Advanced Class

Paul Barker, Marco Cavallini, Beth Flanagan, Sean Hudson, Joshua Lock, Scott Murray, Tim Orling, David Reyna, Rudi Streif, Marek Vasut

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

Advanced Class

- **Class Content:**
  - https://wiki.yoctoproject.org/wiki/DevDay_Prague_2017

- **Requirements:**
  - Wireless
  - SSH (Windows: e.g. "putty")

- **Wireless Registration:**
  - TBD

Agenda – The Advanced Class

9:00- 9:15	Opening session, What's New
9:15- 9:30	Account setup
9:30-10:15	Devtool: creating new content
10:15-10:30	**Morning Break**
10:30-11:15	DT overlays
11:15-12:00	Userspace: packaging, installation, system services
12:00- 1:00	**Lunch**
1:00- 1:45	License Compliance and Auditing
1:45- 2:15	CROPS
2:30- 2:45	**Afternoon Break**
2:45- 3:15	Maintaining your Yocto Project Distribution
3:15- 3:50	Kernel Modules with eSDKs
3:50- 4:30	Analytics and the Event System
4:30- 5:00	Recipe specific sysroots
5:00- 5:30	**Forum, Q and A**

Class Account Setup

Notes for the Advanced Class:

- **The class will be given with YP-2.4 (Rocko)**

- **Wifi Access:**
  - SSID: <TBD>
  - Password: <TBD>

- **Your account's IP access addresses**
  - SSH (password "devday"):

    `ssh ilab01@devdayXXX.yoctoproject.org`

  - HTTP:
    `devdayXXX.yoctoproject.org:8000`

Yocto Project Dev Day Lab Setup

- **The virtual host's resources can be found here:**

  - Your Project: "/scratch/working/build "

  - 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
$ ./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
$ . toaster start webport=0.0.0.0:8000
$ # Build the project
$ bitbake core-image-base

$ # When you are done ...
$ . toaster stop
$ exit
```

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 2.4 (Rocko)

Yocto Project – What is new in 2.4 Rocko

- **Yocto Project 2.4 Themes**
  - Process/Tooling/Workflow Improvements - Patchwork, Patchtest, SWAT, Error reporting, **Reproducability,** Memory Resident Bitbake now default
  - **Improving Testing/QA Automation/Coverage Efficiency** - oeselftest, Test automation, CI/AB - modernization and moving more into YP
  - Creating Leading edge Build Technology - Delivering prebuilt binaries to customers, Improve Binary/Build Reproducibility
  - Enhancing IoT Application Development - CROPS (eclipse support, dev containers), eSDK (team workflow), devtool (team workflow, extend heuristics), juci from openWRT support

Yocto Project – Release Notes

* Linux kernel 4.12, 4.10, 4.9 (LTS/LTSI), 4.4 (LTS)
* gcc 7.2
* glibc 2.26
* Significant work on binary reproducibility - >98% of packages used to build core-image-sato are now reproducible.
* Support for Vulkan 3D graphics/compute API, enabled by default in poky distro configuration
* New "distrooverrides" class to selectively turn DISTRO_FEATURES into overrides (enabling bbappends with functionality conditional upon DISTRO_FEATURES)
* New VOLATILE_LOG_DIR variable to allow making /var/log persistent
* Support for merged / and /usr with "usrmerge" DISTRO_FEATURES item
* Parallelised ipk and deb package creation for improved performance
* Go improvements:
* Python improvements:
* wic image creator enhancements:
* devtool/recipetool enhancements:
* BitBake improvements:
* Package QA improvements:
* RPM improvements

…
And so much more, including Known Issues, Security Fixes and Recipe Updates!

Activity Two

Devtool

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 will cover running it in a Mac OS X (or Windows) environment in the CROPS session (using Docker containers).**

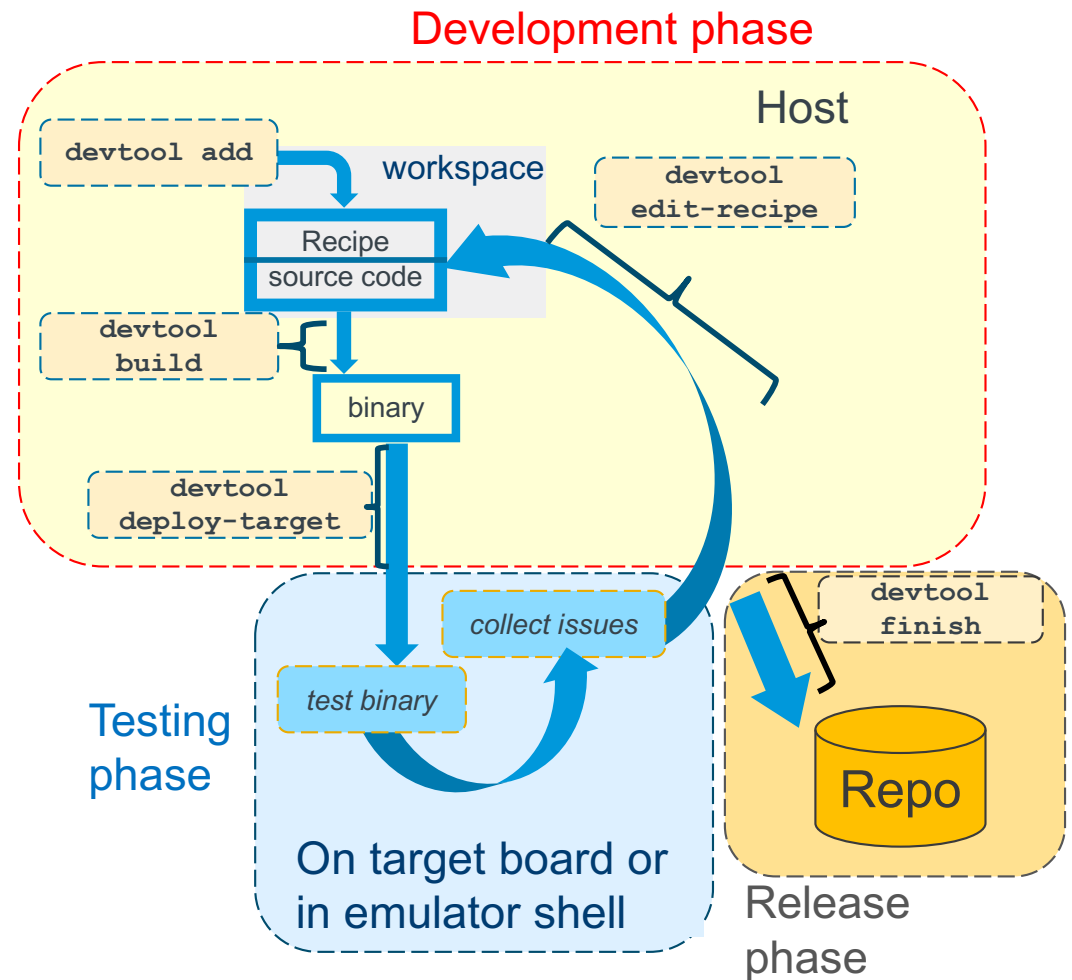- *NOTE: this session will focus on the layer maintainer/system integrator's workflow (build environment)*

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
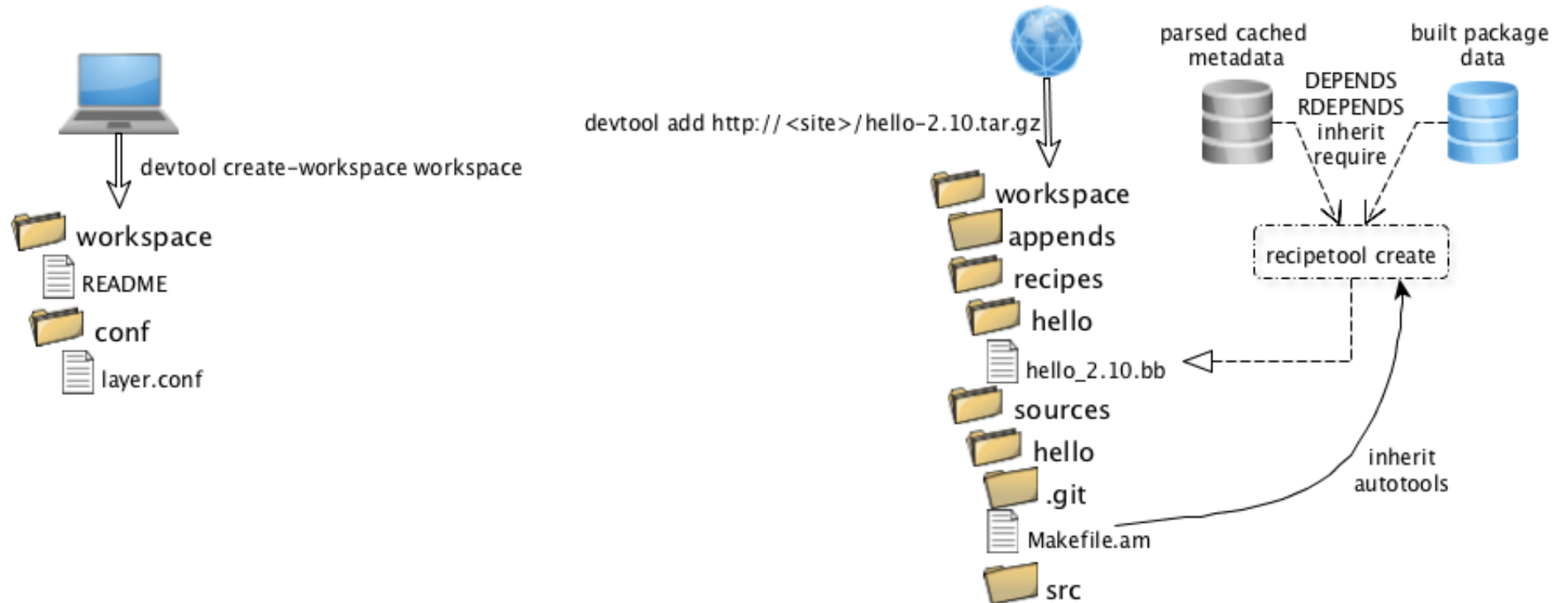- Python modules that use **setuptools** or **distutils**

`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

Development phase

Host

`devtool add`

workspace

`devtool edit-recipe`

Recipe source code

`devtool build`

binary

`devtool deploy-target`

collect issues

test binary

Testing phase

On target board or in emulator shell

`devtool finish`

Repo

Release phase

devtool – Overview

devtool create-workspace workspace

workspace
README
conf
layer.conf

devtool add http://<site>/hello-2.10.tar.gz

workspace
appends
recipes
hello
hello_2.10.bb
sources
hello
.git
Makefile.am
src

parsed cached metadata

built package data

DEPENDS
RDEPENDS
inherit
require

recipetool create

inherit autotools

`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:

