



Extracting analytics from complex OpenEmbedded builds

David Reyna
Wind River Systems

**Embedded Linux Conference •
Portland • 23 February 2017**

Introduction

- **Introduction:**
 - I am David Reyna, senior developer at Wind River Systems, maintainer for Toaster, long time contributor to OpenEmbedded and Yocto Project.
- **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
- **Goal: Introduce, Enable, Call to Opportunity**

Introduction

- **What we will cover:**
 - The problem space for extracting and analyzing data
 - Introduce the bitbake event system and interfaces
 - Introduce the bitbake event database
 - Deep dive on the event system
 - Examples, existing tools, custom tools, and use cases
 - Gotchas
 - Resources
- **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

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

The Problem Space (2)

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



The Event System Features

Event System Features

- **Built into Bitbake**
 - The event system is built deeply into bitbake, and has years of used and testing
- **Rich set of data events**
 - There are more than 40 existing event types, covering builds to tasks to recipes, with status and details describing the results, to progress events to support the visual progress bars
- **IPC over python xmlrpc sockets, with automatic data marshalling**
 - This allows bitbake and its interfaces to run in separate contexts
- **Very flexible**
 - Events are not locked into a rigid design structure
- **Easy to attach custom event handlers**
- **Easy to add custom event types**

Overview of Available Events

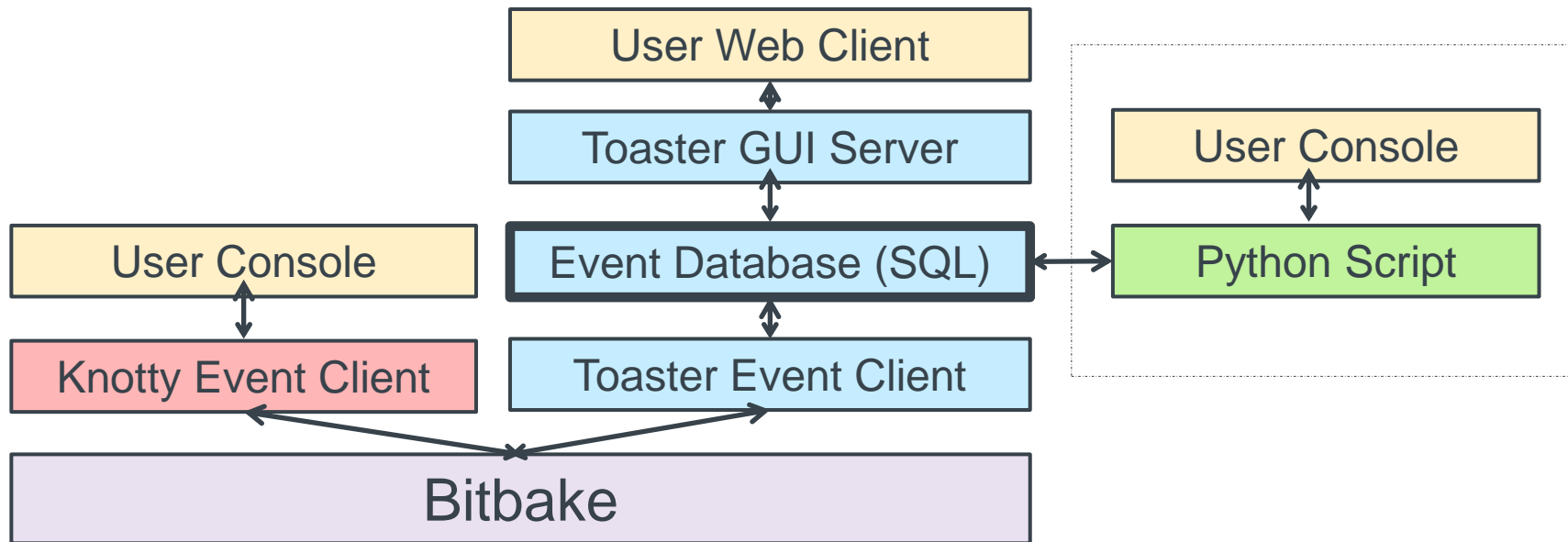
- **BuildInit|BuildCompleted|BuildStarted**
- **ConfigParsed|RecipeParsed**
- **ParseCompleted|ParseProgress|ParseStarted**
- **MultipleProviders|NoProvider**
- **runQueueTaskCompleted|runQueueTaskFailed|runQueueTaskSkipped|
runQueueTaskStarted**
- **TaskBase|TaskFailed|TaskFailedSilent|TaskStarted|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

