Advanced Class

Randy Witt, Brian Avery, David Reyna, Mark Hatle,
Rudi Streif, Sean Hudson, Henry Bruce

**Yocto Project Developer Day •
Portland • 24 February 2017**

Agenda – The Advanced Class

9:00- 9:15	Opening session
9:15- 9:45	CROPS: Cross Platform support
9:45-10:15	Prelink
10:15-10:30	*Morning Break*
10:30-10:45	Account setup
10:45-11:10	WIC: the OE Image Creator
11:10-12:00	Userspace: packaging, installation, system services
12:00- 1:00	*Lunch (and board pass-out)*
1:00- 1:30	Board bring-up
1:30- 2:30	Devtool and ESDK
2:30- 2:45	*Afternoon Break*
2:45- 3:00	Class news
3:00- 3:30	Node.js
3:30- 4:00	Analytics and the Event System
4:00- 4:30	Kernel/Security Forum
4:30- 5:00	Q and A

Activity One

CROPS

Randy Witt, Brian Avery, 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

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 morty
- **crops/toaster-master**
  - Keeps up with the current master of toaster/poky, kicked off via webhook so it's quite up to date.
- **crop/poky**
  - This is an Ubuntu 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
```

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 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 Windows

- The first time, follow the CROPS Windows install instructions
- Copy in the ESDK (and hello.c) via Samba connection \\192.168.99.100\workdir
- Run the container:

```
$ docker run --rm -it -v ypvolume:/workdir crops/extsdk-container \
  --url file:///workdir/poky-glibc-x86_64-core-image-base-armv5e-\
toolchain-ext-2.1+snapshot.sh
workdir$ . ./environment-setup-armv5e-poky-linux-gnueabi
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]=11885f8816
9be4452e8f7ac3e40a4a137ec04d6f, not stripped
workdir$ exit
$
```

CROPS: Example eSDK on Windows, Second Time

- When you close and then reopen Docker, you need to re-setup the ESDK container's environment

```
$ # The environment information can be found here:
$ docker-machine env
$ eval $("C:\Program Files\Docker Toolbox\docker-machine.exe" env)
$
$ # Restart the Samba container
$ docker start samba
$
$ # The ESDK is already extracted in the ESDK container, so no "--url"
$ docker run --rm -it -v ypvolume:/workdir crops/extsdk-container
```

- Note: if you do not reassert the environment, you will get missing pipe and file errors

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 Toaster master can be found here:**

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

CROPS: Future

- **Target for 2.4 is an Eclipse plugin leveraging crops + devtool that should make app development easier and the same across host os's.**
- **Eclipse plug-in will evolve to provide UI for devtool commands**
- **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: Class activity

- **Users are typically able to get Docker and CROPs up and running on a 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

```
#include <stdio.h>

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

- Lead Developers:

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

Activity Two

Prelink

Mark Hatle

Introduction

- **What is prelinking?**
  - Need to know what is run-time linking is first…
- **Why prelink?**
- **Why NOT prelink?**
- **How to enable prelinking**
- **Related items**

Introduction
What is run-time linking?

- **At runtime, the dynamic linker (ld.so) must resolve shared object dependencies.**
- **Once the shared objects are loaded, the system determines if they were loaded at a predetermined address (objects by default are usually NOT loaded at a valid predetermined address).**
- **The dynamic linker must now bind/resolve all symbols (providers and users) to specific addresses. This operation generally changes relocation tables using a Copy-On-Write (COW) process.**

ELF Object (orig)

Sections:

```
Idx Name          Size      VMA               LMA               File off  Algn
4 .dynsym         000014b8  00000000004003f0  00000000004003f0  000003f0  2**3
5 .dynstr         00000d1f  00000000004018a8  00000000004018a8  000018a8  2**0
8 .rela.dyn       00000150  00000000004027d8  00000000004027d8  000027d8  2**3
9 .rela.plt       00001230  0000000000402928  0000000000402928  00002928  2**3
11 .plt           00000c30  0000000000403b70  0000000000403b70  00003b70  2**4
12 .text          00033691  00000000004047a0  00000000004047a0  000047a0  2**4
20 .dynamic       000001f0  00000000006f0e00  00000000006f0e00  000f0e00  2**3
21 .got           00000010  00000000006f0ff0  00000000006f0ff0  000f0ff0  2**3
22 .got.plt       00000628  00000000006f1000  00000000006f1000  000f1000  2**3
23 .data          00001220  00000000006f1640  00000000006f1640  000f1640  2**5
24 .bss           00001a68  00000000006f2860  00000000006f2860  000f2860  2**5
```

- Each of the above contains some type of dynamic data
- Dynamic linker (arch specific) may have to inspect and modify in-place (COW) information based on load address

What is prelinking?

- **Using ld.so, we bind the binaries to help determine a pre-calculated load address for all ELF binaries.**
- **This load address is calculated (libraries only) in such a way that conflicts should not occur, thus ensuring that objects are loaded at the predetermined address.**
- **The ELF object is modified to include the changes.**
- **Some conflict and symbol resolution still may be necessary, but only for items that can’t be pre-determined.**

ELF Shared Library (libc.so)

```
Version:                           0x1
Entry point address:               0x203b0
Start of program headers:          64 (bytes into file)
Start of section headers:          1676576 (bytes into file)
Flags:                             0x0

Relocation section '.rela.dyn' at offset 0x17f68 contains 1273 entries:
Offset          Info           Type           Sym. Value    Sym. Name + Addend
0000003937c8  000000000008 R_X86_64_RELATIVE                    398400
0000003937d8  000000000008 R_X86_64_RELATIVE                    1fbd0
0000003937e0  000000000008 R_X86_64_RELATIVE                    1483a0
0000003937e8  000000000008 R_X86_64_RELATIVE                    148400
```

```
Version:                           0x1
Entry point address:               0x3b4c4203b0
Start of program headers:          64 (bytes into file)
Start of section headers:          1681720 (bytes into file)
Flags:                             0x0

Relocation section '.rela.dyn' at offset 0x17f68 contains 1273 entries:
Offset          Info           Type           Sym. Value    Sym. Name + Addend
003b4c7937c8  000000000008 R_X86_64_RELATIVE                    3b4c798400
003b4c7937d8  000000000008 R_X86_64_RELATIVE                    3b4c41fbd0
003b4c7937e0  000000000008 R_X86_64_RELATIVE                    3b4c5483a0
003b4c7937e8  000000000008 R_X86_64_RELATIVE                    3b4c548400
```

What is prelinking?
New sections

- **.gnu.liblist**
  - Contains one ElfNN Lib structure for each shared library which the object has been pre- linked against, in the order in which they appear in symbol search scope.
- **.gnu.conflict**
  - Contains one ElfNN Rela structure for each needed prelink conflict fixup.
- **.gnu.libstr**
  - Contains strings for .gnu.liblist section where .gnu.liblist is not alowed.
- **.gnu_prelink_undo**
  - Contains private data to permit prelink --undo operation.

