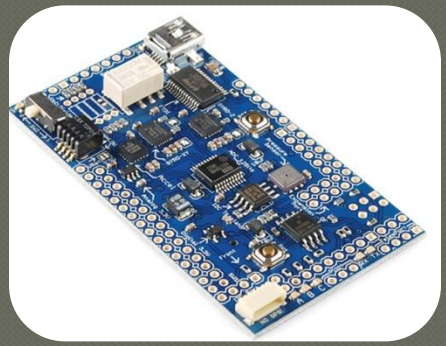
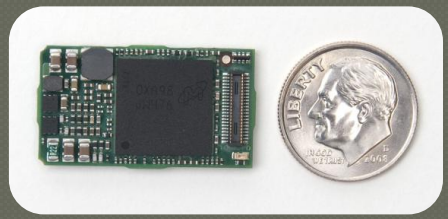
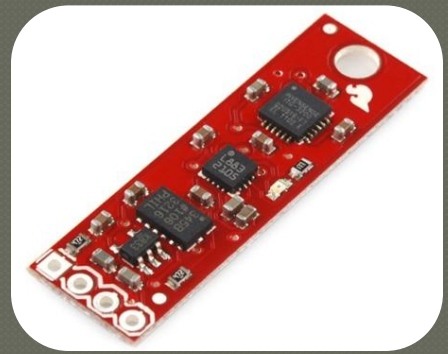
integrating sensors into android hardware

rian sanderson
sensor platforms, inc.

attach sensors to an android dev board

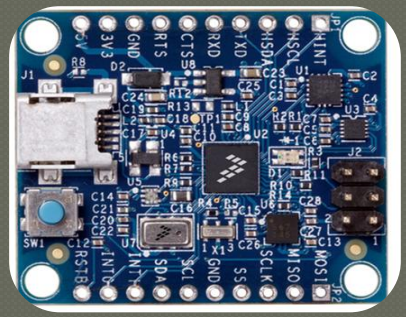






you have an android board, you have a sensor board, you want them to work together

