- **send-pull-request**

Examples:

```
Send-pull-request -a -p pull-XXXX
```

How to Customize Distro

- Example poky-lsb

```
require conf/distro/poky.conf
require conf/distro/include/security_flags.inc
```

```
DISTRO = "poky-lsb"
DISTROOVERRIDES = "poky:linuxstdbase"
```

```
DISTRO_FEATURES_append = " pam largefile opengl"
PREFERRED_PROVIDER_virtual/libx11 = "libx11"
```

```
# Ensure the kernel nfs server is enabled
KERNEL_FEATURES_append_pn-linux-yocto = " features/nfsd/nfsd-
enable.scc"
```

```
# Use the LTSI Kernel for LSB Testing
PREFERRED_VERSION_linux-yocto_linuxstdbase ?= "4.14%"
```

How to Customize Machine

- **odroid-c2.conf**

```
#@TYPE: Machine
#@NAME: odroid-c2
#@DESCRIPTION: Machine configuration for
odroid-c2 systems
#@MAINTAINER: Armin Kuster
<akuster808@gmail.com>

require conf/machine/include/amlogic-
meson64.inc

DEFAULTTUNE ?= "aarch64"
include conf/machine/include/odroid-
default-settings.inc

EXTRA_IMAGEDEPENDS += "u-boot secure-
odroid"

KERNEL_DEVICETREE_FN = "meson-gxbb-
odroidc2.dtb"

KERNEL_DEVICETREE = "amlogic/meson-gxbb-
odroidc2.dtb"
```

- **odroid-c2-hardkernel.conf**

```
#@TYPE: Machine
#@NAME: odroid-c2-hardkernel
#@DESCRIPTION: Machine configuration for
odroid-c2 systems using uboot/kernel
from hardkernel supported vendor tree
#@MAINTAINER: Armin Kuster
<akuster808@gmail.com>

require conf/machine/odroid-c2.conf

SERIAL_CONSOLE = "115200 ttyS0"
UBOOT_CONSOLE = "console=ttyS0,115200"

KERNEL_DEVICETREE_FN odroid-c2-
hardkernel = "meson64_odroidc2.dtb"
KERNEL_DEVICETREE odroid-c2-hardkernel =
"meson64_odroidc2.dtb"
```

How to setup/use feeds ?

- **Configuring feeds in image**

```
$ PACKAGE_FEED_URIS = "http://10.0.0.10:8000/"
```

- **Start a http server in deploydir**

```
$ cd tmp/deploy/ipk
```

```
$ python3 -m http.server 8000
```

- **Run Package manager on booted target**

```
$ opkg update
```

```
$ opkg upgrade
```

Questions



Activity Seven

Maintaining Your Yocto Project Based Distribution

Scott Murray

Goals

- **Lay out some of the distribution maintenance options and their tradeoffs**
- **Discuss some potential pitfalls that may not be immediately obvious**
- **Discuss some specific build, security, and compliance maintenance tasks**

Caveats

- **I've done several distribution upgrades for customers, a few major and a few minor, but their requirements might not reflect yours, and my decisions and advice shouldn't be taken as gospel**
- **Every project has its own business requirements that will drive the decisions on things like upgrade strategy and schedule**

Distribution Maintenance Requirements

- **Most users of the Open Embedded and the Yocto Project poky releases are building sophisticated products that contain a lot more software than embedded systems of the past**
- **With extra software and features, there is an increased need to address bugs and security issues, especially with more and more products including network accessible services**
- **It is now rare that you can sell a product and not have to worry about providing customers some form of software maintenance scheme**

Distribution Maintenance Planning

- **At the moment, Yocto Project releases receive one to two years of upstream support, as only the last three releases are maintained by the project**
- **There are no long term releases (at the moment)**
- **So, you need to consider a commercial support solution (Wind River, Mentor, etc.), or you need to plan on doing it yourself**

Distribution Maintenance Planning (continued)