Event Clients

- 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

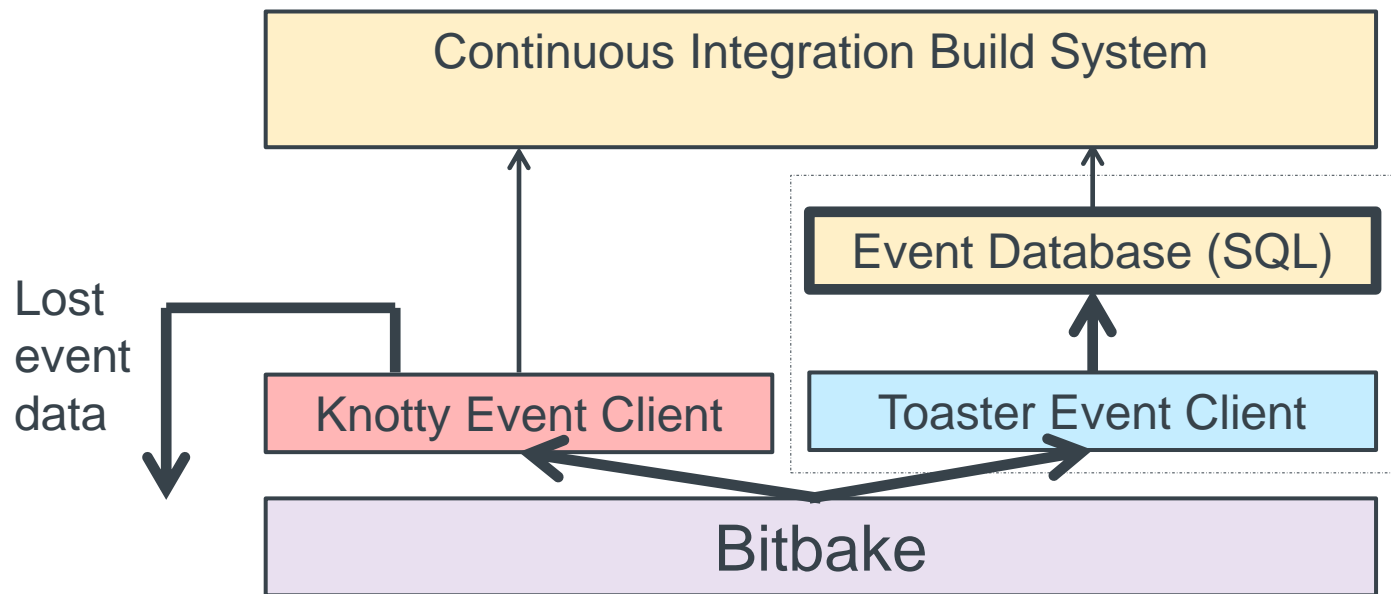
Event Database

- 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)



The Event Database Instance

- **By default, the event database is instantiated with sqlite**
 - Easy to manage
 - Instant access out of the box
 - Small footprint
 - Easy for quick runs, small user base
- **However, for managing multiple builds sqlite soon fails because it cannot fully manage simultaneous access**
- **For production systems, it is recommend to setup a robust system like MySQL or mariaDB and attach it to Toaster in place of sqlite**
 - Harder to set up the first time, but then just works
 - Much more robust
 - Can handle many simultaneous users and builds
 - Instructions can be found here:
https://wiki.yoctoproject.org/wiki/Setting_up_a_production_instance_of_Toaster