```
add                     Add a new recipe
modify                  Modify the source for an existing recipe
upgrade                 Upgrade an existing recipe
```

Getting information:

```
status                  Show workspace status
search                  Search available recipes
```

Working on a recipe in the workspace:

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

Testing changes on target:

```
deploy-target           Deploy recipe output files to live target machine
undeploy-target         Undeploy recipe output files in live target
build-image             Build image including workspace recipe packages
```

Advanced:

```
create-workspace        Set up workspace in an alternative location
extract                 Extract the source for an existing recipe
sync                    Synchronize the source tree for an existing recipe
```

Activity 0 – Setup our build enviroment

- **<u>Start a new Shell!</u>** 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-devday
```

- *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

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 ..
$ yocto-layer create 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**

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

- *Profit!*

Activity 2: Modify a recipe

- ### Sanity check
  ```
  $ pwd
  (should be in /scratch/build-devday)
  ```

- ### *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-devday)
  ```

- *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, Prague!":g'
  src/hello.c
  $ git log
  $ git commit -m "Change world to Prague"
  ```

- *Build and run our modified application*
  ```
  $ devtool build-image core-image-minimal
  $ runqemu slirp nographic qemuarm
  (login as root, no password)
  $ hello
  Hello, Prague!
  $ exit
  (Ctrl-a x to exit qemu)
  ```

Activity 2: Modify a recipe (continued)

- *Publish our modifed 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-Prague.patch
  $ popd
  ```

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

- *Profit!*

Activity 3: Upgrade a recipe

- *Upgrade our nano recipe to the latest version*
  ```
  $ devtool upgrade nano --version 2.8.7
  ```

- (Hack) Fix fetch URL to allow upgrade to v2.8.x
  ```
  $ sed -i -e 's:v2.7:v2.8:g' \
  ../meta-foo/recipes-nano/nano/nano_2.7.4.bb
  ```

- *NOTE: there is a bugzilla open to add the ability to change the fetch URL [https://bugzilla.yoctoproject.org/show_bug.cgi?id=10722]*

- Cleanup our failed upgrade attempt
  ```
  $ rm -rf workspace/sources/nano
  ```

- *Actually upgrade*
  ```
  $ devtool upgrade nano --version 2.8.7
  ```

Activity 3: Upgrade a recipe (continued)

- *Review what changed*
  ```
  $ ls workspace/recipes/nano
  $ cat workspace/recipes/nano/nano_2.8.7.bb
  ```

- 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/0B3KGzY5fW7IaQmgxVXVTSDJHeFU/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 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**

- **ePAPR specification of DT:**

  - **https://elinux.org/images/c/cf/Power_ePAPR_APPROVED_v1.1.pdf**

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:**

```
/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

```
/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 Hands-on 1/2

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

```
$ ${EDITOR} conf/bblayers.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**

```
$ runqemu qemuarm
```

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
```

- **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**

```
fragment@0 {
    reg = <0>;
    /* controlling bridge */
    target-path = "/soc/fpgamgr@ff706000/bridge@0";
    __overlay__ {
        #address-cells = <1>;
        #size-cells = <1>;
        area@0 {
            compatible = "fpga-area";
            #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>;
                };
```

Activity Four

Userspace: Advanced Topics

Rudi Streif

(given by David Reyna)

See Rudi Streif's Book on Yocto Project!

- **"Embedded Linux Systems with the Yocto Project",**
  Hardcover – May 2 2016, Prentice Hall
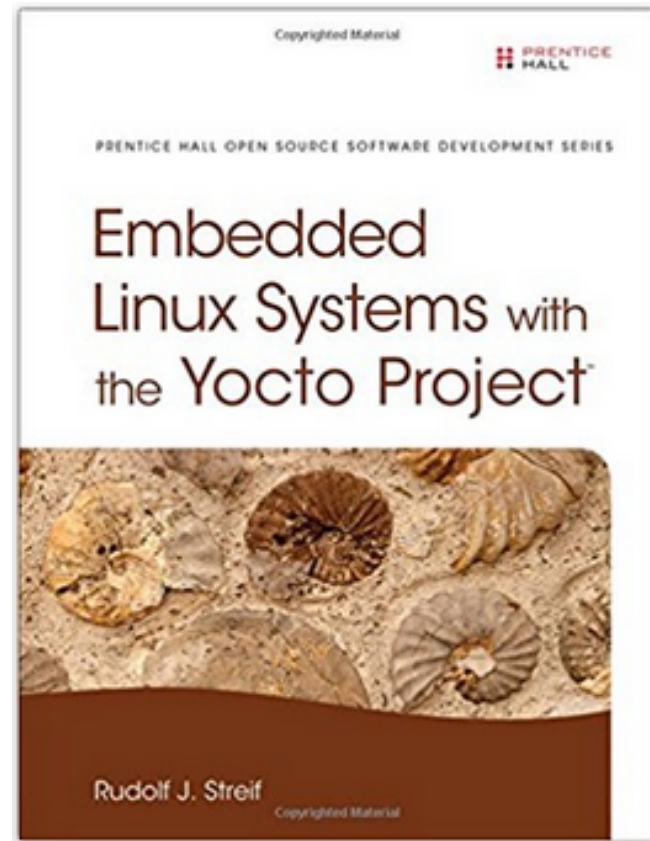
- *Amazon: #10 in Books > Computers & Technology > Hardware > Microprocessors & System Design > Embedded Systems*

What We Are Going To Do

- Most of your development work will likely be developing your own software packages, building them with the Yocto Project and installing them into a root file system built with the Yocto Project.

- Let's look at some typical tasks beyond creating the base recipe:
  - Customizing Packaging
  - Package Installation Scripts
  - System Services

Activity Setup

- **Initialize the Build Environment (IN A CLEAN SHELL)**
  - `cd /scratch/poky`
  - `source oe-init-build-env build`

- *Adjust Configuration (DONE FOR YOU)*
  - `vi conf/local.conf`

```
MACHINE = "qemuarm"
DL_DIR ?= "/scratch/downloads"
SSTATE_DIR ?= "/scratch/sstate-cache"
EXTRA_IMAGE_FEATURES ?= "debug-tweaks dbg-pkgs dev-pkgs package-management"
```

- *Build (DONE FOR YOU)*
  - `bitbake -k core-image-base`

- **Test** (login as 'root', no password needed)
  - `runqemu qemuarm nographic`

- Exit QEMU with `CTRL-A,x`

Activity Setup - Continued

- Create Local Devtool Layer "meta-uspapps"

  - `devtool create-workspace meta-uspapps`

- Observe your source file directory

  - ```
    tree /scratch/src/userspace
    /scratch/src/userspace
    \-- fibonacci
         |--- fibonacci-app
         |    |--- fibonacci-app.c
         |    \-- Makefile
         |--- fibonacci-lib
         |    |--- fibonacci-app.c
         |    |--- fibonacci.c
         |    |--- fibonacci.h
         |    |--- fibonacci-lib.bb
         |    \-- Makefile
         \-- fibonacci-srv
              |--- fibonacci-srv.bb
              |--- fibonacci-srv.init
              |--- fibonacci-srv.service
              |--- fibonacci-srv-tcp
              |--- fibonacci-srv-tcp.c
              |--- fibonacci-srv-unix
              |--- fibonacci-srv-unix.c
              |--- Makefile
              \-- Makefile.all
    ```

Packaging

- Packaging is the process of putting artifacts from the build output into one or more packages for installation by a package management system.

- Packaging is performed by the package management classes:
  - `package_rpm` – RPM style packages
  - `package_deb` – Debian style packages
  - `package_ipk` – IPK package files used by the OPK package manager

- You configure the package management in `conf/local.conf`:

  ```
  PACKAGE_CLASSES ?= "package_rpm"
  ```

  - You can add more than one of the package classes.
  - Only the first one will be used to create the root file system.
  - However, this does not install the package manager itself.

- Install the package manager in `conf/local.conf`:

  ```
  EXTRA_IMAGE_FEATURES ?= "package-management"
  ```

Package Splitting

- Packaging Splitting is the process of putting artifacts from the build output into different packages.

- Package splitting allows you to select what you need to control the footprint of your root file system.

- Package splitting is controlled by the variables:
  - `PACKAGES` – list of package names, default:

```
PACKAGES = "${PN}-dbg ${PN}-staticdev ${PN}-dev ${PN}-doc \
    ${PN}-locale ${PACKAGE_BEFORE_PN} ${PN}"
```

  - `FILES` – list of directories and files that belong into the package:

```
SOLIBS = "*.so.*"
FILES_${PN} = "${bindir}/* ${sbindir}/* ${libexecdir}/* \
    ${libdir}/lib* {SOLIBS} ${sysconfdir} ${sharedstatedir} \
    ${localstatedir} ${base_bindir}/* ${base_sbindir}/* \
    ${base_libdir}/*${SOLIBS} ${base_prefix}/lib/udev/rules.d \
    ${prefix}/lib/udev/rules.d ${datadir}/${BPN}\
    ${libdir}/${BPN}/* ${datadir}/pixmaps \
    ${datadir}/applications ${datadir}/idl ${datadir}/omf \
    ${datadir}/sounds ${libdir}/bonobo/servers"
```

Package Splitting - Continued

- The package classes process the `PACKAGES` list from left to right, producing the `${PN}-dbg` package first and the `${PN}` package last.

- The order is important, since a package consumes the files that are associated with it.

- The `${PN}` package is pretty much the "kitchen sink": it consumes all standard leftover artifacts.

- BitBake syntax only allows prepending (+=) or appending (=+) to variables:
  - Prepend `PACKAGES` – place standard artifacts into different packages
  - Append `PACKAGES` – place any leftover packages in non-standard installation directories those packages.

- The variable `PACKAGE_BEFORE_PN` allows you to insert packages right before the `${PN}` package is created.

Packaging QA

- The insane class adds plausibility and error checking to the packaging process.

- You can find a list of the checks in the Reference Manual:
  http://www.yoctoproject.org/docs/2.4/ref-manual/ref-manual.html#ref-classes-insane

- Some of the more common ones:
  - `already-stripped` – debug symbols already stripped
  - `installed-vs-shipped` – checks for artifacts that have not been packaged
  - `ldflags` – checks if `LDFLAGS` for cross-linking has been passed
  - `packages-list` – same package has been listed multiple times in `PACKAGES`

- Sometimes the checks can get into your way…
  - `INSANE_SKIP_<packagename> += "<check>"`
  - Skips <check> for <packagename>.