- **There are two approaches to maintaining your distribution yourself:**
 - Stay on the proverbial upgrade treadmill and track Yocto Project releases
 - Stick with a Yocto Project release and backport changes as required
- **The first option requires an investment into tracking upstream that may be a change for some companies' development process**
- **The difficulty of the second option starts to scale significantly with a larger number of packages and increasing length of time after release**
- **Upgrade frequency could be gated by your customer requirements, i.e. they may not want or be able to take upgrades quickly. This might influence your decision...**

Yocto Project Release Tracking

- **The selection of release at project start, or as a target for an upgrade affects the maintenance effort going forward**
- **The general recommendation from the community is to track master up until the Yocto Project release just before your target product release date**
 - This maximizes upstream support
- **A common behavior is to start a project using the release available at the time, this is less desirable, as it means you are losing some or all of the benefit of the upstream support window**

Backport Packages or Patches?

- **The community strategy is to backport patches to fix bugs or security issues, rather than upgrading packages to new versions in stable Yocto Project releases**
- **This may not work for some product requirements when you are doing the maintenance yourself**
 - Security team requirements, e.g. checking tools looking for specific versions, or customer optics
 - OpenSSL, OpenSSH, libxml2, etc.
 - Demand for new features provided by a new version of a package

Recipe Backporting Tips

- **If backporting recipes for package upgrades, keep them in a separate layer from the rest of your distribution recipes**
 - Avoids cluttering your actual distribution layer with hopefully temporary cruft
 - And keeps your distribution layer in compliance with Yocto Project expectations
- **On a distribution upgrade, you'll need to rationalize changes, and potentially remove now unnecessary backports**
 - *bitbake-layers show-overlayed*

Vendor BSP Layers

- **May tie you to a Yocto Project release unless you can invest the required effort into upgrading yourself**
- **May have a release schedule that gates upgrades**
 - e.g. meta-ti, meta-renesas
- **May provide an older kernel that gates upgrading other packages**
- **It's not always feasible, but sticking to a BSP layer that meets Yocto Project BSP requirements can hopefully avoid issues**

Other Layers

- **May not have release branches, and float on master**
 - This can cause compatibility problems if you want to use them with an older release
- **May be intermittently maintained**
- **Do some research**
 - Look at the layer's commit history to see how actively it is maintained
 - If it's hosted on github, look at activity and rating there
- **Forewarned is forearmed, you want to avoid surprises during future upgrades**

Maintaining Local Configuration

- **Keep your local metadata configuration (BSP and machine, distribution, etc.) in your own set of layers**
 - Note that they do not necessarily need to be in separate git repositories, but doing so can keep change history clearer
- **Think about using tools for layer repository management such as Android repo, myrepo, Wind River's setuptool, etc., and planning ahead on having a branching and release strategy for metadata repositories**
 - Makes checkout and build more straightforward
 - In addition to facilitating tracking your product releases, this allows upgrade development on a branch

Distribution Configuration Tips

- **Avoid local changes to oe-core, meta-poky, etc. if at all possible**
 - If something cannot be accomplished with a bbappend, it is better to work with upstream to try to come up with a solution, as carrying such changes is just one more thing that can become a time sink on a future upgrade
- **Look at pushing new recipes for FOSS packages upstream to increase the eyes looking at it and get feedback**
 - Increases probability of things "just working" on upgrade
 - But don't just throw it over the wall, seriously consider signing up as maintainer of whatever you upstream

Distribution Configuration Tips (continued)

- **Attempt to minimize local configuration changes**
 - This can be tough since there is a tension between required local customizations and associated potential maintenance burden
- **Some simple changes can give surprising problems on upgrade**
 - e.g. Changes to fsperms.txt, base-files can later result in packaging errors that do not immediately seem related
- **Changes you may need for functional requirements can result in significant effort being required down the road**
 - e.g. FIPS support. If you locally bbappend the RedHat set of OpenSSL patches, keeping things building can be time-consuming