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

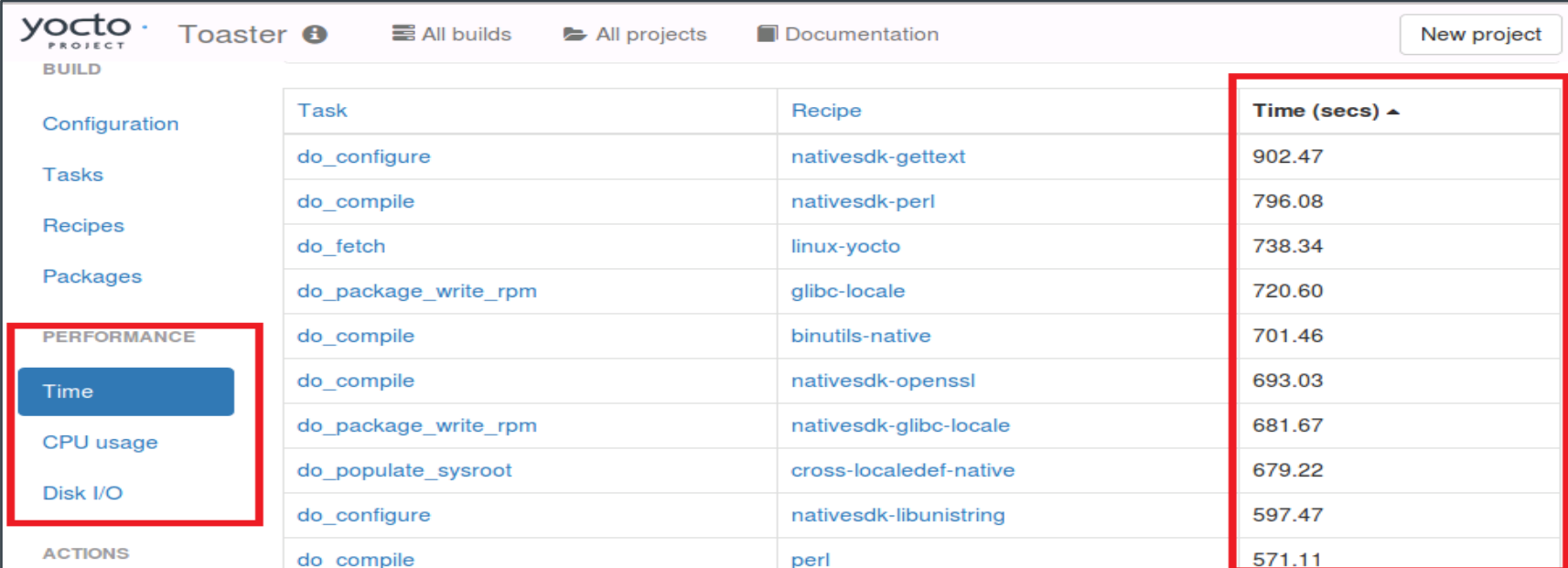
```
$ . oe-init-build-env  
$ source toaster start webport=0.0.0.0:8800  
$ bitbake <whatever>
```

- **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”)