Example – The Fibonacci Library

- Source code in /scratch/src/userspace/fibonacci/fibonacci-lib

  - Builds static and dynamic libraries to calculate the Fibonacci series and an application to test it.

- Create development environment in the project

  - `devtool add fibonacci-lib /scratch/src/userspace/fibonacci/fibonacci-lib`

- Build the recipe

  - `devtool build fibonacci-lib`

- Add to your image (conf/local.conf):   *(vi: EoD = 'G')*

```
IMAGE_INSTALL_append = " fibonacci-lib"
```

- Build and test image (exit with CTRL-A,x)

  - `devtool build core-image-minimal`

  - ```
    runqemu qemuarm nographic
    ...
    root@qemuarm:~# fibonacci
    Enter the number of terms: 4
    First 4 terms of Fibonacci series are:
    0 1 1 2
    ```

Example – The Fibonacci Library (continued)

- Edit the recipe `meta-uspapps/recipes/fibonacci-lib/fibonacci-lib.bb` and place the `fibonacci` test application into its own package `${PN}-examples`

-
```
PACKAGE_BEFORE_PN = "${PN}-examples"

FILES_${PN}-examples = "${bindir}/fibonacci"
```

- Add to your image (`conf/local.conf`):

```
IMAGE_INSTALL_append = " fibonacci-lib fibonacci-lib-examples"
```

- Build and test image
  - `bitbake core-image-minimal`
  - `runqemu qemuarm nographic`

- See the new package (optional):
  - find tmp/work -name "*fibonacci-example*"
    tmp/work/armv5e-poky-linux-gnueabi/fibonacci-lib/1.0-r0/pkgdata/runtime-reverse/libfibonacci-examples
    tmp/work/armv5e-poky-linux-gnueabi/fibonacci-lib/1.0-r0/deploy-rpms/armv5e/libfibonacci-examples-1.0-r0.armv5e.rpm
  - Note: Automatic 'debian.bbclass' is renaming the PACKAGE 'fibonacci-lib' to RPM 'libfibonacci'

Package Installation Scripts

- Package management systems have the ability to run scripts before and after a package is installed, upgraded, or removed.

- These are typically shell scripts and they can be provided by the recipe using these variables:

  - `pkg_preinst_<packagename>`: Preinstallation script that is run *before the package is installed*.

  - `pkg_postinst_<packagename>`: Postinstallation script that is run *after the package is installed*.

  - `pkg_prerm_<packagename>`: Pre-uninstallation script that is run *before the package is uninstalled*.

  - `pkg_postrm_<packagename>`: Post-uninstallation script that is run *after the package is uninstalled*.

```
pkg_postinst_${PN}() {
#!/bin/sh
# shell commands go here
}
```

Script Skeleton

```
pkg_postinst_${PN}() {
#!/bin/sh
if [ x"$D" = "x" ]; then
    # target execution
else
    # build system execution
fi
}
```

Conditional Execution

Example – Conditionally running ldconfig

- The Fibonacci library installs a dynamic library `libfibonacci.so.1.0` on the target system in `/usr/lib`.

- For `ld` to be able to locate the library it must be added to the ld cache and its symbolic name (soname) must be linked. That is done by running `ldconfig` on the target.

- Add a post installation script to the `${PN}` package that only runs `ldconfig` when it is run on the target but not when the build system creates the root file system.

```
pkg_postinst_${PN}() {
#!/bin/sh
if [ x"$D" = "x" ]; then
    # target execution
    ldconfig
    exit 0
else
    # build system execution
    exit 1
fi
}
```

Installation for Packaging

- Makefile Installation

```
INSTALL ?= install
.PHONY: install
Install:
        $(INSTALL) -d $(DESTDIR)/usr/bin
        $(INSTALL) -m 0755 $(TARGET) $(DESTDIR)/usr/bin
```

- Recipe Installation

  - Providing/overriding the do_install task

```
do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${B}/bin/* ${D}{bindir}
}
```

  - The build system defines a series of variables for convenience:

bindir = "/usr/bin"	sysconfdir = "/etc/"
sbindir = "/usr/sbin"	datadir = "/usr/share"
libdir = "/usr/lib"	mandir = "/use/share/man"
libexecdir = "/usr/lib"	includedir = "/usr/include"

Debugging Packaging

- Check the packaging logfiles in `${WORKDIR}/temp`

- Check installation of artifacts in `${WORKDIR}/image`
    - The do_install task installs the artifacts into this directory.
    - If artifacts are missing they are packaged.

- Check packaging artifacts in `${WORKDIR}/package`
    - This where the artifacts are staged for packaging, including the ones created for the debug packages.

- Check package splitting in `${WORKDIR}/packages-split`
    - Packages and their content are staged here by package name before they are wrapped by the package manager.
    - Allows you to verify if the artifacts have indeed been placed into the correct package.

- Check created packages in `${WORKDIR}/deploy-<pkgmgr>`

Package Architecture

- The build system distinguishes packages by their hardware dependencies into three main categories:

  - Tune – Generic CPU architecture such as core2_32, corei7, armv7, etc. This is the default and typically appropriate for userspace packages.

  - Machine – Specific machine architecture. Appropriate for packages that require particular hardware features of a machine. Typically applicable to kernel, drivers, and bootloader.

  - All – Package applies to all architectures such as shell scripts, managed runtime code (Python, Lua, Java, …), configuration files, etc.

- Package architecture is controlled by the `PACKAGE_ARCH` variable:

  - Tune (default) – `PACKAGE_ARCH = "${TUNE_PKGARCH}"`

  - Machine – `PACKAGE_ARCH = "${MACHINE_ARCH}"`

  - All – `inherit allarch`

- Note: Package architecture does not simply determine into what category a package is placed but determines compiler and linker flags and other build options.

System Services

- If your software package is a system service that eventually needs to be started when the system boots you need to add the scripts and service files.

- **SysVInit**
  - Inherit `update-rc.d` class.
  - `INITSCRIPT_PACKAGES` - List of packages that contain the init scripts for this software package. This variable is optional and defaults to `INITSCRIPT_PACKAGES = "${PN}"`.
  - `INITSCRIPT_NAME` - The name of the init script.
  - `INITSCRIPT_PARAMS` - The parameters passed to `update-rc.d`. This can be a string such as `"defaults 80 20"` to start the service when entering run levels 2, 3, 4, and 5 and stop it from entering run levels 0, 1, and 6.

- **Systemd**
  - Inherit `systemd` class.
  - `SYSTEMD_PACKAGES` - List of packages that contain the systemd service files for the software package. This variable is optional and defaults to `SYSTEMD_PACKAGES = "${PN}"`.
  - `SYSTEMD_SERVICE` - The name of the service file.

Example – The Fibonacci Server

- Source code in /scratch/src/userspace/fibonacci/fibonacci-srv
  - Builds a TCP socket server listening on port 9999 for the number of terms and responds with the list of Fibonacci terms.

- Create development environment
  - `cd /scratch/poky/build`
  - `devtool add fibonacci-srv /scratch/src/userspace/fibonacci/fibonacci-srv`

- Add system service startup to the recipe
  `meta-uspapps/recipes/fibonacci-srv/fibonacci-srv.bb`

```
inherit update-rc.d systemd
INITSCRIPT_NAME = "fibonacci-srv.init"
INITSCRIPT_PARAMS = "start 99 3 5 . stop 20 0 1 2 6 ."
SYSTEMD_SERVICE = "fibonacci-srv.service"
```

- Build the recipe
  - `bitbake fibonacci-srv`

- Add to your image (conf/local.conf):

```
IMAGE_INSTALL_append = " fibonacci-srv"
```

- Build and test image
  - `bitbake core-image-minimal`
  - `runqemu qemuarm nographic`
  - `nc localhost 9999`

Changing the System Manager

- SysVInit is the default system manager for the Poky distribution.

- To use systemd add it to your `conf/local.conf` file, or better, to your distribution configuration:

```
DISTRO_FEATURES_append = " systemd"

VIRTUAL-RUNTIME_init_manager = "systemd"
```

- If you exclusively want to use systemd, you can remove SysVInit from you root file system image with:

```
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"

VIRTUAL-RUNTIME_initscripts = ""
```

Activity Five

License Compliance and Auditing

Beth 'pidge' Flanagan and Paul Barker

Togán Labs Ltd.

License Compliance and Auditing

Togán Labs Ltd

- Ireland/UK based Embedded Consultancy

- Oryx Linux and Oryx Linux Plus

- OpenChain Partner

- Made up of OpenEmbedded/Yocto Project Developers

- We REALLY like License Compliance

License Compliance and Auditing - Overview

Topics

- meta-wrong
  - all the horrible in one lovely layer
- meta-spdxscanner
  - temp fork of mainline

meta-wrong recipes

- bad-chksum

- bad-license-mix

- closed-app

- hello-lib

- mit-app

- shotgun-lic

- use-hello-lib

bad-chksum

bitbake bad-chksum -f -c cleanall

bitbake bad-chksum

bad-chksum

Two issues (one, not so obvious)

bad-chksum

Two issues (one, not so obvious)

- Bad checksum

- more ../conf/distro/wrong.conf

  - license-checksum in WARN_QA

  - devs tend to ignore bb.warns

closed-app

bitbake closed-app -f -c cleanall

bitbake closed-app

closed-app

Again, two issues (one, not so obvious)

closed-app

Again, two issues (one, not so obvious)

- build/tmp/work/armv5e-poky-linux-gnueabi/closed-app/1.0.0-r0/closed-app-1.0.0/app.py

  - wrong license

- look at recipe

  - specifically the LIC_FILES_CHKSUM

  - CLOSED ignores checksum

closed-app

A short diversion....

- CLOSED is not a license

- it's being used as a lazy way to subvert some QA warnings

- Use at your peril

bad-license-mix

more bad-license-mix/bad-license-mix_1.0.0.bb

bad-license-mix

LICENSE = "CLOSED & GPLv2"

- theoretically possible

- but we need to look at the code

  more tmp/work/armv5e-poky-linux-gnueabi/bad-license-mix/1.0.0-r0/bad-license-mix-1.0.0/app-closed.py

  more tmp/work/armv5e-poky-linux-gnueabi/bad-license-mix/1.0.0-r0/bad-license-mix-1.0.0/app.py

bad-license-mix

Solution here?

- Developer open source training

- This can sometimes be difficult to catch with copy-paste code

shotgun-lic

more shotgun-lic/shotgun-lic_1.0.0.bb

shotgun-lic

more shotgun-lic/shotgun-lic_1.0.0.bb

- LICENSE is theoretically valid

- gold star for

  - LICENSE_PATH += "${LAYERDIR}/files/licenses" in layer.conf

  - Not using CLOSED for MyWeirdProprietaryLicense

Let's look at the source!

shotgun-lic

tmp/work/armv5e-poky-linux-gnueabi/shotgun-lic/1.0.0-r0/shotgun-lic-1.0.0

- Two license files
  - COPYING
  - MyWeirdProprietaryLicense

- Let's look at the code in random_lib and another_random_lib

shotgun-lic

Uhhh….

- Which files are which license?

- Why not use DEPENDS?

  - Sometimes valid reasons why you don't
    - **don't control upstream source**
    - **but this is non-distributable**

mit-app

bitbake mit-app -f -c cleanall

bitbake mit-app

mit-app

No errors!

But does this mean nothing is wrong…?

This is where license scanning helps you!

mit-app

Two files:

- app.py
  - LIC_FILES_CHKSUM looks at this
  - License is correct

- local_lib.py
  - Not covered by LIC_FILES_CHKSUM
  - Contains a GPLv2 header

The application needs fixing!

hello-lib & use-hello-lib