Distribution Configuration Tips (continued)

- **Some tools/projects can increase the burden of doing an upgrade**
 - e.g. node.js, ruby, rust, go
 - If you have any local recipes to add modules, dependencies can drive recipe upgrades you might have not expected to need to do
 - as well, backporting a recipe for a package based on one of these might have a significant ripple effect
- **gcc upgrades can be painful if you have much in-house software**
 - Recent releases have been significantly improving warnings, which is a problem if you use/rely on `-Werror`
 - I've done a few upgrades where the effort required to fix in-house code issues outweighed that required for upgrading the YP release by several times
 - If you're staying on the upgrade treadmill, it's possible you may hit this every couple of years, as the pace of gcc releases has increased, and OE only carries recipes for two gcc version branches at a time. Forward porting gcc recipes is involved and considered a bad strategy

Preparing for a Distribution Upgrade

- **Can be difficult resource-wise, but consider regularly test building against master**
 - Gives a heads up on changes that impact your build
 - If you have robust upgrade strategy and are doing regular product updates, having it be part of your workflow should ease keeping up as opposed to coming in completely fresh on a new YP release
- **Look at [yoctoproject.org](http://www.yoctoproject.org) documentation on per-release changes**
 - <http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#migration>

Build-related Maintenance

- **For reproducibility and disaster recovery, it is useful to archive build output such as images and SDKs**
 - If you make use of SDKs, it is common to have significant churn during development as additions are made. Using the `SDK_VERSION` variable to number your internal SDK releases can help with reproducibility
- **Archiving the downloads directory for use as a source pre-mirror can be quite useful**
 - In some organizations, doing so can be required for either reproducibility or for building on systems without network access
 - You may need to investigate tools for binary storage to reduce disk space usage

Security-related Maintenance

- **Security fixes are a likely source of pressure for maintenance releases, knowing what possible issues affect your distribution is extremely valuable**
- **Morty and newer releases have `cve-check.bbclass`, which uses `cve-check-tool` to check built packages for CVEs**
 - Not hard to port back to older YP releases
 - Note that there is ongoing discussion about the need for a better scanning tool
- **The yocto-security mailing list is used to notify about high profile security fixes**
 - Should soon also act as a source of notifications about labelled CVE security fixes coming into the supported releases
- **Otherwise, you'll need to rely on sources such as:**
 - cve.mitre.org
 - cvedetails.com
 - Other Linux distributions' (RedHat, Debian, etc.) security notification sites

Compliance-related Maintenance

- **Planning for source and build system release upfront can save a lot of anxiety later**
- **It is recommended that you use the provided license publishing and source archiving tools**
 - <http://www.yoctoproject.org/docs/2.3.2/dev-manual/dev-manual.html#maintaining-open-source-license-compliance-during-your-products-lifecycle>
 - You may need to post-process the output to match your legal team's requirements, it is better to have an idea of what those are as early as possible
- **Similar to archiving of the downloads directory, keeping source archiving enabled and storing the output may be easier than doing special compliance collection builds, if you can manage the space requirements**

References

- [Best Practices to Follow When Creating Layers](#)

<http://www.yoctoproject.org/docs/2.3.2/dev-manual/dev-manual.html#best-practices-to-follow-when-creating-layers>

- [Making Sure Your Layer is Compatible With Yocto Project](#)

<http://www.yoctoproject.org/docs/2.3.2/dev-manual/dev-manual.html#making-sure-your-layer-is-compatible-with-yocto-project>



Activity Eight

Devtool – Part 2

Tim Orling

- **Slides will be found later today at:**
 - https://wiki.yoctoproject.org/wiki/DevDay_Portland_2018
- **We will announce when then are merged to main deck!**



Activity Nine

A User's Experience

Henry Bruce

What I'll be talking about

- **Learnings from my painful ramp on Yocto**
- **Get similar experiences from the audience**
- **Funnel these learnings into topics in the new Development Tasks Manual**
- **Review improvements in usability over the past few years**