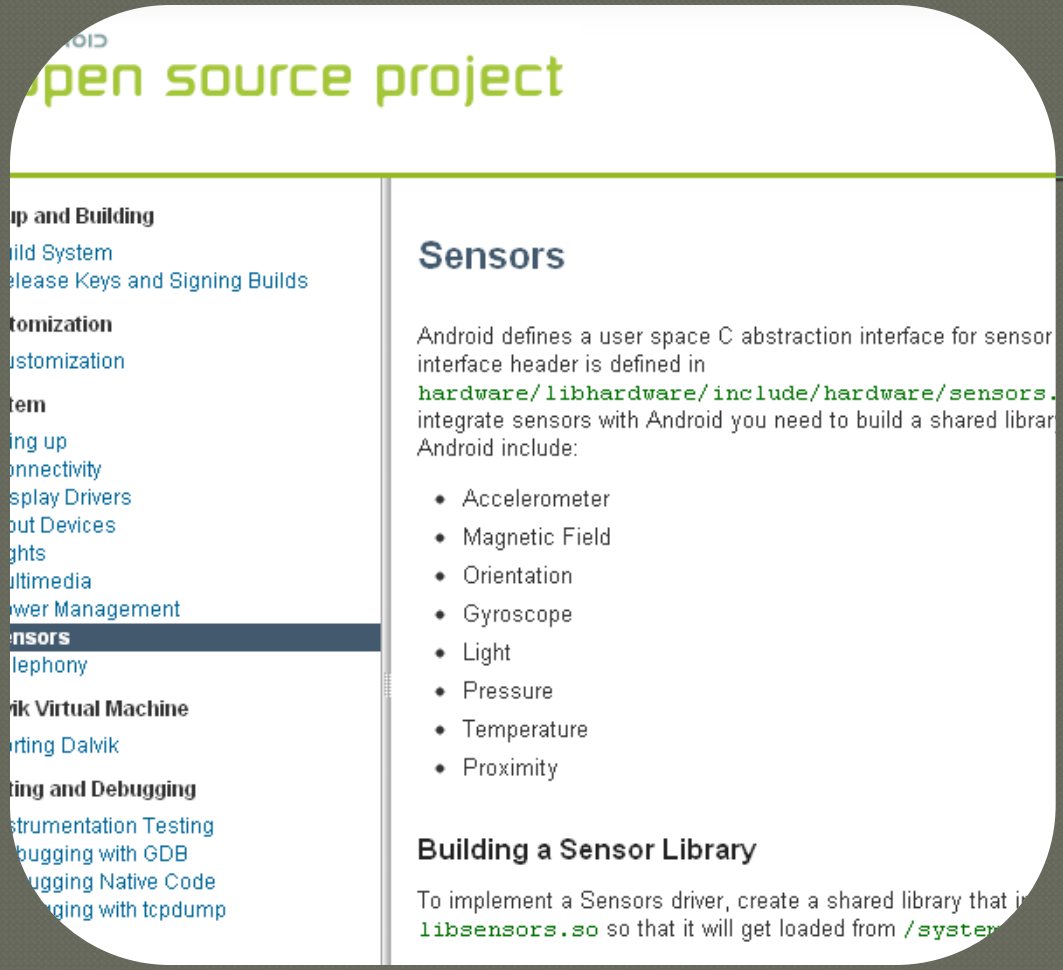
connect off the shelf devices



make external sensors available through standard Androidd·SensorManager APIs

utlize hidden functionality already on your phone

the problem



google this:

integrating android sensor hardware

the source code is out there but not much of the big picture

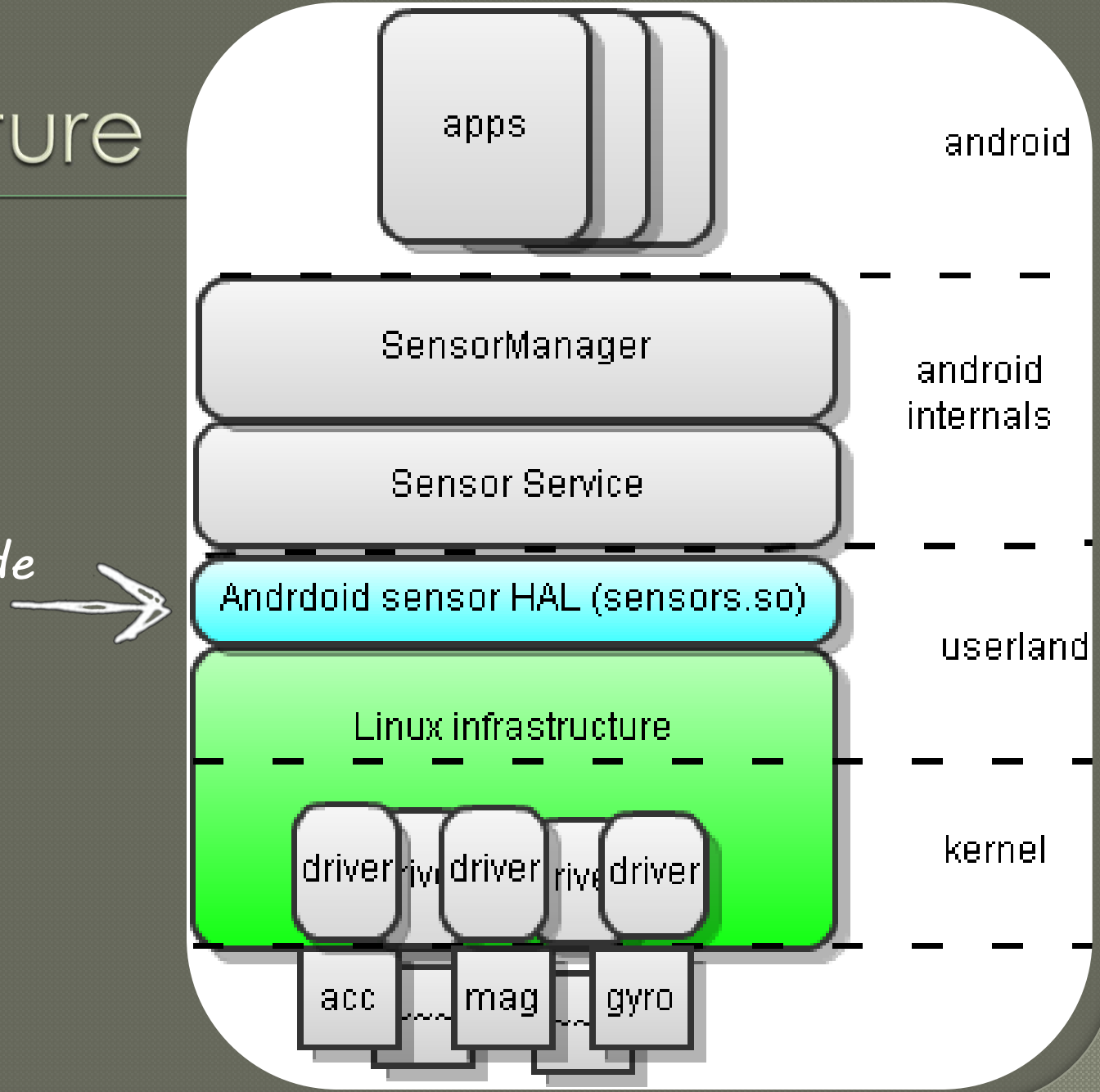
this talk should enable you to

- start experimenting with sensors already on your device
- Integrate new sensors into your device
- debug each of the layers in the sensor stack

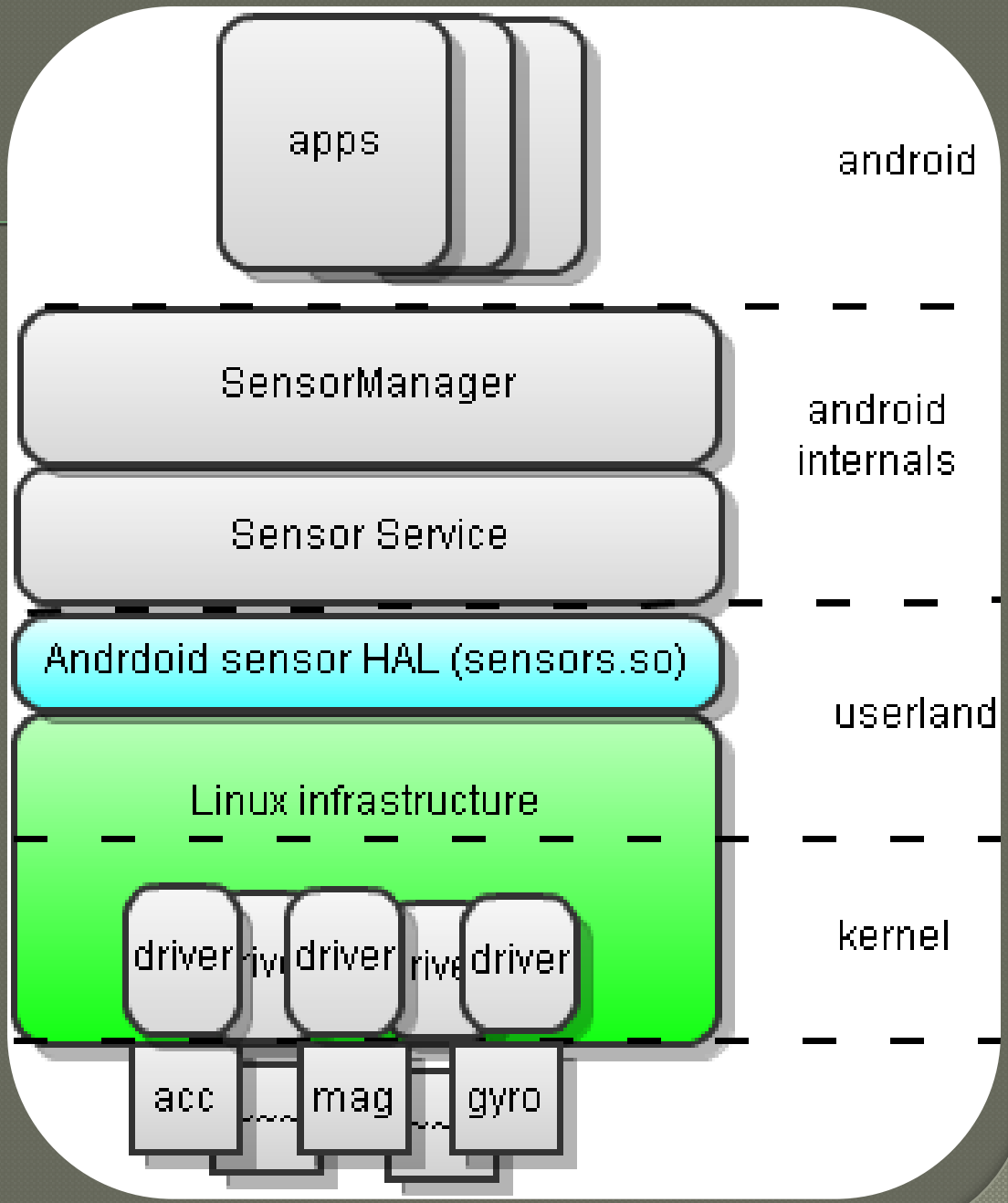
download these slides and check out the speakers notes if you'd like to read more