```
bitbake hello-lib -f -c cleanall

bitbake use-hello-lib -f -c cleanall

bitbake use-hello-lib
```

hello-lib & use-hello-lib

No errors again!

But let's look closer…

hello-lib & use-hello-lib

Licenses:

- hello-lib: LGPLv2
  - contains hello_lib.py

- use-hello-lib: CLOSED
  - imports hello_lib

- Valid usage of an LGPL library

hello-lib & use-hello-lib

Let's look deeper:

- hello-lib contains hello_lib.py

  - License header is GPLv2 not LGPLv2

  - This is the sort of issue license scanning will detect

- So let's fix hello-lib_1.0.0.bb:

  - LICENSE = "GPLv2"

bitbake use-hello-lib (again)

hello-lib & use-hello-lib

Still no errors…

- But using GPLv2 library from a closed app is not valid

- License scanning tools won't catch this

- This is where you need to use judgement or legal advice

meta-spdxscanner

- Using the Togán Labs fork of meta-spdxscanner
  - **https://gitlab.com/toganlabs/meta-spdxscanner**
  - **requires meta-gplv2**
- Not a fan of DoSOCSv2, looking at moving
  - **scancode**
  - **fossology**
- Want to help? pidge@toganlabs.com

meta-spdxscanner

- spdx-runs/gobject-introspection.spdx
  - **find PackageLicenseInfoFromFiles**
- the license of source and the license of package is usually different
  - **This is ok**
  - **Things we don't ship (setup.py)**
  - **But we need to compare LICENSE to what we see here.**

meta-spdxscanner

- recipe states
  - **LICENSE = "LGPLv2+ & GPLv2+"**
- scan states

meta-spdxscanner

- recipe states
  - **LICENSE = "LGPLv2+ & GPLv2+"**

- scan states
  - **GPL-3.0+ & LicenseRef-Freeware & LicenseRef-MIT-style & LicenseRef-Public-domain & LicenseRef-See-file & X11 & GPL-2.0 & GPL-2.0-with-autoconf-exception & LGPL-2.0 & LGPL-2.1+ & LicenseRef-GPL-3.0+-with-bison-exception & MIT & BSD-2-Clause & LicenseRef-See-doc.OTHER & LicenseRef-GPL-exception & GPL-2.0+ & LicenseRef-FSF & LGPL-2.0+**

meta-spdxscanner

- Find the GPL files!
  - **What is scannerparser.c**
- Look in the source, see if it's something we distribute
  - **if so, we need to fix the LICENSE**
  - **maybe on a package layer**
    - **LICENSE_${PN}-dbg**

License Auditing and Compliance

Q&A

Activity Six

CROPS
Tim Orling, Brian Avery, Randy Witt, David Reyna

CROPS: Containers for Yocto Project

- **CROss PlatformS (CROPS)* provides a consistent developer experience across Windows, Mac OS X and Linux distros through the use of containers**

- **Why Containers?**
  - Avoid host contamination
  - Easy route to multiple OS support, including Linux!
  - Repeatable builds
  - Fewer Linux distros to test
  - A path to tools in the cloud

  *The instructor also thinks of it as Containers Run Other People's Software

CROPS: Available Today

- **crops/extsdk-container**
  - Container that can support Extensible SDKs
  - Also supports standard SDKs

- **crops/toaster**
  - Latest released version of toaster/poky currently pyro

- **crops/toaster-master**
  - Keeps up with the current master of toaster/poky, kicked off via webhook so it's quite up to date.

- **crops/poky**
  - This is an Ubuntu (or other distro) container with the necessary packages to run poky installed, but not poky itself.
  - To run poky, you need a copy of it on your file system which you then map into the container.
  - This will work equally well for poky or an install of oe-core.

CROPS: Setup Docker

- **Install Docker**

  - For Linux, Docker is typically available via the distro package manager, otherwisw go to the Docker web site: https://docs.docker.com/engine/installation/linux/

  - For Windows and Mac, follow the CROPS Instructions here: https://github.com/crops/docker-win-mac-docs/wiki

- **Note: crops/samba container**

  - One of the nice features for windows/mac is the crops/samba container that exposes the docker volume to the host side via samba/cifs . This works around the fact that neither the windows nor mac filesystems have sufficient features to support a `bitbake` build. The docker volume is persistent just like a directory on a linux host would be.

CROPS: eSDK First Time

- **Follow the instructions at:**
  https://github.com/crops/extsdk-container

- **Linux:**
  ```
  $ docker run --rm -it -v /home/myuser/sdkstuff:/workdir
  crops/extsdk-container --url
  http://someserver/extensible_sdk_installer.sh
  ```

- **Windows:**
  ```
  $ docker run --rm -it -v myvolume:/workdir crops/extsdk-container
  --url http://someserver/extensible_sdk_installer.sh
  ```

- **Mac OS X:**
  ```
  $ docker run --rm -it -v myvolume:/workdir crops/extsdk-container
  --url http://someserver/extensible_sdk_installer.sh
  ```

CROPS: eSDK "--url" command

- **The "--url" tells the CROPS eSDK container where to find the eSDK**

- **That can be a website or you could copy into the container's workdir, and use:**
  `--url=`**`file:///workdir/extensible_sdk_installer.sh`**
  `or even `**`-url=`**`**/workdir/extensible_sdk_installer.sh`**

- **On Mac OS X or Windows, that would be provided via the Samba connection**

- **A useful CROPS eSDK command is "--help"**
  - This will print out all the startup options for the container.

CROPS: Example eSDK on Mac OS X

- **The first time, follow the CROPS Mac OS X install instructions**

- **Download the eSDK:**

```
$ wget http://downloads.yoctoproject.org/releases/yocto/milestones/\
yocto-2.4_M3/toolchain/x86_64/ \
poky-glibc-x86_64-core-image-sato-armv5e-toolchain-ext-2.3.sh
```

- **Create volume and run the samba container:**

```
$ docker volume create --name myvolume
$ docker run -it --rm -v myvolume:/workdir busybox \
  chown -R 1000:1000 /workdir

$ docker create -t -p 445:445 --name samba -v myvolume:/workdir crops/samba

$ docker start samba
```

CROPS: Example eSDK on Mac OS X

- **Mac OS X specific workaround (not on Windows):**
  - OS X will not let you connect to a locally running samba share. Therefore, create an alias for *127.0.0.1* of *127.0.0.2*.

```
$ sudo ifconfig lo0 127.0.0.2 alias up
```

- **Open the workdir with file browser:**
  - Open *Finder*, then hit '*Command-K*'. In the "*Server Address*" box type **smb://127.0.0.2/workdir** and click "Connect".

- **Copy the eSDK installer to the workdir:**

```
$ cp ~/Downloads/poky-glibc-x86_64-core-image-sato-armv5e-toolchain-ext-2.3.sh \
    /Volumes/workdir/
```

CROPS: Example eSDK on Mac OS X

- **Run the container:**

```
$ docker run --rm -it -v myvolume:/workdir crops/extsdk-container \
--url file:///workdir/poky-glibc-x86_64-core-image-sato-armv5e-toolchain-ext-2.3.sh


workdir$ . ./environment-setup-armv5e-poky-linux-gnueabi
workdir$ touch hello.c
```

- **In Finder view, right click on hello.c and edit in your favorite editor (e.g. Visual Studio Code)**

```
#include <stdio.h>

int main(void)
{
    printf("Hello, Prague 2017!\n");
    return 0;
}
```

CROPS: Example eSDK on Mac OS X

- **Compile and examine:**

```
/workdir$ $CC hello.c

/workdir$ file a.out

a.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux.so.3, for GNU/Linux 3.2.0,
BuildID[sha1]=17bdf1d27076e3e579c20007f60397c96984a012, not stripped

/workdir$ exit
```

CROPS: Example eSDK on Mac OS X, Second Time

- **When you close and then reopen Docker, you will need to restart samba**

```
$ # Restart the Samba container
$ docker start samba
$
$ # The eSDK is already extracted in the eSDK container, so no "--url"
$ docker run --rm -it -v myvolume:/workdir crops/extsdk-container
```

CROPS: Example Toaster on Mac

- **Run the container:**

```
$docker run --rm -it  -v myvol:/wd -p 127.0.0.1:12000:8000 crops/toaster \
--workdir=/wd

### Shell environment set up for builds. ###

…

Check if toaster can listen on 0.0.0.0:8000

OK

…

Running migrations:

  No migrations to apply.

Starting webserver...

Webserver address:  http://0.0.0.0:8000/

Successful start.

toasteruser@f07ebe8b10fe:/workdir/build$
```

CROPS: Example Poky on Mac

- **Run the container:**

```
docker run --rm -it  -v pokyvol:/wd crops/poky --workdir=/wd

pokyuser@5451bf7edfec:/wd$ ls

pokyuser@5451bf7edfec:/wd$ git clone git://git.yoctoproject.org/poky

Cloning into 'poky'...

remote: Counting objects: 354342, done.

remote: Compressing objects: 100% (85618/85618), done.

remote: Total 354342 (delta 263023), reused 353729 (delta 262410)

Receiving objects: 100% (354342/354342), 130.36 MiB | 11.28 MiB/s, done.

Resolving deltas: 100% (263023/263023), done.