ELF Object (prelinked)

Sections:

```
Idx Name          Size      VMA               LMA               File off  Algn
4 .dynsym         000014b8  0000000000400428  0000000000400428  00000428  2**3
5 .gnu.liblist    00000064  00000000004018e0  00000000004018e0  000018e0  2**2
6 .gnu.conflict   00000888  0000000000401948  0000000000401948  00001948  2**3
9 .rela.dyn       00000150  00000000004027d8  00000000004027d8  000027d8  2**3
10 .rela.plt      00001230  0000000000402928  0000000000402928  00002928  2**3
12 .plt           00000c30  0000000000403b70  0000000000403b70  00003b70  2**4
13 .text          00033691  00000000004047a0  00000000004047a0  000047a0  2**4
21 .dynamic       000001f0  00000000006f0e00  00000000006f0e00  000f0e00  2**3
22 .got           00000010  00000000006f0ff0  00000000006f0ff0  000f0ff0  2**3
23 .got.plt       00000628  00000000006f1000  00000000006f1000  000f1000  2**3
24 .data          00001220  00000000006f1640  00000000006f1640  000f1640  2**5
25 .dynbss        00000808  00000000006f2860  00000000006f2860  000f2860  2**5
26 .bss           00001260  00000000006f3068  00000000006f3068  000f3068  2**5
27 .dynstr        00000d45  00000000008f3068  00000000008f3068  000f3068  2**0
29 .gnu.prelink_undo 000008f8  0000000000000000  0000000000000000  000f3dd0  2**3
```

- **Modified: .dynsym, .rela.dyn, ..rela.plt, …, .dynstr**
- **New: .gnu.liblist, .gnu.conflict, .dynbss, .gnu.prelink_undo**

Why prelink?

- **Faster Application Load Times**
  - Avoid unnecessary binding operations
  - More re-use of cached pages
  - Faster boot time
- **Less Memory Required**
  - Fewer relocations means few COW pages
  - More re-use of cached pages
- **Battery/Power Savings**
  - Fewer CPU cycles, more power savings
  - Less ram used, more power savings

Binary Load times – Busybox (/bin/sh) and GNU ld

Conf	relocs	cached	Rel relocs	Reloc time	Obj Ld Tm
arm - BB	-96	+28	-1218		
arm - ld	-374	-1653	-6323		
mips - BB	-119	+21	0		
mips - ld	-928	+20	0		
x86 - BB	-99	+30	-1264		
x86 - ld	-489	-3432	-17802		
x86-64 -BB	-93	+71	-1201	-13595365	-18293460
x86-64 - ld	-482	-3306	-17717	-130749928	-55978938

Using ‘LD_DEBUG=statistics’

’-’ indicates savings, ‘+’ indicates more

Load times – boot, login, ‘free’, halt

Arch	Lifetime	Free mem	Shared	Buffers	Cached
arm - min	-3.7s	-120k	0k	0k	-80k
arm - base	-8.0s	-188k	0k	-8k	+44k
mips - min	-5.0s	-180k	0k	-12k	+56k
mips - base	-9.0s	+620k	-11k	+8k	+180k
x86 - min	-2.9s	+84k	0k	0k	+8k
x86 - base	-5.9s	-60k	0k	-16k	-40k
x86-64-min	-4.9s	+196k	0k	+120k	+60k
x86-64-base	-7.6s	-172k	0k	+8k	-79k

Min – core-image-minimal, poky
Base – core-image-base, poky

’-’ indicates savings, ‘+’ indicates more

Why NOT prelink?

- **Load addresses are pre-determined. Could make it easier for an attacker to craft certain types of attacks.**
  - Incompatible with ASLR (Address Space Layout Randomization) (limited effectiveness on 32-bit)
  - Mitigation: use randomized prelink layout option (-R)
- **Not available on your architecture**
  - Arm64 – work in progress (slowly)
  - PPC64 not supported (elf v2)

How to enable/disable prelink

- **Local.conf, USER_CLASSES - ‘image-prelink’**

```
# Additional image features
#
# The following is a list of additional classes to use when building images which
# enable extra features. Some available options which can be included in this
variable
# are:
#   - 'buildstats' collect build statistics
#   - 'image-mklibs' to reduce shared library files size for an image
#   - 'image-prelink' in order to prelink the filesystem image
#   - 'image-swab' to perform host system intrusion detection
# NOTE: if listing mklibs & prelink both, then make sure mklibs is before prelink
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
```

Related items

- **Normally the prelinker uses ld.so. However you can't use ld.so in a cross environment.**
- **Cross prelinker adds 'prelink-rtld' that emulates the functionality of ld.so.**
- **Prelink-rtd will let you dump conflicts, load maps, as well as simple 'ldd' like functionality.**
- **ldd: prelink-rtld --root=<path> --target-paths /bin/bash**
  - Load addresses are emulated, unless prelinked
  - ld.so like environment vars, RTLD_DEBUG=… etc…

Activity Three

Class Account Setup

Notes for the Advanced Class:

- **The class will be given with YP-2.2 (Morty)**
- **Wifi Access:**
  - SSID: <TBD>
  - Password: <TBD>
- **Your account's IP access addresses**
  - SSH (password "devday"):

    `ssh ilab01@devday-a.yocto.io -p 10000(+your session number)`
  - HTTP:
    http://**`devday-a.yocto.io:30000(+your session number)`**

Yocto Project Dev Day Lab Setup

- **The virtual host's resources can be found here:**
  - Your Project: "/scratch/working/build-mbm“
  - Extensible-SDK Install: "/scratch/sdk/mbm“
  - Sources: "/scratch/src“
  - Poky: "/scratch/poky"
  - Downloads: "/scratch/downloads"
  - Sstate-cache: "/scratch/sstate-cache"
  - QEMU/Toaster Install: "/scratch/build“
- **You will be using SSH to communicate with your virtual server.**
- **You may want to change the default password (“devday”) after you log on, in case someone accidently uses the same account address as yours.**

FYI: Host Setup Gotchas

- **YP-2.2 has new host dependencies, for example:**

```
$ sudo apt-get update
$ sudo apt-get install git-core diffstat unzip texinfo \
    gcc-multilib build-essential chrpath socat libsdl1.2-dev \
    xterm sysstat python python3  xz-utils locales cpio
```

- **YP-2.2 Python 3 also has some dependencies.**
  - If you see the error "**`Please use a locale setting which supports utf-8`**", then you need to update/set your locale
  - One way to do that is to add this to your "~/.bash_profile"

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

  - And run this for good measure:

```
$ sudo locale-gen en_US.utf8
$ sudo dpkg-reconfigure locales      # choose en_US.utf8 as your default locale (~149)
```