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

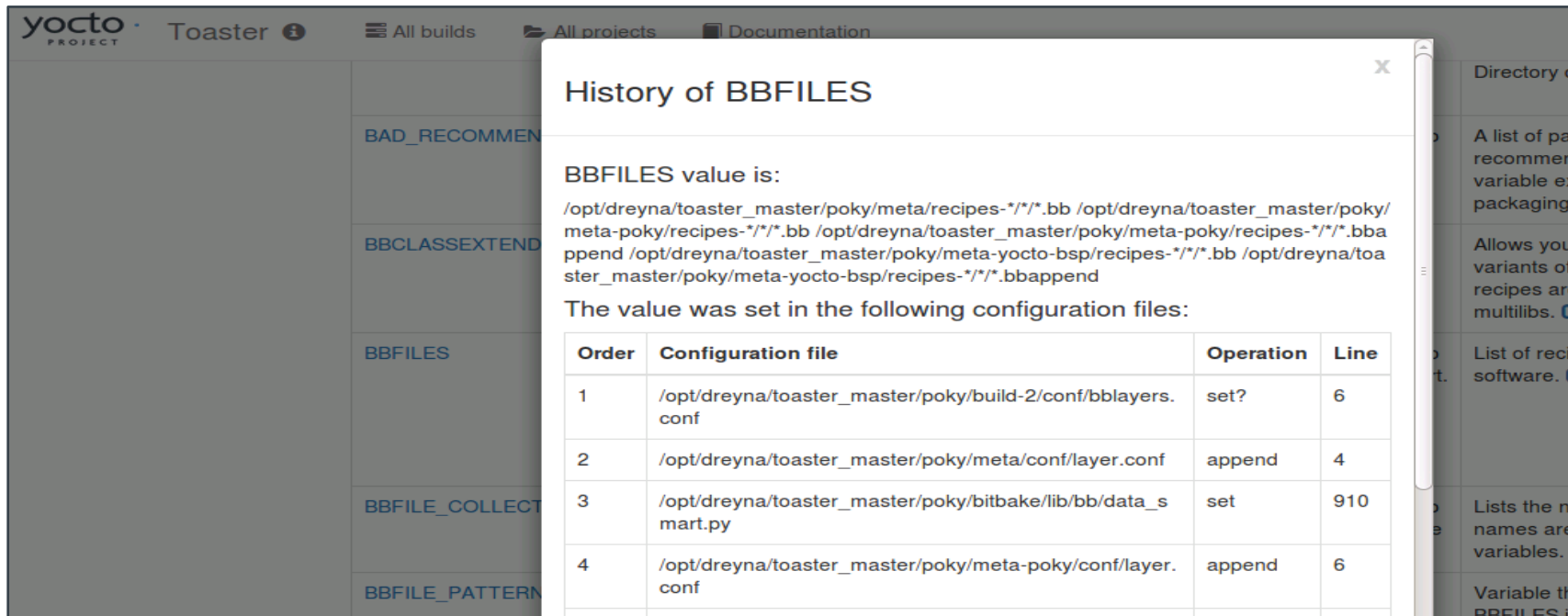


The screenshot shows the Yocto Project Toaster GUI. The left sidebar has a 'PERFORMANCE' section with three tabs: 'Time' (selected), 'CPU usage', and 'Disk I/O'. The main area displays a table of build tasks and their execution times. The table has three columns: 'Task', 'Recipe', and 'Time (secs)'. The 'Time' column is highlighted with a red box. The table contains 10 rows of data.

| Task | Recipe | Time (secs) ▲ |
|----------------------|------------------------|---------------|
| do_configure | nativesdk-gettext | 902.47 |
| do_compile | nativesdk-perl | 796.08 |
| do_fetch | linux-yocto | 738.34 |
| do_package_write_rpm | glibc-locale | 720.60 |
| do_compile | binutils-native | 701.46 |
| do_compile | nativesdk-openssl | 693.03 |
| do_package_write_rpm | nativesdk-glibc-locale | 681.67 |
| do_populate_sysroot | cross-localedef-native | 679.22 |
| do_configure | nativesdk-libunistring | 597.47 |
| do_compile | perl | 571.11 |

Existing Toaster Analytics

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



The screenshot shows the Yocto Project Toaster GUI with a modal dialog titled "History of BBFILES". The dialog displays the current value of the BBFILES variable and a table of its modification history.

BBFILES value is:

```
/opt/dreyna/toaster_master/poky/meta/recipes-*//*/*.bb /opt/dreyna/toaster_master/poky/meta-poky/recipes-*//*/*.bb /opt/dreyna/toaster_master/poky/meta-poky/recipes-*//*/*.bbappend /opt/dreyna/toaster_master/poky/meta-yocto-bsp/recipes-*//*/*.bb /opt/dreyna/toaster_master/poky/meta-yocto-bsp/recipes-*//*/*.bbappend
```

The value was set in the following configuration files:

| Order | Configuration file | Operation | Line |
|-------|--|-----------|------|
| 1 | /opt/dreyna/toaster_master/poky/build-2/conf/bblayers.conf | set? | 6 |
| 2 | /opt/dreyna/toaster_master/poky/meta/conf/layer.conf | append | 4 |
| 3 | /opt/dreyna/toaster_master/poky/bitbake/lib/bb/data_smart.py | set | 910 |
| 4 | /opt/dreyna/toaster_master/poky/meta-poky/conf/layer.conf | append | 6 |



The Event System Deep Dive

The components of the event system

- **Quick overview of the event system codebase**
- **Event Class:**
 - `bitbake/lib/bb/event.py`
- **Event creation examples:**
 - `bitbake/lib/bb/build.py`
- **Event client registration examples:**
 - `bitbake/lib/bb/ui/toasterui.py`
 - `bitbake/lib/bb/ui/knotty.py`
- **Attaching event triggers to bitbake tasks (Toaster)**
 - `meta/classes/toaster.bbclass`
- **Event handler examples, plus database population (Toaster):**
 - `bitbake/lib/bb/ui/buildinfohelper.py`