General areas I'll be covering

- **Proxies**
- **Debugging build errors**
- **Writing recipes**
- **Recipes vs. Packages**
- **Application Development**
- **Cool things I stumbled across**
- **Improvements**

Some context

- **Started as an open source neophyte**
 - Had never really used git or dug into Linux
- **Spent six months in extreme pain**
 - Mainly due to OpenJDK
- **For the next year I was learning**
- **After 2 years I felt I could competently help others**
- **Over 3 years later, there's still so much to learn**
- **I should have taken better notes**

Proxies

- **A common problem for new users**
- **Proxy wiki page has 135k hits**
 - https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy
- **Environment variable approach covers most cases**
 - But fails when non-fetch tasks reach out to network
 - This includes most node.js recipes
 - How important is network isolation for post fetch tasks?
- **Chameleonsocks has been failsafe for me**
 - But some say this an abuse of docker
- **What's your solution?**

When things go wrong

- **You've gone through the quick start guide and have figured out how to add packages to an image**
- **You're feeling pretty good but then you get a build error.**
- **Due to many moving parts it's easy to panic when something breaks**
 - Or at least it was for me

It broke – what would have helped?

- **Nicer output from bitbake on bad directory/file names**
- **Understanding the task pipeline**
 - fetch / unpack / configure / build / install / package
- **Knowing how to generate dependency graph**
- **Decoding “magic” folder names in tmp/work**
- **Understanding recipe vs. package**
- **Knowing how to run specific task for specific recipe**
- **Knowing what’s packaged and in rootfs**

Recipes

- **Plenty of resources to writing simple recipes**
 - But then there seems to be a gap
- **Can be hard to work out what a recipe is doing**

```
pn = d.getVar('PN', 1)
metapkg = pn + '-dev'
d.setVar('ALLOW_EMPTY_' + metapkg, "1")
blacklist = [ metapkg ]
metapkg_rdepends = [ ]
packages = d.getVar('PACKAGES', 1).split()
for pkg in packages[1:]:
    if not pkg in blacklist and pkg.endswith('-dev'):
        metapkg_rdepends.append(pkg)
d.setVar('RRECOMMENDS_' + metapkg, ' '.join(metapkg_rdepends))
```

- **Walk through a couple of good citizens in oe-core?**

Recipes and packages

- **Easy to assume there is 1:1 mapping**
- **Sometimes there isn't**
 - devtool search rocks
- **Sub-packages can trip you up**
 - OpenCV vs. UPM
- **Creating sub-packages for large project seems to be the “right” pattern**
 - But I can't find obvious guidance in docs
- **Thoughts?**

Application Development

- **I was initially confused by the terminology**
 - ADT, SDK, eSDK, toolchain
- **In retrospect ADT seemed the clearest naming**
 - I'm now working on a real-time SDK
 - Yocto built Linux is our initial target platform
 - I tell my team to develop for the target using the Yocto SDK
 - Confusion all round
- **Eclipse**
 - Broken when I first tried
 - I need to get back to it

Improvements

- **eSDK and devtool**
- **Recipetool**
 - ROS support
 - Is it worth investing more, or do returns diminish?
- **Package feeds**
 - Credit to dnf (setting server means build checks if it's there)
 - But package-index is a big gotcha
- **Development Tasks Manual**
- **CROPS**
 - Who's using it?