FYI: How class project was prepared

```
$ cd /scratch
$ git clone -b morty git://git.yoctoproject.org/poky.git
$ cd poky
$ git clone -b morty git://git.openembedded.org/meta-openembedded
$ git clone -b morty git://git.yoctoproject.org/meta-intel
$ cd /scratch/working
$ source /scratch/poky/oe-init-build-env  build-mbm
$ echo "MACHINE = \"intel-corei7-64\"" >> 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++ \
      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
```

FYI: 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

FYI: 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

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 Four

WIC

Eduard Bartosh, David Reyna

WIC: Open Embedded Image Creation

- Physical devices accept and boot images in various ways depending on the specifics of the device. If your device require multiple partitions on an SD card, flash, or an HDD, you can use wic to create the properly partitioned image.
- The wic command generates partitioned images from existing OpenEmbedded build artifacts.
- Image generation is driven by partitioning commands contained in an provided or custom Openembedded kickstart file (.wks)
- The wic was designed to be completely extensible through a plug-in interface

WIC: Requirements

- You need to have the build artifacts already available, e.g. created an image like "core-image-minimal"
- You must build several native tools, which are tools built to run on the build system:

```
$ bitbake parted-native dosfstools-native mtools-native
```

- You must have sourced one of the build environment setup scripts (i.e. oe-init-build-env or oe-init-build-env-memres).
- Do this now:

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

WIC: Help

- Wic is documented on-line here:
  http://www.yoctoproject.org/docs/2.1/mega-manual/mega-manual.html#creating-partitioned-images

- You can get help from wic, where the available commands and help topics

```
$ wic -h
$ wic --help
```

- The wic help lists the available commands and help topics, from which you can request deeper help information

  $ wic help command

  $ wic help help_topic

WIC: Help

- You can find out more about the images wic creates using the existing kickstart files with the form “wic list <image> help” where “<image>” is for example "directdisk" or "mkefidisk“:

```
$ wic list mkefidisk help

Creates a partitioned EFI disk image that the user
can directly dd to boot media.
$
```

WIC: Operational Modes

- You can use wic in two different modes, depending on how much control you need for specifying the Openembedded build artifacts that are used for creating the image: Raw and Cooked:
  - Raw Mode: You explicitly specify build artifacts through command-line arguments.
  - Cooked Mode: The current MACHINE setting and image name are used to automatically locate and provide the build artifacts.
- You do not need root privileges to run wic. In fact, you should not run as root when using the utility.

WIC: Cooked Mode

- The general form of the wic command using Cooked Mode is:

  $ wic create *kickstart_file* -e *image_name*

  - *kickstart_file*: An OpenEmbedded kickstart file. You can provide your own custom file or supplied file.
  - *image_name*: Specifies the image built using the OpenEmbedded build system.

- Example:

  $ wic create mkefidisk -e core-image-minimal

WIC: Raw Mode

- The general form of the 'wic' command in raw mode is:
  $ wic create image_name.wks [options] [...]

- Where:
  image_name.wks (An OpenEmbedded kickstart file)
  -o OUTDIR, --outdir=OUTDIR
  -e IMAGE_NAME, --image-name=IMAGE_NAME
  -r ROOTFS_DIR, --rootfs-dir=ROOTFS_DIR
  -b BOOTIMG_DIR, --bootimg-dir=BOOTIMG_DIR
  -k KERNEL_DIR, --kernel-dir=KERNEL_DIR
  -n NATIVE_SYSROOT, --nativesysroot=NATIVE_SYSROOT
  -s, --skip-build-check
  -D, --debug

- Example:
  $ wic create directdisk -r *rootfs_dir* -b *bootimg_dir* \
  --k *kernel_dir* -n *native_sysroot*

WIC: Available kickstart files

- You can use wic to get a list of available kickstarts

```
$ wic list images
galileodisk-sd                  Galileo Gen 1/2 disk image (SD card)
systemd-bootdisk-uuid           EFI disk image with systemd-boot
galileodisk-usb                 Galileo Gen 1/2 disk image (USB storage)
systemd-bootdisk                EFI disk image with systemd-boot
directdisk                      'pcbios' direct disk image
directdisk-multi-rootfs         multi rootfs image using rootfs plugin
directdisk-bootloader-config 'pcbios' direct disk image with custom…
mkhybridiso                     hybrid ISO image
sdimage-bootpart                SD card image with a boot partition
mkgummidisk                     EFI disk image
directdisk-gpt                  'pcbios' direct disk image
qemux86-directdisk              qemu machine 'pcbios' direct disk image
mkefidisk                       EFI disk image
$
```

WIC: Example Kickstart File

- Here is the “mkefidisk.wks” kickstart file

```
$ cat ../../poky/scripts/lib/wic/canned-wks/mkefidisk.wks
# short-description: Create an EFI disk image
# long-description: Creates a partitioned EFI disk image that user
# can directly dd to boot media.

part /boot --source bootimg-efi --sourceparams="loader=grub-efi" \
   --ondisk sda --label msdos --active --align 1024

part / --source rootfs --ondisk sda --fstype=ext4 --label platform \
   --align 1024 --use-uuid

part swap --ondisk sda --size 44 --label swap1 --fstype=swap

bootloader --ptable gpt --timeout=5 --append="rootfstype=ext4 \
   console=ttyS0,115200 console=tty0“
$
```

WIC: Using WIC for MinnowBoard Max

```
$ wic create mkefidisk -e core-image-minimal
Checking basic build environment...
Done.
Creating image(s)...
Info: The new image(s) can be found here:
/var/tmp/wic/build/mkefidisk-201310230946-sda.direct
The following build artifacts were used to create the image(s):
ROOTFS_DIR: /home/trz/yocto/yocto-image/build/tmp/work/minnow-…
BOOTIMG_DIR: /home/trz/yocto/yocto-image/build/tmp/work/minnow-…
KERNEL_DIR: /home/trz/yocto/yocto-image/build/tmp/sysroots/minnow/…
NATIVE_SYSROOT: /home/trz/yocto/yocto-image/build/tmp/sysroots/…

The image(s) were created using OE kickstart file:
/home/trz/yocto/yocto-image/scripts/lib/image/canned-wks/mkefidisk.wks

$ sudo dd if=/var/tmp/wic/build/mkefidisk-201310230946-sda.direct \
  of=/dev/sdb bs=1M
$ sudo eject /dev/sdb
```

WIC: Custom Kickstart Files

- You can create your own custom kickstart and easily have wic find and use them. See the WIC documentation.
- Existing kickstart files can be found here, and used as templates:

  `poky/scripts/lib/wic/canned-wks`

- You can place your custom kickstart file with the canned ones, or add it to your layer
- Currently wic implementation only supports "partition" and "bootloader“ (more to come). See also:

  http://fedoraproject.org/wiki/Anaconda/Kickstart#part_or_partition
  http://fedoraproject.org/wiki/Anaconda/Kickstart#bootloader

WIC: Plug-ins

- Plug-ins allow wic functionality to be extended and specialized by users.
- Source plug-ins provide a mechanism to customize various aspects of the image generation process in wic, mainly the contents of partitions.
- The plug-ins provide a mechanism for mapping values specified in .wks files using the --source keyword to a particular plugin implementation that populates a corresponding partition.
- Plug-ins can be included as part of your custom layer.
- See the documentation for detailed information and examples.