Checking connectivity... done.

pokyuser@5451bf7edfec:/wd$ . ./poky/oe-init-build-env

pokyuser@5451bf7edfec:/wd/build$
```

CROPS: Toaster and Poky

- **The CROPS Poky can be found here:**

  https://hub.docker.com/r/crops/poky/

- **The CROPS Toaster release can be found here:**

  https://hub.docker.com/r/crops/toaster/

- **The CROPS extsdk-container can be found here:**

  https://hub.docker.com/r/crops/extsdk-container/

CROPS: Future

- **Target for 2.5 is an Eclipse environment where no Yocto Project specific plugin is needed. We are actively working with upstream Linux Tools/Docker Tools and CDT (C/C++ Development Tools).**

- **The Yocto Project magic will be in the metadata inside the toolchain container.**

- **This approach is also expected enable remote/Cloud Docker container instances.**

- **Some of the required upstream functionality expected to be in a December point release of CDT & Linux Tools.**

CROPS: Reference

- **You can use the Docker infrastructure (docker commit to an image, docker save to a tar.gz) to capture your container and pass it to others for exact analysis, for example for errors and regressions.**

CROPS: Call to Action

- **Users are typically able to get Docker and CROPs up and running on a Mac OS X or Windows host in less than 30 minutes, most of that is the Docker and CROPS container installation time.**

- **See if you can do that as fast on your host today or this week, and build and run "hello.c".**

Reference

- **The CROPs community is very active. Here is how you can update your cached containers:**

```
docker pull crops/extsdk-container

docker pull crops/poky

docker pull crops/toaster
```

- **Here is a quick "hello.c" for your eSDK container**

```c
#include <stdio.h>

int main(void)
{
    printf("Hello Berlin 2016!\n");
    return 0;
}
```

- **Lead Developers:**

  randy.e.witt@intel.com
  brian.avery@intel.com

Resources

- **Randy Witt's ELC Presentation (this is a must see):**
  - https://elinux.org/images/9/94/2017_ELC_-_Yocto_Project_Containers.pdf
  - https://www.youtube.com/watch?v=JXHLAWveh7Y

- **Yocto Project Dev Day Portland, 2017 Presentation (Windows):**
  - https://wiki.yoctoproject.org/wiki/images/f/f6/Yocto_DevDay_Advanced_Class_Portland.pdf

- **Github:**
  - https://github.com/crops

- **Docker Hub:**
  - https://hub.docker.com/r/crops

- **Freenode IRC:**
  - #crops

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 uprading 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 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 recommned 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

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

```c
#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 >**
>
> $ cd /scratch/working/build/tmp/deploy/sdk/
>
> $ ./poky-glibc-x86_64-core-image-base-armv5e-toolchain-ext-2.4.sh \
>   -d /scratch/sdk/qemuarm -y

- **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

Host

eSDK:~$

Target

root@qemuarm:~$
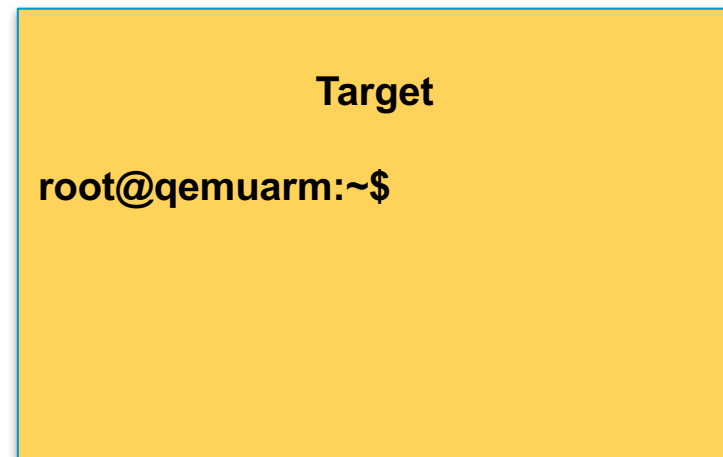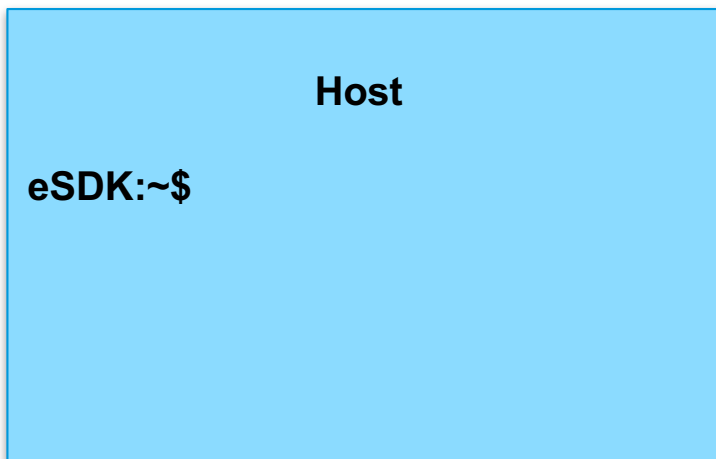
Kernel modules with eSDKs – Globalsetup

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

```
$ source  /scratch/sdk/qemuarm/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

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

```
$ 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 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 simplemodule
```

*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/hello.conf

< insert the following line and save >

hellokernel
```

- • **Then reboot the Qemu machine and verify**

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

Activity Nine

Analytics and the Event System

David Reyna

Analytics and the Event System - Overview

- **The Event System**

- **Example 1: Custom command line analytic tool**

- **Example 2: Custom Event Interface (knice)**

- **Example 3: Custom event types**

- **Example 4: Debugging coincident data in bitbake**

- **Example 5: Toaster**

Introduction

- **Thesis:**

  - The bitbake event system, together with the event database that comes with Toaster, can be used to generate and provide access to analytical data and provide a new unique toolset to solve difficult problems

- **What we will cover today:**

  - The problem space for extracting and analyzing data

  - Introduce the bitbake event system, interfaces

  - Event Examples: Toaster, CLI tools, customized bitbake UI

  - Resources

- **The full presentation can be found here:**

  - http://events.linuxfoundation.org/sites/events/files/slides/BitbakeAnalytics_ELC_Portland.pdf

- **What that presentation additionally covers:**

  - Deep dive on the event system code and components

  - Event database, database schema, custom events, custom tools, use cases, gotchas

The Problem Space (as I see it)
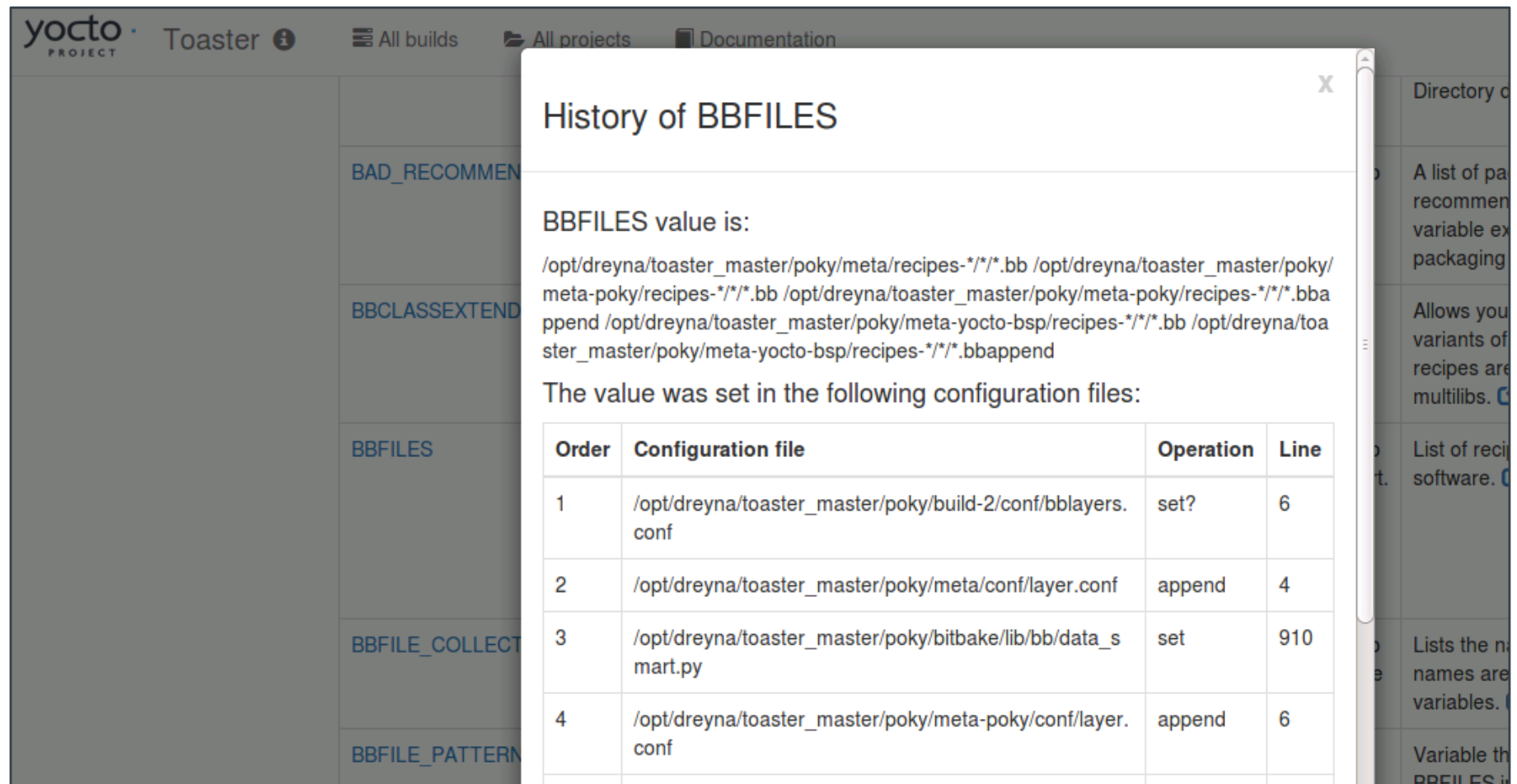
- **Types of addressable problems with analytics:**
    - Issues with time or coincident sensitivity
    - Issues with transient data values
    - Issues with transient UFOs (Unidentified Failing Objects)
    - Issues with trends (size, time, cache misses, scaling)
    - If the problem is a needle, where is the haystack to look in

- **We need:**
    - Easy access to deep data, time, and ordering
    - Reliable interaction with bitbake
    - Easy access to the data with tools, both provided and custom
    - Ability to acquire long term data, from a day to many months
    - Keep bitbake as pristine as possible

- **My builds are working, do I need this?**
    - Excellent, you are in good shape! However, if they stop working or when you do new work or try to scale, here are additional tools for your toolbox

The Problem Space (2)

- **Well known and documented data from bitbake builds:**
  - Logs (Build/Error logs)
  - Artifacts (Kernel, Images, SDKs)
  - Manifests (Image content, Licenses)
  - Variables (bitbake -e)
  - Dependencies

- **However…**
  - These only capture the final results of the build, not how the build progressed nor the intermediate or analytical information.
  - It is hard for example to correlate logs with other logs, let alone with other builds

- **The Answer!**
  - The bitbake event system
  - The bitbake event database
  - Events are <u>easy</u> to create, fire, listen to, and catch
  - There are more than 40 existing event types
  - Uses IPC over python xmlrpc sockets, with automatic data marshalling

Toaster Analytics – Intermediate Data Example

- **The Toaster database/GUI can for example display the intermediate values of bitbake variables, specifically each variable's modification history down to the file and line.**

Overview of Available Events

- **BuildInit|BuildCompleted|BuildStarted**
- **ConfigParsed|RecipeParsed**
- **ParseCompleted|ParseProgress|ParseStarted**
- **MultipleProviders|NoProvider**
- **runQueueTaskCompleted|runQueueTaskFailed|runQueueTaskSkipped| runQueueTaskStarted**
- **TaskBase|TaskFailed|TaskFailedSilent|TaskStarted| TaskSucceeded**
- **sceneQueueTaskCompleted|sceneQueueTaskFailed|sceneQueueTaskStarted**
- **CacheLoadCompleted|CacheLoadProgress|CacheLoadStarted**
- **TreeDataPreparationStarted|TreeDataPreparationCompleted**
- **DepTreeGenerated|SanityCheck|SanityCheckPassed**
- **MetadataEvent**
- **LogExecTTY|LogRecord**
- **CommandCompleted|CommandExit|CommandFailed**
- **CookerExit**

Event Clients (you are already an event user!)