Cool things I stumbled across

- **PACKAGECONFIG**
- **INSANE_SKIP**
- **Overrides**
- **Layer dependencies**
- **Setting package variables from outside recipe**
- **Conditional logic with python**
 - Adding package to image if its layer is present
- **What's your favorite?**



Activity Ten

Recipe Specific Sysroots

Joshua Lock

(given by Sean Hudson)

Recipe Specific Sysroots - Overview

Topics

- **Definitions**
- Determinism improvements in YP 2.3 +
- Future reproducibility work

Recipe Specific Sysroots

- **Reproducible**

- **Repeatable:** rerun a build and have it succeed (or fail) in the same way
- **Deterministic:** given the same inputs the build system should produce equivalent outputs
- **Binary reproducible:** given the same inputs the system should produce bit-for-bit identical outputs

Recipe Specific Sysroots

Reproducibility and Yocto Project

- **Repeatability** was a founding goal of the Yocto Project
 - Not as common place at the time of the project's inception
- **Determinism** of the YP build system has improved over time
 - Vast leap forward with most recent, Pyro, release
- Being able to build **binary reproducible** artefacts is a goal for future development
 - Some concrete tasks planned for 2.4



Recipe Specific Sysroots

Binary Reproducible

- **Fully deterministic build system, producing bit-for-bit identical output given the same inputs**
- **Build environment is recorded or pre-defined**
- **Mechanism for users to:**
 - Recreate the environment
 - Repeat the build
 - Verify the output matches

<https://reproducible-builds.org/>

Recipe Specific Sysroots

Yocto Project Reproducibility Features

- **DL_DIR** – shareable cache of downloads
- **Easily replicated build environment** - configuration in known locations, printed build header
- **Shared state mechanism** – reusable intermediary objects when inputs haven't changed
- **SSTATE_MIRRORS** – remotely addressable cache of
- **Uninative** – static libc implementation for use with native tools, improves sstate reuse
- **Fixed locale** – ensures consistent date/time format, sort order, etc

Recipe Specific Sysroots

Topics

- Definitions
- **Determinism improvements in YP 2.3 +**
- Future reproducibility work

Recipe Specific Sysroots

Shared sysroots – a long-standing source of non-determinism

- Shared sysroot used by YP build system until 2.3/Pyro release
- Cause of non-determinism, particularly with long-lived workspaces
 - automatic detection of items in the sysroot which weren't explicitly marked as a dependency
 - items which appear lower in common YP build graphs such as libc, kernel or common native dependencies such as glib-2.0-native

Recipe Specific Sysroots

Recipe specific sysroots improve determinism

- **per-recipe sysroot** which only includes sysroot components of explicit dependencies
- sysroot artefacts are installed into a **component specific location**
- built by hard-linking dependencies files in from their component trees
- reinstall sysroot content when the task checksum of the dependency changes
- resolves the issue of autodetected dependencies and implicit dependencies through build order

Recipe Specific Sysroots

Implementations challenges

- Artefacts in the component sysroots can include hard-coded paths – we need to be able to fix them for installed location
 - The code knows to look at certain common sites for hard-coded paths and can be taught to fixup in more locations by appending to the `EXTRA_STAGING_FIXMES` variable
- A recipe is composed of several tasks to run in the course of building its output; fetch, unpack, configure, etc.
 - Many of these tasks have task-specific dependencies, we need to re-extend the sysroot when tasks explicitly require items in the sysroot. i.e.
`do_package_write_deb` need `dpkg-native` `do_fetch` for a git repo requires `git-native`

Recipe Specific Sysroots

Implementations challenges (II)

- post-install scriptlets need to be executed for each recipe-specific sysroot
 - We handle this by installing postinst scriptlets into the recipe-specific sysroot with a postinst- prefix and running all of the scriptlets as part of the sysroot setup
- Still need to be able to replicate old shared-sysroot behaviour in certain scenarios, i.e. eSDK
 - bitbake build-sysroot recipe target takes everything in the components directory which matches the current MACHINE and installs it into a shared sysroot

Recipe Specific Sysroots

Adapting to recipe specific sysroots

Would have liked to be pain-free transition, but there is some conversion required for recipe-specific sysroots.