WIC:

- Wic files can be automatically generated by adding WKS_FILE to your “local.conf”
- New kickstart commands are being added by demand, for example “include” and “config”
- We invite you to add kickstart files to your BSP layers for fast and easy OOB for your users
- Lead Developer:
  Bartosh, Eduard <eduard.bartosh@intel.com>

Introducing BMAPTOOL : Sparse Boot Images

- The new BMAPTOOL creates sparse images for fast image disk create (dd copies all of the blank space)
- The example usage of bmaptool is:

```
$ bitbake bmap-tools-native
$ native bmaptool –version
```

- Note: the “native” tool runs scipts in the project’s environment. It may get rename to “oe-native”. Discussion about its name on the mailing list: http://lists.openembedded.org/pipermail/openembedded-core/2016-October/127161.html
- Here is an info about bmap-tools Ubuntu package: http://packages.ubuntu.com/xenial/bmap-tools
- And about the tool itself: https://github.com/01org/bmap-tools/tree/master/docs

Activity Five

Userspace: Advanced Topics

Rudi Streif

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/working
  - source ../poky/oe-init-build-env build-userspace
- *Adjust Configuration (DONE FOR YOU)*
  - vi conf/local.conf

```
MACHINE = "qemux86-64"
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-minimal
- **Test**
  - runqemu qemux86-64 nographic

Activity Setup - Continued

- Create Layer
  - devtool create-workspace meta-uspapps
- Copy Source Files
  - cd ..
  - cp -r /scratch/src/userspace/uspsrc .

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/2.3/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/working/uspsrc/fibonacci-lib
  - Builds static and dynamic libraries to calculate the Fibonacci series and an application to test it.
- Create development environment
  - cd /scratch/working/build-userspace
  - devtool add fibonacci-lib /scratch/working/uspsrc/fibonacci-lib
- Build the recipe
  - bitbake fibonacci-lib
- Add to your image (conf/local.conf):

```
IMAGE_INSTALL_append = " fibonacci"
```

- Build and test image
  - bitbake core-image-minimal
  - runqemu qemux86-64 nographic

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 = " libfibonacci libfibonacci-examples"
```

- Build and test image
  - bitbake core-image-minimal
  - runqemu qemux86-64 nographic

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"
sbindir = "/usr/sbin"
libdir = "/usr/lib"
libexecdir = "/usr/lib"
sysconfdir = "/etc/"
datadir = "/usr/share"
mandir = "/use/share/man"
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/working/uspsrc/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/working/build-userspace
  - devtool add fibonacci-srv /scratch/working/uspsrc/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"
INITSCRIPT_PARAMS = "start 99 3 5 . stop 20 0 1 2 6 ."
SYSTEMD_SERVICE = "fibonacci-srv.service"
```

- Build the recipe
  - bitbake fibonacci-lib
- Add to your image (conf/local.conf):

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

- Build and test image
  - bitbake core-image-minimal
  - runqemu qemux86-64 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 Six

Board Bring-up

David Reyna

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…")

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*

Activity Seven

Devtool and Extensible SDKs

Sean Hudson

Devtool and eSDKs – WR Linux 9 Overview

- **The “devtool” is a collection of tools to help development, in particular user space development.**
- **It uses a local directory called “workspace” to hold its local and custom content**
- **It supports for example:**
  - Importing applications and creating wrapper recipe files and then packing the results Patching existing packages Devtool supports both projects and eSDKs. We will be focusing on eSDKs in this presentation.
- **The Extensible SDK (eSDK) is a portable and standalone development environment , basically an SDK with an added bitbake executive via devtool.**
- **eSDKs run on Linux hosts. They do not run on Windows (*unless you use the OE ‘CROPS’ container technology*).**
- ***NOTE: be careful not to mix the shell environments of the eSDK and the project, else unpredictable behavior will definitely occur.***

Devtool - A tool for the application developer

Before devtool, developer teams writing new applications had the following options:

1) Use build engineer's setup & write new recipe & use bitbake to rebuild image

Drawbacks:

- push changes to repo each iteration
- Long build times for image rebuilds

2) Use build engineer provided sdk/toolchain

Drawbacks:

- Difficult to update the sdk if app is library or depends on lib in development - may require multiple sdk updates per day
- Doesn't work for testing distro changes (like systemd-related work)

  Can't easily create/test the updated package as built by build engineer

3) Use **externalsrc** to work in own sandbox, building with recipes

Drawbacks:

- Difficult to get all the details right ***UNTIL devtool***

Devtool - Baking in a sandbox

Class will cover these use cases for devtool

- **Setup using an extensible SDK**
- **Development cycle with a new recipe**
  - Create a recipe from a source tree, then we will build, deploy, edit, build, and deploy
- **Development cycle with existing recipe**
  - Extract recipe and source, the edit, build, and deploy
  - Update the eSDK with changes

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

Devtool – Starting the Extensible SDK

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

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

- Source the devtool build environment

```
$ . environment-setup-corei7-64-poky-linux
```

- *Note that we have a kernel and a rootfs*

```
$ pushd tmp/deploy/images/intel-corei7-64
$ ls *qemux86-64.ext4 *qemux86-64.bin
core-image-base-intel-corei7-64.ext4
$ popd
```

- *Note: you can use devtool to re-build the rootfs (it was pre-built for you)*

```
$ devtool build-image core-image-base
```

FYI: Devtool QEMU Extensible SDK

- ***Here are instructions for QEMU if your board is not running***
- **Start a new Shell!** Otherwise, the existing bitbake environment can cause unexpected results

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

- Source the devtool build environment

```
$ . environment-setup-core2-64-poky-linux
```

- *Note that we have a kernel and a rootfs:*

```
$ pushd tmp/deploy/images/qemux86-64
bzImage-qemux86-64.bin
core-image-base-qemux86-64.ext4
$ popd
```

- *Note: you can use devtool to re-build the rootfs (it was pre-built for you)*

```
$ devtool build-image core-image-base
```

General Devtool Development Cycle

- **1. Add application to workspace:**
  devtool add [--version xxx] *myapp /path/to/source*
- **2. Build it:**
  devtool build *myapp*
- **3. Write to target device (w/network access):**
  devtool deploy-target *myapp root@ipaddr*
- **4. Edit source code & repeat steps 2-3 as necessary**

Devtool - hooking your application into the build

- **Run the devtool ‘add recipe-name /path/to/source’**

```
$ devtool add --version 1.0 bballs \
          /scratch/src/devtool_example/bballs
```

  - This generates a minimal recipe in the esdk/workspace layer
  - This adds EXTERNALSRC in an arm-sdk/workspace/appends bbappend file that points to the source
  - 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

Build files in the sdk directory