- **Bitbake actually runs in a separate context, and expects an event client (called a "UI") to display bitbake's status and output**

- **Here are the existing bitbake event clients:**

  - **Knotty**: this is the default bitbake command line user interface that you know and love. It uses events to display the famous dynamic task list, and to show the various progress bars

  - **Toaster**: this is the bitbake GUI, which provides both a full event database and a full feature web interface. We will be using this as our primary example since it contains the most extensive implementation and support for events

  - **Depexp**: this executes a bitbake command to extract dependency data events, and then uses a GTK user interface to interact with it

  - **Ncurses**: this provides a simple ncurses-based terminal UI

Event Database

- The event database is built into Toaster to maintain persistent build data
- It can however just as easily be used directly with command line scripts or other SQL compatible tools
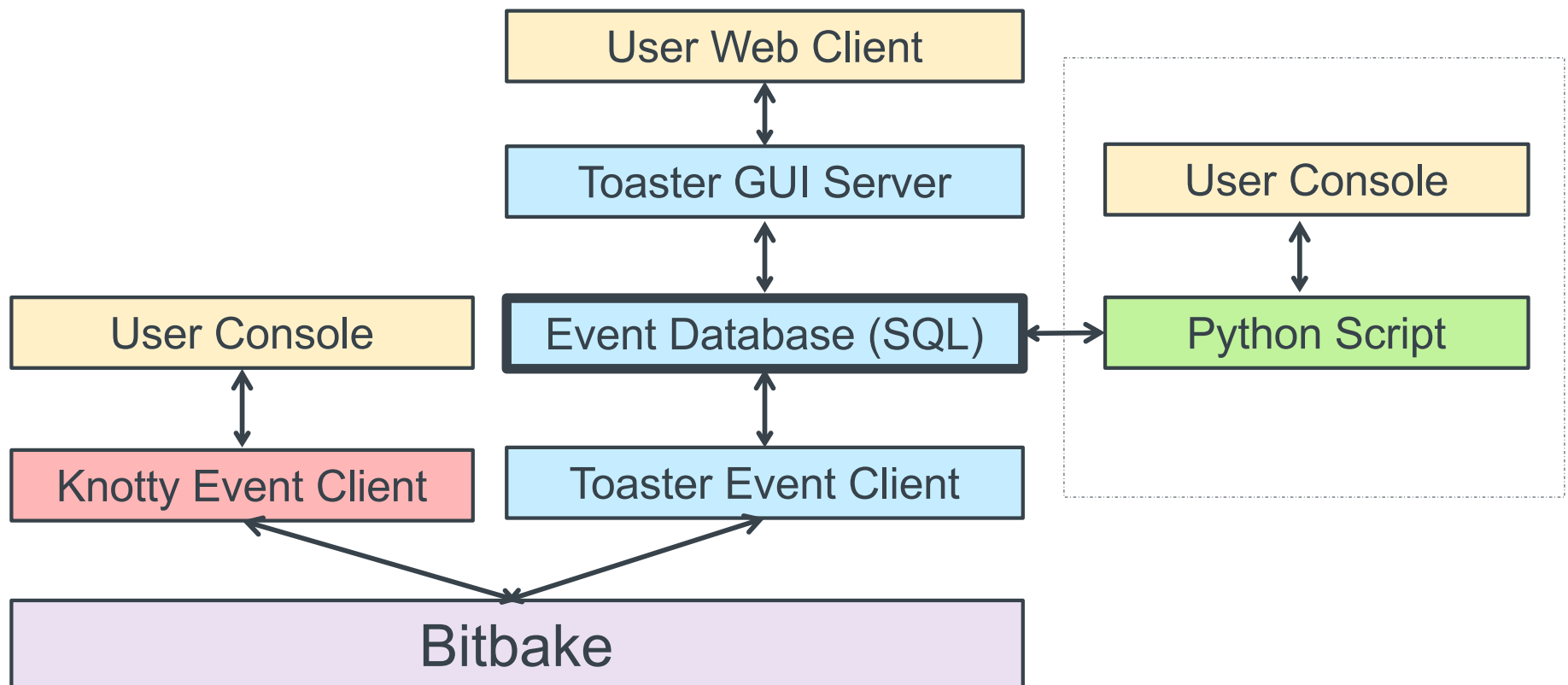
Example Event Database with CI Builders

- **If you enable the Toaster UI in a CI system, you can additionally get the event artifacts together with your build artifacts (you will definitely need to select a production level database)**
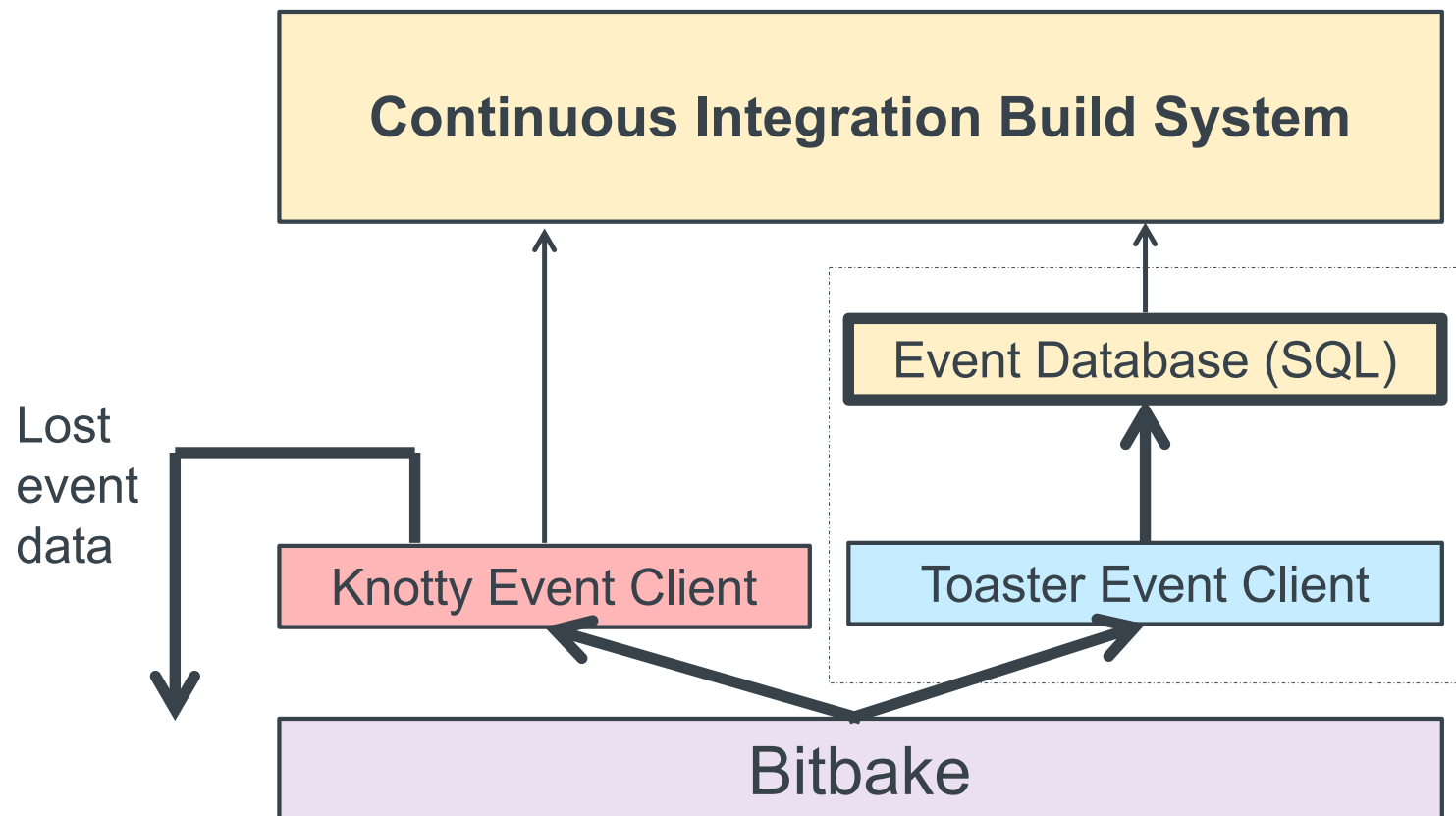
Adding Build Data to the Event Database

- **There are two easy ways to get build data into the event database**

  - Create and execute your builds from within the Toaster GUI

  ```
  $ cd /scratch
  $ source poky/oe-init-build-env
  $ source toaster start webport=0.0.0.0:8000   # local only: "localhost:8000"
  $ firefox localhost:8000   # here, connect browser from your using Toaster URL
  ```

  - Start Toaster, and run your command line builds in that environment

  ```
  $ . Poky/oe-init-build-env
  $ source toaster start webport=0.0.0.0:8000
  $ bitbake <whatever>
  ```

- **The 'source toaster' performs these tasks**

  - Creates the event database if not present, applies any schema updates

  - Starts the web client (this can be ignored for command line usage)

  - Sets the command line environment to use Toaster as the UI for bitbake ("BITBAKE_UI")

Analytics and the Event System - Overview

- The Event System

- **Example 1: Custom command line analytic tool**

- Example 2: Custom Event Interface (knice)