The Event Class

- The event class is minimal, it only predefines a single member ("pid")

```
class Event(object):  
    """Base class for events"""  
    def __init__(self):  
        self.pid = worker_pid
```

- All events are sub-classed and extended, for example:

```
class TaskBase(event.Event):  
    """Base class for task events"""  
    def __init__(self, t, logfile, d):  
        self._task = t  
        self._package = d.getVar("PF")  
        self._mc = d.getVar("BB_CURRENT_MC")  
        self.taskfile = d.getVar("FILE")  
        self.taskname = self._task  
        self.logfile = logfile  
        self.time = time.time()  
        event.Event.__init__(self)  
        self._message = "recipe %s: task %s: %s" % (d.getVar("PF"),  
            t, self.getDisplayName())  
  
class TaskStarted(TaskBase):  
    """Task execution started"""  
    def __init__(self, t, logfile, taskflags, d):  
        super(TaskStarted, self).__init__(t, logfile, d)  
        self.taskflags = taskflags
```

Event Creation

- Events are easily defined and fired

- Example 1: `bitbake/lib/bb/build.py`

```
def _exec_task(fn, task, d, quieterr):  
    ...  
    event.fire(TaskStarted(task, logfn, flags, localdata), localdata)
```

- Example 2: `meta/classes/toaster.bbclass`

```
python toaster_artifact_dumpdata() {  
    """  
    Dump data about SDK variables  
    """  
    event_data = {  
        "TOOLCHAIN_OUTPUTNAME": d.getVar("TOOLCHAIN_OUTPUTNAME")  
    }  
    bb.event.fire(bb.event.MetadataEvent("SDKArtifactInfo",  
        event_data), d)  
}
```

Event Client Registration Example

- Here is how a bitbake UI registers the events it will handle (Toaster)

```
bitbake/lib/bb/ui/toasterui.py:
```

```
_evt_list = [
    "bb.build.TaskBase",
    "bb.build.TaskFailed",
    "bb.build.TaskFailedSilent",
    "bb.build.TaskStarted",
    "bb.build.TaskSucceeded",
    ...
    "bb.runqueue.sceneQueueTaskCompleted",
    "bb.runqueue.sceneQueueTaskFailed",
    "bb.runqueue.sceneQueueTaskStarted",
    "logging.LogRecord"]

def main(server, eventHandler, params):
    result, error = server.runCommand(["setEventMask",
        server.getEventHandle(),
        llevel, debug_domains, _evt_list])
```

Attaching event triggers to bitbake tasks

- Example 1: meta/classes/toaster.bbclass. Here we hook into a task's post functions.

```
python toaster_artifact_dumpdata() {  
    """  
    Dump data about SDK variables  
    """  
    event_data = {  
        "TOOLCHAIN_OUTPUTNAME": d.getVar("TOOLCHAIN_OUTPUTNAME")  
    }  
    bb.event.fire(bb.event.MetadataEvent("SDKArtifactInfo",  
        event_data), d)  
}  
  
do_populate_sdk[postfuncs] += "toaster_artifact_dumpdata "  
do_populate_sdk[vardepsexclude] += "toaster_artifact_dumpdata "  
  
do_populate_sdk_ext[postfuncs] += "toaster_artifact_dumpdata "  
do_populate_sdk_ext[vardepsexclude] += "toaster_artifact_dumpdata "
```

Attaching event triggers to bitbake tasks (2)

- **Example 2: meta/classes/toaster.bbclass:** Here we use “addhandler” to have our handler be automatically invoked on the selected events.

```
# get list of artifacts from sstate manifest
python toaster_artifacts() {
    if e.taskname in ["do_deploy", "do_image_complete", "do_populate_sdk",
        "do_populate_sdk_ext"]:
        ...
    bb.event.fire(bb.event.MetadataEvent("toaster_artifacts", data), e.data)
}

addhandler toaster_artifacts
toaster_artifacts[eventmask] = "bb.runqueue.runQueueTaskSkipped
    bb.runqueue.runQueueTaskCompleted"
```

Event Receiver Loop Example (Toaster)