```
mbm> ls -FC
buildtools
cache
conf
devtool_example.tar
downloads
environment-setup-core2-64-wrs-
linux
environment-setup-x86-wrsmllib32-
linux
layers
preparing_build_system.log
README_templates
site-config-core2-64-wrs-linux
site-config-x86-wrsmllib32-linux
sstate-cache
sysroots
tmp
version-core2-64-wrs-linux
version-x86-wrsmllib32-linux
workspace
```

Workspace layer layout

```
mbm> tree workspace
.
├── appends
│   └── bballs_1.0.bbappend
├── conf
│   └── layer.conf
├── README
└── recipes
    └── bballs
        └── bballs_1.0.bb

4 directories, 4 files
```

Devtool - edit recipe

- **Edit the default workspace recipe**

```
$ vi workspace/recipes/bballs/bballs_1.0.bb

do_install () {
     # NOTE: unable to determine what to put here
     # - there is a Makefile but no target named
     # "install", so you will need to define this
     # yourself
-    :
+    install -d ${D}${bindir}
+    install -m 0755 bballs ${D}${bindir}
 }
```

- **Build the application**

```
$ devtool build bballs
```

Devtool - build/deploy/run app

- ***QEMU USERS: In a separate shell, source the eSDK environment and run qemu (this will be your target terminal)***

```
<open new clean shell>
$ . environment-setup-corei7-64-poky-linux
$ devtool runqemu nographic                          (FYI: "CTRL-A x" to close)
```

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

```
root@intel-corei7-64:~# ifconfig
```

- **<u>In the eSDK shell</u>, deploy the output (*the target's ip address may change)***

```
$ devtool deploy-target -s bballs root@<target_ip>
```

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

- **In the target shell, run application, and observe the display (2+2+2 bouncing balls)**

```
# /usr/bin/bballs
```

- **Stop the application with CTRL-C**

Devtool Run Application

- **Run application on target (CTLR-C to exit)**

```
root@intel-corei7-64:~# bballs
^^^^^^^^^^^^^^^^^
|                 |
|                 |
|           <     |
|<          b     |
| *               |
|     b           |
| *               |
|                 |
^^^^^^^^^^^^^^^^^
^C
root@intel-corei7-64:~#
```

Devtool - iterate

- **Iterate …**
  - edit source file to instantiate more balls

```
$ vi /tmp/devtool_example/bballs/b_main.cpp
-int num_hard = 2;
-int num_soft = 2;
-int num_spin = 2;
+int num_hard = 5;
+int num_soft = 5;
+int num_spin = 5;
```

  - …rebuild, redploy, retest, see more stuff bouncing!

```
$ devtool build bballs
$ devtool deploy-target bballs root@192.168.7.2
```

Devtool Run Updated Application

- **Re-run application on target (CTLR-C to exit)**

```
root@intel-corei7-64:~# bballs
^^^^^^^^^^^^^^^^^
| b       b      |
| *              |
|*   < b * *<    |
|              < |
|       <        |
|   *         b  |
|                |
|    b           <
^^^^^^^^^^^^^^^^^
^C
root@intel-corei7-64:~#
```

devtool - sandbox-to-repo

- **You can use ‘modify’ and ‘update-recipe’ to work with source in your sandbox, and update the sdk/git-repo recipe as a patch/srcrev**
  - devtool modify -x ... : extract source from for a recipe in a layer, into your sandbox
  - iterate: modify source in sandbox, build , deploy, test
  - devtool update-recipe... create a patch to the eSDK or commit to the source git repo
  - *NOTE: the source for the existing package ‘which’ (and all the other packages) was included in eSDK, and that is why we can directly edit and compile it. In this case we are going to sandbox those changes in a separate location.*

devtool modify: Extract source and modify

- ***(NOTE: If you do not have git configured for your host, preset some values now)***

```
$ git config --global user.email JoeSmith@nowhere.com
$ git config --global user.name  "Joe Smith"
```

- **Run the devtool command to extract src and setup sandbox**

```
$ devtool modify -x which /scratch/src/devtool_example/which
```

- **Modify the package source in your sandbox (line 459)**

```
$ vi /scratch/src/devtool_example/which/which.c
        - print_usage(stdout);
        + printf("hello class!\n");
```

- **On host, rebuild the package and deploy it**

```
$ devtool build which
$ devtool deploy-target which root@192.168.7.2
```

- **On target, demonstrate the change**

```
root@intel-corei7-64:~# which --help
Hello class
root@intel-corei7-64:~#
```

devtool update-recipe : Push changes back to eSDK's repository layer

- **In your sandbox, commit the change for the tip of the rev**

```
$ pushd /scratch/srcdevtool_example/which
$ git add which.c
$ git commit -m "changes for class"
$ popd
```

- **Update-recipe to modify the recipe in the eSDK, keeping existing patches**

```
$ devtool update-recipe -n which
```

- **Verify that the eSDK has been updated with the patch**

```
$ grep changes-for-class $(find . -name "which*bb")
./layers/oe-core/meta/recipes-extended/which/which_2.21.bb:
file://0001-changes-for-class.patch \
$ find . -name 0001-changes-for-class.patch
./layers/oe-core/meta/recipes-extended/which/which-2.21/0001-
changes-for-class.patch
```

- ***NOTE: If you reset the recipe, extract 'which' again, you will see the change (but extract to new location)***

devtool - A few more commands

- **We've shown usage for sub-commands add, modify, build, deploy-target (implicitly undeploy-target), runqemu**
- **devtool status: shows status of workspace**

```
$ devtool status
bballs: /scratch/src/devtool_example/bballs
(/scratch/sdk/mbm/workspace/recipes/bballs/bballs_1.0.bb)
which: /scratch/src/devtool_example/which
```

- **devtool package <recipe>: creates installable packages for a recipe**

```
$ devtool package bballs
...
NOTE: Your packages are in /…/tmp/deploy/rpm
$ pushd tmp/deploy/rpm
$ ls */*bball*
core2_64/bballs-1.0-r0.core2_64.rpm
core2_64/bballs-dbg-1.0-r0.core2_64.rpm
core2_64/bballs-dev-1.0-r0.core2_64.rpm
$ popd
```

Devtool - A few more commands

- devtool reset <recipe>: removes a recipe from the workspace, but not the source tree
- devtool extract <recipe> <dest-src-tree>: extracts a recipe's source files to a source tree, but not into workspace, and not ready for building
- devtool search: broad regex search into package data, including recipe DESCIPTION so less useful for packages with names like 'which'

Yocto devtool - References

1. **Yocto devtool documentation**

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

2. **Tool Author Paul Eggleton's ELC Presentation:**

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

3. **Trevor Woerner's Tutorial**

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

Activity Eight

Node.js

Henry Bruce

Introduction

- What can you expect to learn from this class
- What you won’t get from this class
- 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 morty
- **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/working/build-mbm
```