- fix missing dependencies – commonly native dependencies, i.e. glib-2.0-native
- `SSTATEPOSTINSTFUNCS` → `SYSROOT_PREPROCESS_FUNCS`
 - `SSTATEPOSTINSTFUNCS` are a hook to call specific functions after a recipe is populated from shared state, commonly used for fixing up paths.
 - As shared state objects will now be installed into the recipe-component location, then linked into the recipe specific sysroot, we need to be able to perform such fixes in each constructed sysroot.
 - `SYSROOT_PREPROCESS_FUNCS`: is list of functions to run after sysroot contents are staged and the right place to perform relocation in RSS world

Recipe Specific Sysroots

Adapting to recipe specific sysroots (II)

- Add `PACKAGE_WRITE_DEPS` for any postinsts requiring native tools at rootfs construction
 - YP build system tries to run preinst and postinsts at rootfs construction time, deferring any which fail to first boot.
 - Any special native tool dependencies of `pkg_preinst` and `pkg_postinst` must be explicitly listed in `PACKAGE_WRITE_DEPS` to ensure they are available on the build host at rootfs construction time.

Recipe Specific Sysroots

Unexpected consequences

- Recipe specific sysroots aggravated an existing source of non-determinism
- PATH included locations in the host for boot-strapping purposes
- Host tools were being used, where available, when native dependencies were missing

Recipe Specific Sysroots

Resolved with PATH filtering

- All required host utilities must be explicitly listed
- These are all symlinked into a directory
- PATH is then cleared and set to this filtered location
 - HOSTTOOLS: being unavailable causes an early failure (when they can't be linked in place)
 - HOSTTOOLS_NONFATAL: aren't a build failure when absent, i.e. optional tools like ccache or proxy helpers

Recipe Specific Sysroots - Overview

Topics

- Definitions
- Determinism improvements in YP 2.3 +
- **Future reproducibility work**

Recipe Specific Sysroots

Improved build system determinism

Next set our sights on the next level reproducible definition: binary reproducible builds.

Common issues that affect binary reproducibility include:

- **Compressing files with different levels of parallelism**
- **Dates, times, and paths embedded in built artefacts**
- **Timestamps of outputs changing**

Recipe Specific Sysroots

Future reproducibility work

- Layer fetcher/Workspace setup tool – to improve ease of build environment replication
- SOURCE_DATE_EPOCH – open spec to ensure consistent date/time stamps in generated artefacts
- strip-nondeterminism – post-processing step to forcibly remove traces of non-determinism
- etc...

Example Patches for Recipe Specific Sysroots

Juro Bystricky (34):

license.bbclass: improve reproducibility

classutils.py: deterministic sorting

e2fsprogs-doc: binary reproducible

python3: improve reproducibility

busybox.inc: improve reproducibility

image-prelink.bbclass: support binary reproducibility

kernel.bbclass: improve reproducibility

image.bbclass: support binary reproducibility

gmp: improve reproducibility

python2.7: improve reproducibility

attr: improve reproducibility

acl_2.25: improve reproducibility

zlib_1.2.11.bb: remove build host references

flex_2.6.0.bb: remove build host references


bash.inc: improve reproducibility

package_manager.py: improve reproducibility ...



yocto
PROJECT™

Questions and Answers



**Thank you for your
participation!**

yocto ·
PROJECT

 THE
LINUX
FOUNDATION



Appendix: Board Bring-up

MinnowBoard Max Turbot SD Card Prep

- Here is how to flash the microSD card for the MBM
- Insert the microSD card into your reader, and attach that to your host
 1. Find the device number for the card (e.g. “/dev/sdc”). For example run “dmesg | tail” to find the last attached device
 2. Unmount any existing partitions from the SD card (for example “umount /media/<user>/boot”)
 3. Flash the image

```
$ sudo dd if=tmp/deploy/images/intel-corei7-64/core-image-base-intel-corei7-64.hddimg of=<device_id> bs=1M
```
 4. On the host, right-click and eject the microSD card’s filesystem so that the image is clean

MinnowBoard Max Turbot SD Card Prep

- **Note: you can instead use the automatically generated WIC image**

1. Flash the image

```
$ sudo dd if=scratch/working/build-  
mbm/tmp/deploy/images/intel-corei7-64/core-image-base-  
intel-corei7-64.wic of=<device_id> bs=1M
```