- Example: `bitbake/lib/bb/ui/toasterui.py`: The main loop simply waits for events to occur, compares their instance type, and routes them appropriately.
- There is a timeout when waiting for an event so that the UI will not be blocked, and can for example check for CTRL-C's from the user.

```
def main(server, eventHandler, params):  
    ...  
    while True:  
        try:  
            event = eventHandler.waitEvent(0.25)  
            if event is None:  
                continue  
            if isinstance(event, bb.event.HeartbeatEvent):  
                continue  
            if isinstance(event, bb.event.ParseStarted):  
                ...  
                buildinfohelper.set_recipes_to_parse(event.total)  
                continue  
            if isinstance(event, (bb.event.BuildStarted, bb.event.BuildInit)):  
                ...
```




Example 1: Creating a custom command line event analytic tool

Example Custom Command Line Tool

- Given that the event database is an SQL database, it is easy to write custom python scripts to extract and analyze the build analytics
- In this section, we present a minimal python script to access the Toaster event database
- We will also present an example of a more full featured python application that does data analysis on task and recipe parallel execution, as a proof of concept ([source code available](#))
- *Note: you can find the event database table schemas and its member's name, type, and order directly from the Toaster event database, for example:*

```
$ sqlite3 toaster.sqlite  
# sqlite> .tables  
... orm_build ...  
# sqlite> .schema orm_build
```

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 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()))
$
$ python3 sample_toaster_db_read.py
Build=(1, 'qemux86', 'poky', '2.2+snapshot-20170120', '
    2017-01-20 08:02:36.924607', '2017-01-20 08:20:35.548057', 0,
    '/opt/toaster_master/poky/build-toaster-2/tmp/log/cooker/qemux86/
    build_20170120_000245.938.log', '1.32.0', 2, 850, 850, '20170120080256')
Target=(1, 'nativesdk-chrpath', '', 0, 0, None, 1, None)
$
```

Full Feature Event Database Python Script

- In this section we will present 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.

Full Feature Event Database Python Script

- **Here is the methodology**
 - Select a build
 - Read the build's task list (which include each task's start and stop), and also attach all associated parent recipes
 - For each task, find all other tasks whose execution overlaps
 - For each recipe, find all other recipes whose execution overlaps (from its first to last task)
- **Here are some of the goals**
 - Compute histograms of the overlap, understand the nature and extent of overlaps
 - Review and understand the tasks/recipes that have no overlaps
 - Examine areas of low overlap, and see if the parallelism can be improved to reduce build time
 - As a long term goal, compare to intermittent package failures to identify potential race conditions

Task and Recipe Build Analysis Script

- Here is the list of available commands and features