- **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

Activity Nine

Analytics and the Event System

David Reyna

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)

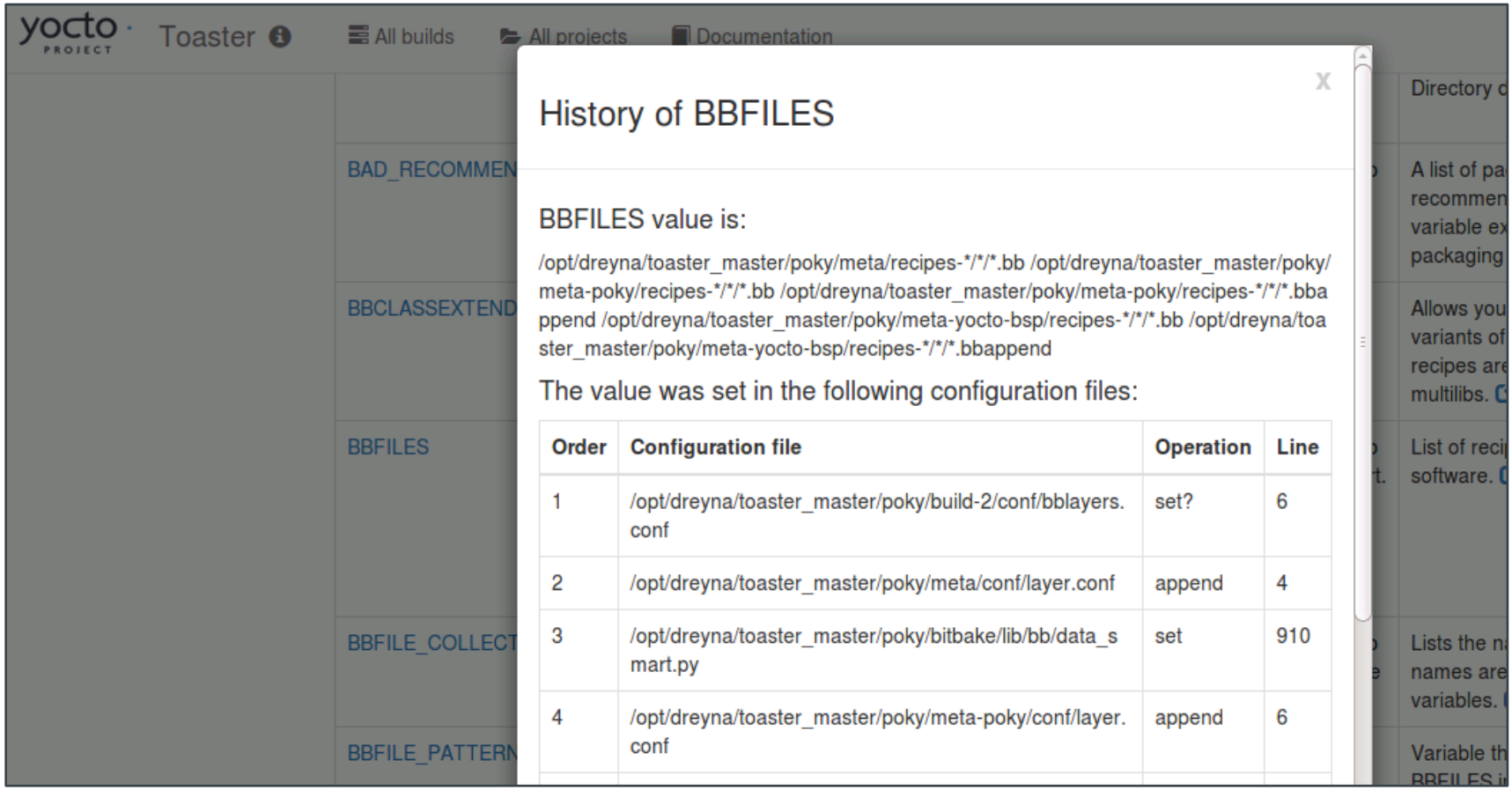
- **Types of addressable problems with analytics:**
  - Issues with time or coincident sensitivity
  - Issues with transient data values
  - Issues with transient UFOs (Unidentified Failing Objects)
  - Issues with trends (size, time, cache misses, scaling)
  - If the problem is a needle, where is the 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 easy 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

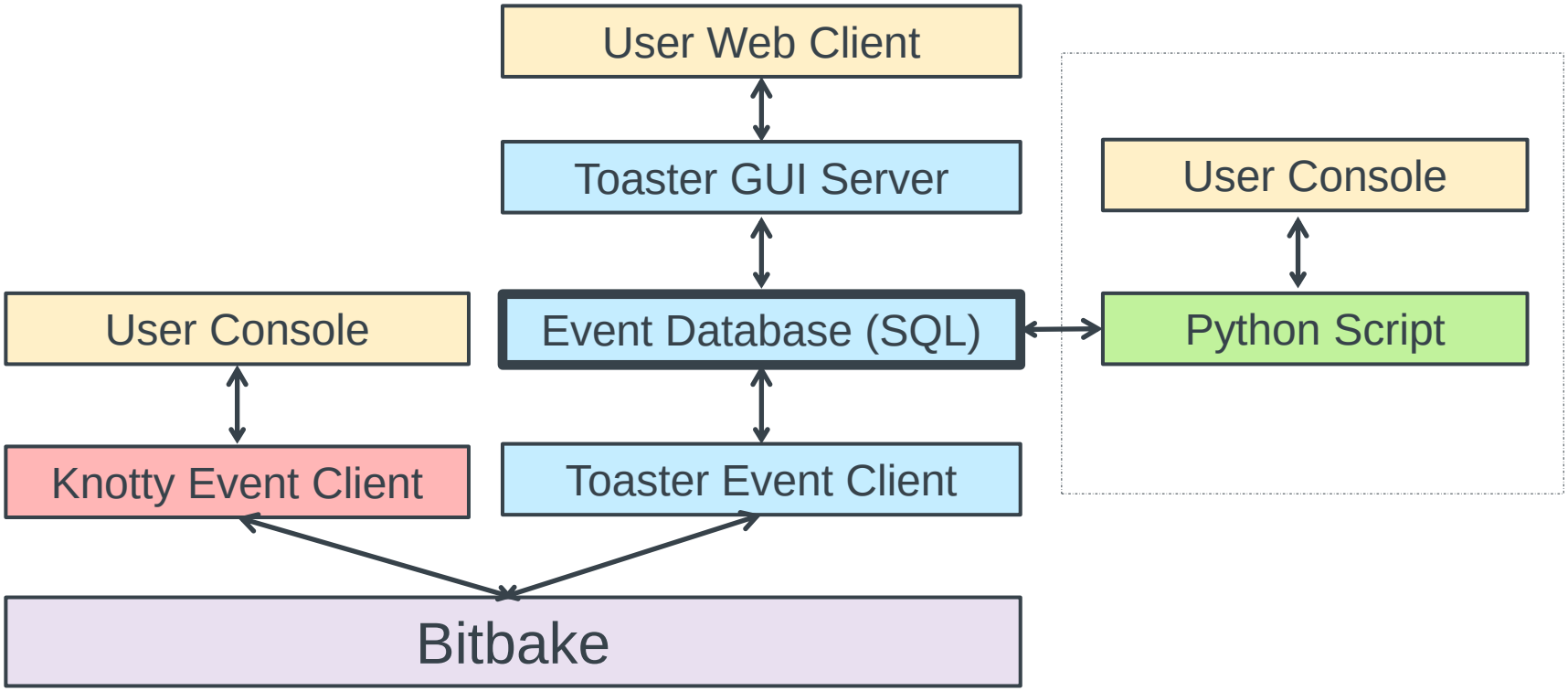
- BuildInit|BuildCompleted|BuildStarted
- ConfigParsed|RecipeParsed
- ParseCompleted|ParseProgress|ParseStarted
- MultipleProviders|NoProvider
- runQueueTaskCompleted|runQueueTaskFailed|runQueueTaskSkipped|
  runQueueTaskStarted
- TaskBase|TaskFailed|TaskFailedSilent|TaskStarted|
  TaskSucceeded
- sceneQueueTaskCompleted|sceneQueueTaskFailed|sceneQueueTaskStarted
- 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