2. Note that when the target boots, the WIC version of the image the kernel boot output does not appear on the serial console. This means that after 14 seconds of a blank screen you will then see the login prompt

MinnowBoard Max Turbot Board Bring-up

- Setting up the board connections
 1. Unpack the target
 2. Insert the provided micro-SD card (pin side up)
 3. Attach the ethernet cable from the target to the hub
 4. Attach the FTDI 6-pin connector. **The BLACK wire is on pin 1**, which has an arrow on the silk-mask and is on the center-side of the 6-pin inline connector near the microSD connector
 5. Connect the FTDI USB connector to your host
(Note: the USB serial connection will appear on your host as soon as the FTDI cable is connected, regardless if the target is powered)
- Start your host's console for the USB serial console connection
 - On Linux, you can use the screen command, using your host's added serial device (for example `/dev/ttyUSB0`):

```
$ screen /dev/ttyUSB0 115200,cs8
```

(FYI: "CTRL-A k" to kill/quit)
 - On Windows, you can use an application like "Teraterm", set the serial connection to the latest port (e.g. "COM23"), and set the baud rate to 115200 ("Setup > Serial Port... > Baud Rate...")

MinnowBoard Max Turbot Board Bring-up (2)

- Start the board
 1. Connect the +5 Volt power supply to the target
 2. You should see the board's EFI boot information appear in your host's serial console
- Run these commands to boot the kernel

```
Shell> connect -r
```

```
Shell> map -r
```

```
Shell> fs0:
```

```
Shell> bootx64
```

- You should now see the kernel boot
- At the login prompt, enter “**root**”
- *Note: see the appendix on instructions on how we create the microSD card images*

Beaglebone Black - Setup

- Create project directory, update local.conf and bblayers.conf

```
$ export INSTALL_DIR=`pwd`
$ git clone -b rocko git://git.yoctoproject.org/poky
$ source poky/oe-init-build-env `pwd`/build_beagle
$ echo 'MACHINE = "beaglebone"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver openssh"' \
  >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross-arm"' \
  >> conf/local.conf
$ bitbake core-image-base
```

- Nothing to change in bblayers.conf, beaglebone is supported in meta-yocto-bsp

BeagleBone Black - MicroSD

```
# Format blank SD Card for Beaglebone Black
$ export DISK=/dev/sd[c] <<<Use dmesg to find the actual device name
$ sudo umount ${DISK}1 <<<Note the addition of the '1'
$ sudo dd if=/dev/zero of=${DISK} bs=512 count=20
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
1,12,0xE,*
,,-
__EOF__
$ sudo mkfs.vfat -F 16 ${DISK}1 -n boot
$ sudo mkfs.ext4 ${DISK}2 -L rootfs

# Now unplug and replug your SD Card for automount
$ cd tmp/deploy/images/beaglebone
$ sudo cp -v MLO-beaglebone /media/guest-mXlApE/BOOT/MLO
$ sudo cp -v u-boot.img /media/guest-mXlApE/BOOT/
$ sudo tar xf core-image-base-beaglebone.tar.bz2 \
  -C /media/guest-mXlApE/rootfs
$ sync (flush to device, not necessary, but illustrative)
$ umount /media/guest-mXlApE/rootfs /media/guest-mXlApE/boot
```

Dragonboard 410c - Setup

- **See this URL to see instructions on how to install Yocto Project:**

<https://github.com/Linaro/documentation/blob/master/Reference-Platform/CECommon/OE.md>

- **To get a serial boot console, you will need to get a specialized FTDI cable. Here are some sources:**

<https://www.96boards.org/products/accessories/debug/>

- **For the slow GPIO bus (at 1.8V), it is recommended to use a protected and/or voltage shifting shield, for example the new Grove baseboard for the Dragonboard**