```
python3 ~/lx90/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

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

Histogram of Parallel Task/Recipe Execution

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 | | | |

| | Most viewed | Vind river | Linux | Fedora | Azure | Answer | Vind river Linux | Clinic | XCC | TMC | Google Maps | ELEC | |
|---|-------------|------------|-------|--------|-------|--------|------------------|--------|-----|-----|-------------|------|--------------------------------------|
| - | - | - | - | - | - | - | - | - | - | - | - | - | openssl:do_unpack |
| - | - | - | - | - | - | - | - | - | - | - | - | - | expat:do_patch |
| - | - | - | - | - | - | - | - | - | - | - | - | - | inputproto-native:do_fetch |
| - | - | - | - | - | - | - | - | - | - | - | - | - | libxcb-native:do_fetch |
| - | + | - | - | - | - | - | - | - | - | - | - | + | xtrans-native:do_fetch |
| - | - | - | - | - | - | + | - | - | - | - | - | - | xproto:do_patch |
| - | - | - | - | - | - | - | - | - | - | + | - | - | kbroto-native:do_fetch |
| - | - | - | - | - | - | - | - | - | + | - | - | - | libx11-native:do_fetch |
| - | - | - | - | - | - | - | - | - | - | + | - | - | xtrans:do_fetch |
| - | - | - | + | - | - | - | - | - | - | - | - | - | xcb-proto-native:do_unpack |
| - | - | - | - | - | - | + | - | - | - | - | - | - | ncurses:do_patch |
| - | - | - | - | - | - | - | - | + | - | - | - | - | libpthread-stubs-native:do_unpack |
| - | - | - | - | - | - | - | - | - | - | - | + | - | xextproto-native:do_unpack |
| - | - | - | + | - | - | - | - | - | - | - | - | - | libxau-native:do_unpack |
| - | + | - | - | - | - | - | - | - | - | - | - | - | libpcres:do_patch |
| - | - | - | - | - | - | + | - | - | - | - | - | - | libxdmcp-native:do_unpack |
| - | - | - | + | - | - | - | - | - | - | - | - | - | openssl:do_patch |
| - | - | - | + | - | - | - | - | - | - | - | - | - | lao-native:do_fetch |
| - | - | - | - | - | - | + | - | - | - | - | - | - | xcb-proto-native:do_patch |
| - | - | - | - | - | - | - | - | + | - | - | - | - | libpthread-stubs-native:do_patch |
| - | - | - | - | + | - | - | - | - | - | - | - | - | xtrans:do_unpack |
| - | - | - | - | - | - | - | - | + | - | - | - | - | xextproto-native:do_patch |
| - | + | - | - | - | - | - | - | - | - | - | - | - | xtrans-native:do_unpack |
| - | - | - | - | - | - | + | - | - | - | - | - | - | gdk-doc-native:do_fetch |
| - | - | - | - | - | - | - | - | + | - | - | - | - | inputproto-native:do_unpack |
| - | - | - | - | + | - | - | - | - | - | - | - | - | libxau-native:do_patch |
| - | + | - | - | - | - | - | - | - | - | - | - | - | libxdmcp-native:do_patch |
| - | - | - | - | + | - | - | - | - | - | - | - | - | python3-native:do_fetch |
| - | - | - | - | + | - | - | - | - | - | - | - | - | gettext:do_patch |
| - | - | - | - | - | - | - | + | - | - | - | - | - | kbroto-native:do_unpack |
| - | - | - | - | - | - | - | - | + | - | - | - | - | libpthread-stubs-native:do_configure |
| - | - | - | - | - | - | - | + | - | - | - | - | - | util-linux-native:do_fetch |
| - | - | - | - | - | - | - | - | - | + | - | - | - | bash-completion:do_unpack |
| - | - | - | - | - | - | - | - | - | + | - | - | - | gdcm:do_unpack |
| - | - | - | - | - | - | + | - | - | - | - | - | - | xextproto-native:do_configure |
| - | - | - | - | - | - | - | + | - | - | - | - | - | libxdmcp-native:do_configure |
| - | - | - | - | - | - | - | - | - | - | + | - | - | libxcb-native:do_unpack |
| - | - | - | - | - | - | - | - | - | - | - | - | - | xtrans:do_patch |
| - | - | - | - | - | - | - | - | - | - | - | + | - | util-linux:do_unpack |
| - | - | - | - | - | - | - | - | - | - | + | - | - | xtrans-native:do_patch |



Example 2: 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"]
```



Example 3: 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.

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

```
[build]$ 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

```
- "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  
...  
NOTE: NICE: bb.event.DepTreeGenerated received! | ETA: 0:00:00  
...
```



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.



Gotchas and Resources

Gotchas

- **Event content is very flexible, which both good and bad**
 - If you rely on specific events, you may want to have a unit test to insure that the event content does not change on you
- **Not all events have full or consistent data**
 - Some of the data in the events is not always consistent or complete
 - The Toaster event handlers in fact sometimes need to fall back to some trial and error to resolve that data
 - One example is that for some events, the names for native and/or SDK tasks are expected to have a “virtual:native[sdk]:” prefix, but that prefix is sometimes missing and must be deduced
 - You can examine Toaster’s “buildinfohelper.py” for examples and guidance
- **The event data is buffered to not block bitbake, so it may take an extra few minutes before the data is ready (so do not shut down Toaster!)**

Resources

- **Source code and example event database**
 - This is available as part of the Yocto Project Developer Day Advanced Class <https://www.yoctoproject.org/yocto-project-dev-day-north-america-2017>,
 - 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>
- **Here is a Youtube video on Toaster**
 - <https://youtu.be/BIXdOYLgPxA>
- **Here is the Toaster email list**
 - toaster@yoctoproject.org

Resources (2)


- **Basic information about bitbake UI's**
 - [http://elinux.org/Bitbake Cheat Sheet](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](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](https://wiki.yoctoproject.org/wiki/Toaster_and_bitbake_communications)



Questions and Answers

What data do you want/need?

What analytics use cases do you want to solve?

A decorative pattern of overlapping hexagons in various shades of gray, located in the top-left corner of the slide.

Thank you for your participation!