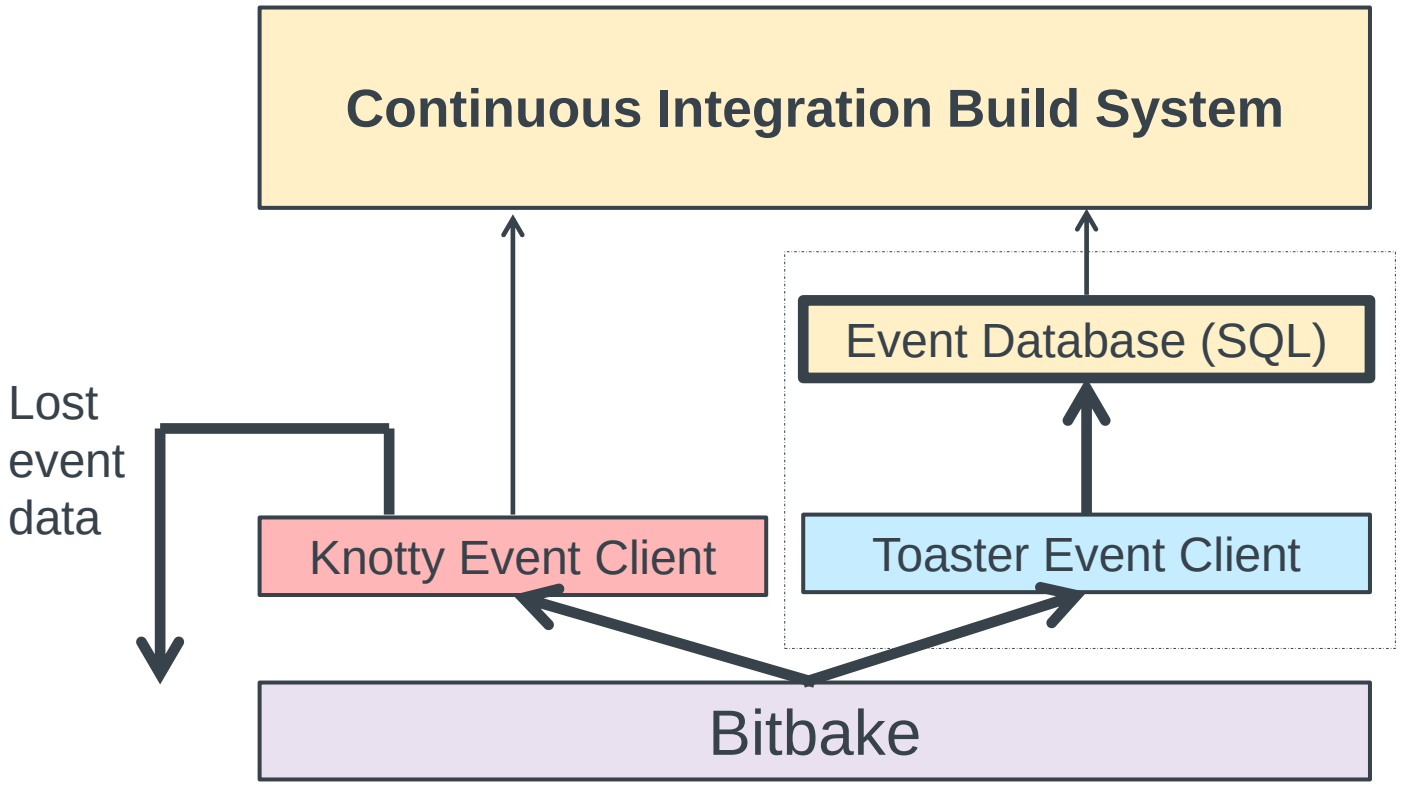
Event Database

- 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

```
$ . oe-init-build-env
$ source toaster start webport=127.0.0.1:8800
$ firefox localhost:8800
```

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

```
$ . oe-init-build-env
$ source toaster start webport=0.0.0.0:8800
$ 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”)

Example 1: Custom command line analytic tool

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