- Example 3: Custom event types

- Example 4: Debugging coincident data in bitbake

- Example 5: Toaster

Minimal Event Database Python Script

- Accessing the data in the event database is very simple. In this example we will print the data from the first-most Build record, and also look up and print the associated Target record

```
$ cat /scratch/src/events/sample_toaster_db_read.py
#!/usr/bin/env python3

import sqlite3
conn = sqlite3.connect('toaster.sqlite')
c = conn.cursor()

c.execute("SELECT * FROM orm_build")
build=c.fetchone()
print('Build=%s' % str(build))

c.execute("SELECT * FROM orm_target where build_id = '%s'" % build[0])
print('Target=%s' % str(c.fetchone()))
$
$ /scratch/src/events/sample_toaster_db_read.py
Build=(1, 'qemux86-64', 'poky', '2.2.1', '2017-02-12 23:55:52.137355', \
'2017-02-13 00:16:30.794711', 0, '/…/build_20170212_235552.805.log', \
'1.32.0', 1, 1478, 1478, '20170212235604')
Target=(1, 'core-image-base', '', 1, 0, '/…/license.manifest', 1, \
'/…/core-image-base-qemux86-64-20170212235604.rootfs.manifest')$
```

Full Feature Event Database Python Script

- **In this section we will work with an example python application that extracts and analyzes event data**

- **Specifically, we will attempt to investigate the question:**

  "How exactly do the tasks of a build overlap execution with other tasks, and on a higher level how to recipes overlap execution with other recipes, plus what data can extract around this question"

- **While this may not be a deep problem, and there are certainly OE tools that already provide similar information (e.g. pybootchart), the point is that (a) this was very easy and fast to write, and (b) you can now fully customize the analysis and output to your needs and desires.**

Task and Recipe Build Analysis Script

- Here is the list of available commands and features

```
$ more /scratch/src/events/event_overlap.py
# see db setup and schema info
$ /scratch/src/events/event_overlap.py --help
Commands: ?
 ?                               : show help
 b,build    [build_id]           : show or select builds
 d,data                          : show histogram data
 t,task     [task]               : show task database
 r,recipe   [recipe]             : show recipes database
 e,events   [task]               : show task time events
 E,Events   [recipe]             : show recipe time events
 o,overlap [task|0|n]            : show task|zero|n_max execution overlaps
 O,Overlap [recipe|0|n]          : show recipe|zero|n_max execution overlaps
 g,graph    [task]    [> file]   : graph task execution overlap
 G,Graph    [recipe] [> file]    : graph recipe execution overlap
 h,html     [task]    [> file]   : HTML graph task execution overlap [to file]
 H,Html     [recipe] [> file]    : HTML graph recipe execution overlap [to file]
 q,quit                          : quit

Examples:
  * Recipe/task filters accept wild cards, like 'native-*, '*-lib*'
  * Recipe/task filters get an automatic wild card at the end
  * Task names are in the form 'recipe:task', so 'acl*patch'
    will specifically match the 'acl*:do_patch' task
  * Use 'o 2' for the tasks in the two highest overlap count sets
  * Use 'O 0' for the recipes with zero overlaps
```

Histogram of Parallel Task/Recipe Execution ('d')

```
Commands: d
Histogram:For each task, max number of tasks executing in parallel
         0    1    2    3    4    5    6    7    8    9

       ----------------------------------------------
  0)    0  621   16   22   50   49   56   83   94   45
 10)   57   82   87   81   47   56   58   62   64   88
 20)  121  182  268  221  148

Histogram:For each recipe's task set, max number of recipes executing in parallel
         0    1    2    3    4    5    6    7    8    9

       ----------------------------------------------
  0)    0    5    1    1    1    1    1    1    3    3
 10)    1    2    2    2    2    1    1    3    1    6
 20)    1    1    2    1    1    2    2    1    1    1
 30)    1    1    2    2    1    3    1    2    2    1
 40)    1    1    1    1    1    1    1    1    3    1
 50)    1    2    4    2    2    1    1    1    1    2
 60)    1    2    1    1    1    2    1    1    1    2
 70)    1    1    2    2    2    2    1    3    3    1
 80)    3    2    1    1    1   10    7    8    8    8
 90)    7    7    2    2    3    2    2    1    1    2
100)    2    1    1    1    2    2    1    3    2    3
110)    2    1    2    1    1    1    1    2    1    1
120)    2    1    1    2    1    1    1    2    1    2
130)    1    1    1    1
```

Histogram of Overlapping Task/Recipe Execution

```
…
Histogram:For each task, max number of tasks that overlap its build
        0    1    2    3    4    5    6    7    8    9
    ------------------------------------------
  0) 614    9   10   29   28   42   46   55   51   47
10)  56   52   48   59   28   33   63   29   43   60
20)  60   94  119  223  105   95   53   57   36   40
30)  20   26   15   17   13    8   11    9    9    2
40)   7   10    9    7    3    6    6    3    6    6
50)   6    6    6    2    2    5    3    1    3    1
60)   4    2    5    1    2    2    1    2    3    5
... (sparse) ...
980)   0    0    1

Histogram:For each recipe's task set, max number of recipes that overlap its build
        0    1    2    3    4    5    6    7    8    9
    ------------------------------------------
  0)  67    0    0    0    0    0    0    0    0    0
 10)   0    0    0    0    0    0    0    0    0    0
... (all zeros) ...
 80)   0    0    0    0    5    1    1    8    4    1
 90)   3    2    0    1    4    4    3    0    0    0
100)   0    0    0    0    0    0    2    1    0    0
110)   2    0    1    2    0    0    3    0    1    2
120)   0    0    0    0    0    0    0    0    2    0
130)   0   26    8    5    2    6    5    0    0    1
... (sparse) ...
170)   0    4    0    0    0    0    0    0    0    0
180)   0    0    0    0    0    0   69
```

Initial Results

- **Here are some initial results when examining a "core-image-minimal" project with Task Count=2658 and Recipe Count=254**

- **We have as many as 148 tasks being able to run with all 24 available threads used**

- **There were 621 tasks that ran solo**

- **There were zero recipes that ran solo**

- **There was one task "linux-yocto:do_fetch" whose execution overlapped with 983 other tasks; the second most overlap was "python3-native:do_configure" with an overlap count of 798**

- **There were 69 recipes that overlaps with 186 other recipes, with the next highest overlap being 4 recipes that overlap with 171 other recipes**

- **The below sample HTML output page on task overlaps shows the amount of information available, with the recipe page too large to show in this context**

Initial Results

- **Let us see the available builds:**

```
Command: b
List of available builds:
   BuildId=1) CompletedOn=2017-02-13 00:16:30.794711, Outcome=SUCCEEDED,
      Project=Command line builds, Target=core-image-base, Task=''
   BuildId=2) CompletedOn=2017-02-13 00:46:40.724932, Outcome=FAILED,
      Project=Command line builds, Target=core-image-base, Task=populate_sdk_ext
   BuildId=3) CompletedOn=2017-02-13 00:46:26.513568, Outcome=SUCCEEDED,
      Project=Command line builds, Target=core-image-base, Task=''
   BuildId=4) CompletedOn=2017-02-23 09:02:31.109727, Outcome=SUCCEEDED,
      Project=Command line builds, Target=quilt-native, Task=''
```

- **Select the minimal build #4**

```
Command: b 4
Fetching build #4
Build: CompletedOn=2017-02-23 09:02:31.109727, Outcome=SUCCEEDED,
   Project='Command line builds' Target='quilt-native', Task='', Machine='qemux86-64'
Success: build #4, Task Count=9, Recipe Count=1
```

- **Run the commands** `d,t,r,o,O,g,G` **to get a sense of the minimal outputs**

Initial Results

- **Now select the large build (#1) and explore. We shall use the recipe filter 'zlib' to limit the output:**

```
Command: b 1
Command: o zlib
Command: e zlib
Command: t zlib
Command: r zlib
Command: o zlib
Command: O 0
```

- **Make sure your window is very wide, and then run this command to see a graph of the task overlaps for zlib:**

```
Command: g zlib
```

Sample HTML Output of Task Overlap

Analytics and the Event System - Overview

- **The Event System**

- **Example 1: Custom command line analytic tool**

- **Example 2: Custom Event Interface (knice)**

- **Example 3: Custom event types**

- **Example 4: Debugging coincident data in bitbake**

- **Example 5: Toaster**

Custom Event UI

- **If the knotty UI is too simple (since it does not collect data) and the Toaster UI too large for your analytic needs, you can make your own bitbake UI and have it handle specific events as you need. Here is a simple tutorial on how to do that.**

- **What we will do is start with the "knotty" UI, and then customize it as the "knice" UI.**

```
$ pushd ../bitbake/lib/bb/ui
$ cp knotty.py knice.py
$ sed -i -e "s/notty/nice/g" knice.py
$ vi knice.py
```

- **We make a simple change:**

```
-print("Nothing to do.  Use 'bitbake world' to build everything, \
    or run 'bitbake --help' for usage information.")
+print("NICE: Nothing to do.  Use 'bitbake world' to build everything, \
    or run 'bitbake --help' for usage information.")
```

- **Now we run it:**

```
$ popd
$ bitbake -u knice
NICE: Nothing to do.  Use 'bitbake world' to build everything, or run
'bitbake --help' for usage information.
```

Custom Event UI (2)

- Now let us instrument an event by updating "knice.py".

- First, let us add "bb.event.DepTreeGenerated" to the event list

```
$ vi ../bitbake/lib/bb/ui/knice.py

-      "bb.event.ProcessFinished"]
+      "bb.event.ProcessFinished","bb.event.DepTreeGenerated"]
```

- Now let us add a print statement to the otherwise empty "bb.event.DepTreeGenerated" handler code

```
      if isinstance(event, bb.event.DepTreeGenerated):
+          logger.info("NICE: bb.event.DepTreeGenerated received!")
          continue
```

- Now we run it and see our code run!

```
[build]$ bitbake -u knice quilt-native [ | grep NICE ]
...
NOTE: NICE: bb.event.DepTreeGenerated received!          | ETA:  0:00:00
...
```

Resources

- **Source code and example event database**
  - This is available as part of the Yocto Project Developer Day Advanced Class (see https://www.yoctoproject.org/yocto-project-dev-day-north-america-2017, and https://wiki.yoctoproject.org/wiki/DevDay_US_2017)

- **Here is the Toaster documentation, and Youtube video!**
  - http://www.yoctoproject.org/docs/latest/toaster-manual/toaster-manual.html#toaster-manual-start
  - https://youtu.be/BIXdOYLgPxA

- **Basic information about bitbake UI's**
  - http://elinux.org/Bitbake_Cheat_Sheet

- **Here is design information on the event model for Toaster**
  - https://wiki.yoctoproject.org/wiki/Event_information_model_for_Toaster

- **Here is the original design information on Toaster and bitbake communication**
  - https://wiki.yoctoproject.org/wiki/Toaster_and_bitbake_communications

Analytics and the Event System - Overview

- **The Event System**

- **Example 1: Custom command line analytic tool**

- **Example 2: Custom Event Interface (knice)**

- **Example 3: Custom event types**

- **Example 4: Debugging coincident data in bitbake**

- **Example 5: Toaster**

Custom events

- Normally, for a custom event you merely sub-class the event class or some other existing class, and add your new content

- In this example, we show how we can easily extend "MetadataEvent" and use it on the fly, since the sub-event 'type' is an arbitrary string and the data load is a simple dictionary.