```
$ cd /scratch
$ . oe-init-build-env
$ cat 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()))
$
$ ./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

```
$ ./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

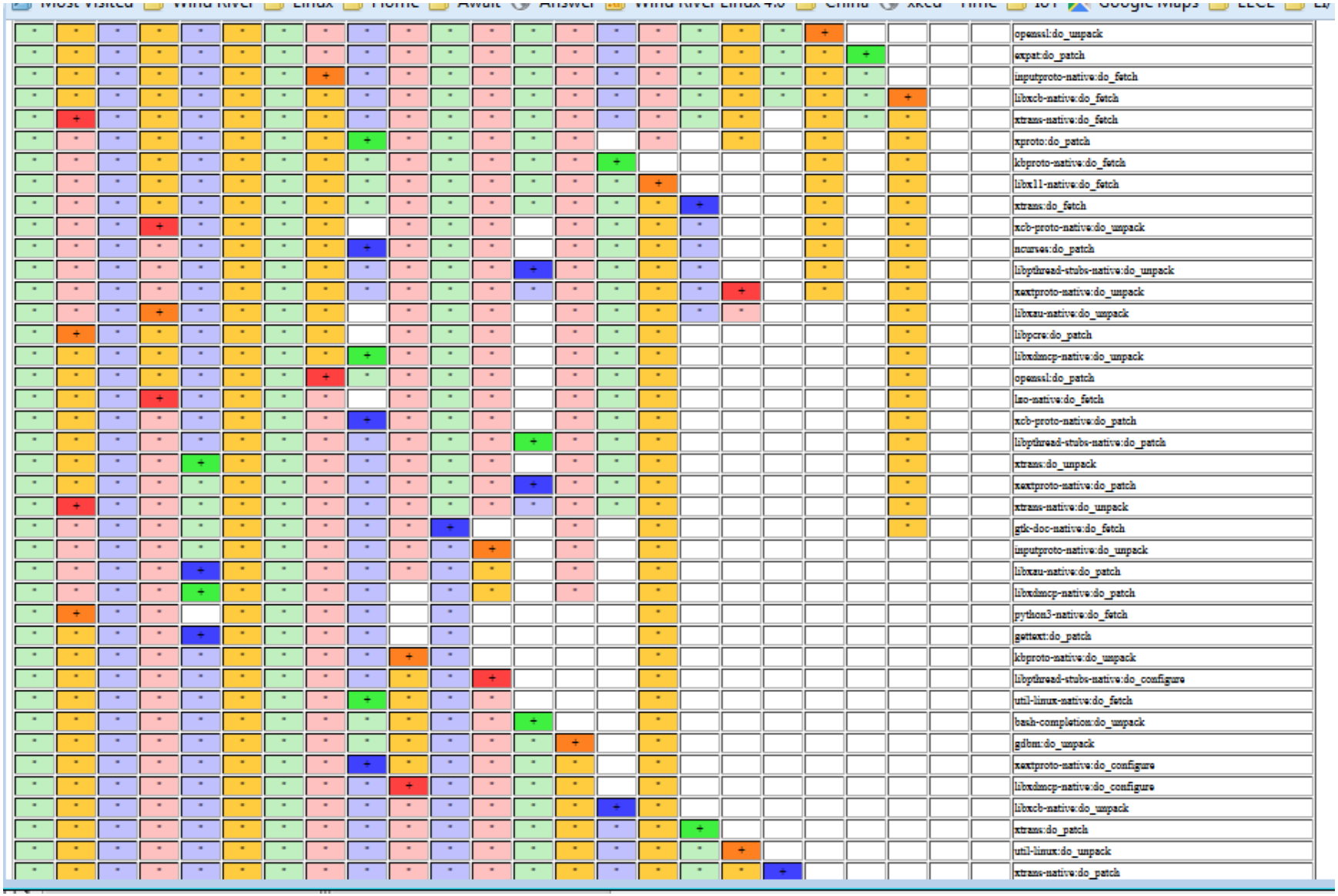
- Now select the large build (#1) and explore. We shall use the recipe filter ‘zlib’ to limit the output:

```
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



Example 2: Custom Event Interface (knice)

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/BlXdOYLgPxA

- **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

Bonus Example 3: Custom event types

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"]
```

Bonus Example 4: Debugging coincident data in bitbake

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.

Bonus 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
```

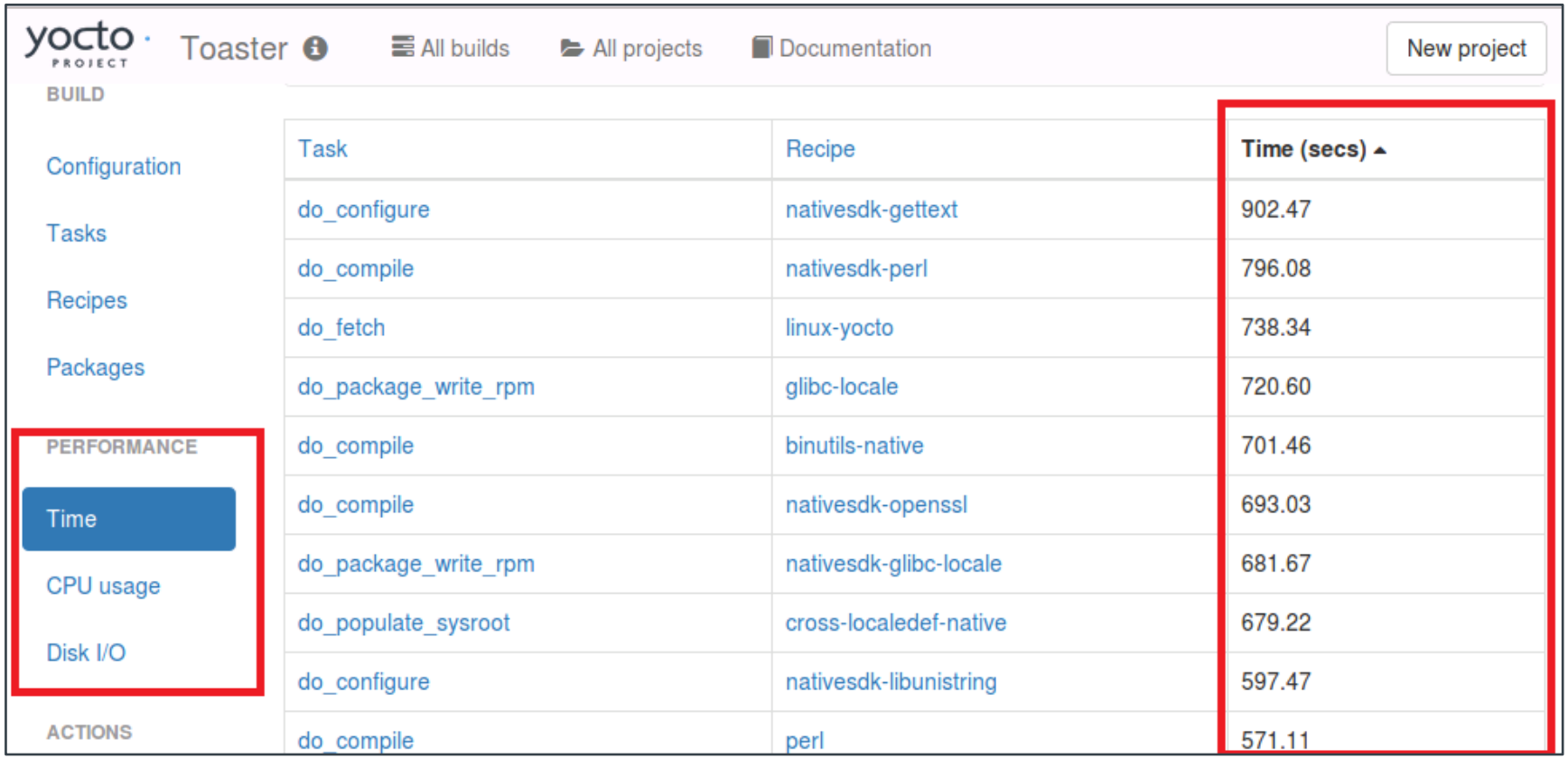
- **On your host, open your browser to:**

  http://devday-a.yocto.io:30000(+your session number)

- **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 Nine

Kernel And Security Open Forum Staff

Questions and Answers

Thank you for your participation!