- Event creation:

```
my_event_data = {
   "TOOLCHAIN_OUTPUTNAME": d.getVar("TOOLCHAIN_OUTPUTNAME")
}
bb.event.fire(bb.event.MetadataEvent("MyMetaEvent", my_event_data), d)
```

- Event handler:

```
if isinstance(event, bb.event.MetadataEvent):
    if event.type == "MyMetaEvent":
        my_toochain = event.data["TOOLCHAIN_OUTPUTNAME"]
```

Analytics and the Event System - Overview

- **The Event System**

- **Example 1: Custom command line analytic tool**

- **Example 2: Custom Event Interface (knice)**

- **Example 3: Custom event types**

- **Example 4: Debugging coincident data in bitbake**

- **Example 5: Toaster**

Using Events for debugging bitbake

- **You can also use the event system in debugging bitbake or your classes.**

- **Example 1: The quintessential example is to use "logger.info()" to insert print statements into the code. This is implemented as an event, meaning that will it be passed to the correct external UI and not lost in some random log file.**

- **Example 2: The ESDK file used to be copied to the build's "deploy/sdk" directory as part of the task "populate_sdk_ext". However, it is somehow happening later, and it is hard reading the code to determine when and where that is now occurring. We can use the event stream to help narrow down the candidates.**

  - First, we add a log call into the event read loop in "bitbake/lib/bb/ui/toasterui.py". This will provide a log of the received events as they go by, and also reveal when the ESDK file is created.

```
logger.info("FOO:"+str(event)+","+
    str(os.path.isfile('<path_to_esdk_file>')) )
```

- **I then run a build (in the Toaster context):**

Using Events for debugging bitbake (2)

- Second, we then run a build (in the Toaster context) and collect the events:

```
$ bitbake do_populate_sdk_ext > my_eventlog.txt
```

- Third, we examine the log to find when the file's state changed.

```
...
NOTE: FOO:<bb.event.DepTreeGenerated object at 0x7f94ec829710>,True
NOTE: FOO:<bb.event.MetadataEvent object at 0x7f94ec829358>,False
...
NOTE: FOO:<LogRecord: ... "Executing buildhistory_get_extra_sdkinfo ...">,False
...
NOTE: FOO:<LogRecord: BitBake.Main, ... sstate-build-populate_sdk_ext ...">,False
NOTE: FOO:<bb.build.TaskSucceeded object at 0x7f94e7f5f358>,True
…
```

- We see that the existing ESDK file was removed after "bb.event.DepTreeGenerated", and placed after "sstate-build-populate_sdk_ext". In other words it was moved out of the main "populate_sdk_ext" task and into its sstate task. QED.

Analytics and the Event System - Overview

- **The Event System**

- **Example 1: Custom command line analytic tool**

- **Example 2: Custom Event Interface (knice)**

- **Example 3: Custom event types**

- **Example 4: Debugging coincident data in bitbake**

- **Example 5: Toaster**

Adding Build Data to the Event Database

- **There are many existing analytic views in Toaster**

- **Start the Toaster GUI in the build directory (with open ports)**

  ```
  $ source toaster start webport=0.0.0.0:8000
  ```

- **On your host, open your browser to:**

  devdayXXX.yoctoproject.org:8000

- **Click on "All Builds", and select a build**

- **Click on "Time", "CPU Usage", and "Disk I/O"**

- **Click on "Tasks", and see the task order and cache usage**

Existing Toaster Analytics

- **The Toaster GUI already provides analytical data on builds, for example on sstate cache success rate, task execution time, CPU usage, and Disk I/O**

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 shared state objects

- **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-specifc 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 …

Questions and Answers

Open Topics

- Kernel:
  - Is the YP-2.4 kernel already obsolete?
  - Kernel fragments for any kernel without explicit inherit?
  - Enhanced kernel audit details?
  - Distro and kernel feature integration?

- Security
  - The Yocto Project has a general policy for sustaining (released) branches. We tend to fix individual security issues (CVE) instead of upgrade.
  - There is a Yocto Project security mailing list: yocto-security@yoctoproject.org
  - Low volume. We are working on automatically mailing patches that include the CVE tag to the mailing list so it is searchable, but we have not yet done so.

Open Topics

- **This is tracked by including the relevant CVE tag, pointing to the CVE information in the patches themselves, such as:**

```
+From 7340f67b9860ea0531c1450e5aa261c50f67165d Mon Sep 17 00:00:00 2001
+From: Paul Eggert <eggert@Penguin.CS.UCLA.EDU>
+Date: Sat, 29 Oct 2016 21:04:40 -0700
+Subject: [PATCH] When extracting, skip ".." members
+
+* NEWS: Document this.
+* src/extract.c (extract_archive): Skip members whose names contain
+"..".
+
+CVE: CVE-2016-6321
+Upstream-Status: Backport
+
+Cherry picked from commit: 7340f67 When extracting, skip ".." members
+
+Signed-off-by: Sona Sarmadi <sona.sarmadi@enea.com>
```

- **The above version includes a fix, documents it (CVE: CVE-2016-6321), and then also documents where the fix came from (upstream commit 7340f67 of that project).**

Thank you for your participation!

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 neccesary, 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

Bonus Activity

Node.js

Henry Bruce

Introduction

- **Credits: Brendan Le Foll and Paul Eggleton**

What we'll be doing

- **Understanding Node.js support in Open Embedded**

- **Using devtool to auto-generate Node.js recipes**

- **Building and deploying a package**

- **On-target Node.js application development**

- **Using devtool to package the application**

- **Discuss known issues and plans for future work**

Node.js and Open Embedded

- **Layer index recipe search returns ~10 hits**
  - We'll be working with the meta-oe recipe
  - (4.x, oldest LTS version)

- **More versions are available in meta-nodejs**

- **Devtool support was introduced in krogoth**
  - Use pyro

- **There's still work to do. See bug #10653.**

Using devtool to generate recipe

- ## Go to the build directory (with a clean shell)

  ```
  $ cd /scratch
  $ . poky/oe-init-build-env
  ```

- ## Create recipe from module in registry

  ```
  a) $ devtool add "npm://registry.npmjs.org;name=mraa;version=1.5.1"
     -- or --
  b) $ devtool add /scratch/src/nodejs/mraa-1.5.1.tgz
  ```

- ## Parses package.json for basis of recipe

  - Package name and version
  - Description, homepage
  - Location of source
  - Licenses

- ## Recursively goes through dependencies

  - Creates shrinkwrap and lockdown files

  ```
  $ devtool edit-recipe mraa
  ```

Under the hood

- **NPM makes it hard to limit network access to fetch task**

- **Fetch task walks dependency tree fetching tarballs from NPM registry**

- **Build task uses 'npm install' with registry disabled (OE specific patch) to create node_modules**

- **Install tasks puts node_modules in correct place**

Building and deploying

- **Build is really a pre-package task (apart from native gyp builds)**
  ```
  $ devtool build mraa
  ```

- **Deploy as normal**
  ```
  $ devtool deploy-target -s mraa
  root@target_addr
  ```

- **Is module installed on the target?**
  ```
  # npm -g ls mraa
  ```

Running on target

```
# export NODE_PATH=/usr/lib/node_modules
# node
> var mraa = require('mraa')
> console.log('mraa board: ' + mraa.getPlatformName())
> var gpio = new mraa.Gpio(360, true, true)
> gpio.dir(mraa.DIR_OUT)
> gpio.write(1)
```

Developing on target

- **Many ways of doing this. Let's keep it simple**

- **On your target**

```
# mkdir mmax-blinker
# cd mmax-blinker
```

- **Write some code and test it**
```
# cp $NODE_PATH/mraa/examples/javascript/Blink-IO.js .
# vi Blink-IO.js
```

change GPIO to "raw" id 360
add #!/usr/bin/node

```
# node Blink-IO.js
```

Create NPM module on target

- **Create package.json**
  ```
  # cp $NODE_PATH/mraa/COPYING .
  # npm init
  # vi package.json
  ```

  Add "bin" entry. Local dependency for mraa (or skip)

- **Install**
  ```
  # npm -g install
  ```

- **Test**
  ```
  # mmax-blinker
  ```

Create package for your application

- **Copy files to build host**

  ```
  scp -r root@x.x.x.x:mmax-blinker mmax-blinker
  ```

- **Check dependencies**

  - Local dependencies are for development only

- **Now create package**

  ```
  $ devtool add /path/to/mmax-blinker
  $ devtool edit-recipe mmax-blinker
  ```

Build, deploy and run application

- ## Build

  ```
  $ devtool build mmax-blinker
  ```

- ## Deploy

  ```
  $ devtool deploy-target mmax-blinker root@x.x.x.x
  # ln -s /usr/lib/node_modules/mmax-blinker/Blink-IO.js /usr/bin/mmax-blinker
  # chmod +x /usr/bin/mmax-blinker
  ```

- ## Run

  ```
  # mmax-blinker
  ```

Keep in Touch

- **https://wiki.yoctoproject.org/wiki/Nodejs_Workflow_Improvements**

FYI: How to add Nodejs to your project

```
$
$ cd /scratch/poky
$ git clone -b morty git://git.openembedded.org/meta-openembedded
$ git clone -b morty git://git.yoctoproject.org/meta-intel
$ source /scratch/poky/oe-init-build-env
$ echo "MACHINE = \"intel-corei7-64\"" >> conf/local.conf
$ echo "IMAGE_INSTALL_append = \" nodejs nodejs-npm curl \"" \
  >> conf/local.conf
$ echo "BBLAYERS += \"/scratch/poky/meta-intel \"" \
        >> conf/bblayers.conf
$ echo "BBLAYERS += \"/scratch/poky/meta-openembedded/meta-oe \"" \
        >> conf/bblayers.conf
$ bitbake core-image-base
$ bitbake nodejs-native
$ bitbake cmake-native
$ bitbake parted-native dosfstools-native mtools-native
$
```