big picture

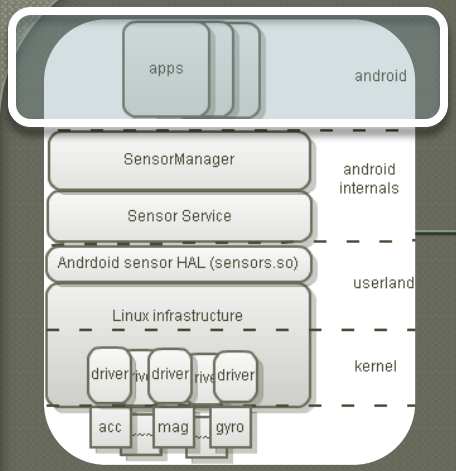
insert your code here



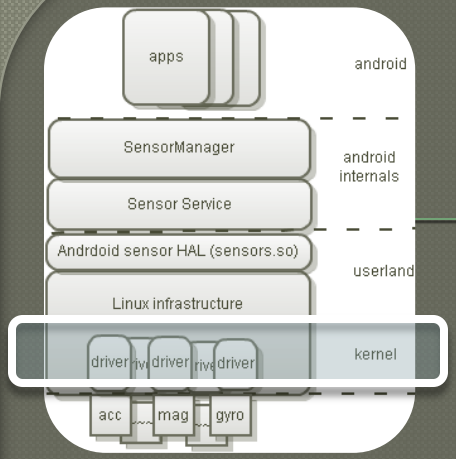
data flow


apps
android
SensorManager
android
internals
Sensor Service
Andrdoid sensor HAL (sensors.so)
userland
Linux infrastructure
driver
driver
driver
kernel
acc
mag
gyro

app level debugging



AndroSensor displays sensor info exactly as an app developer will see it



sensor drivers

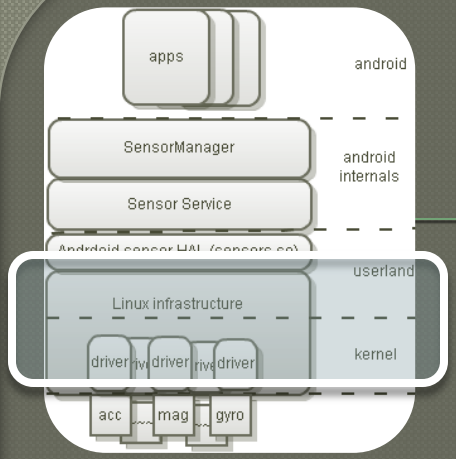
tools for debug

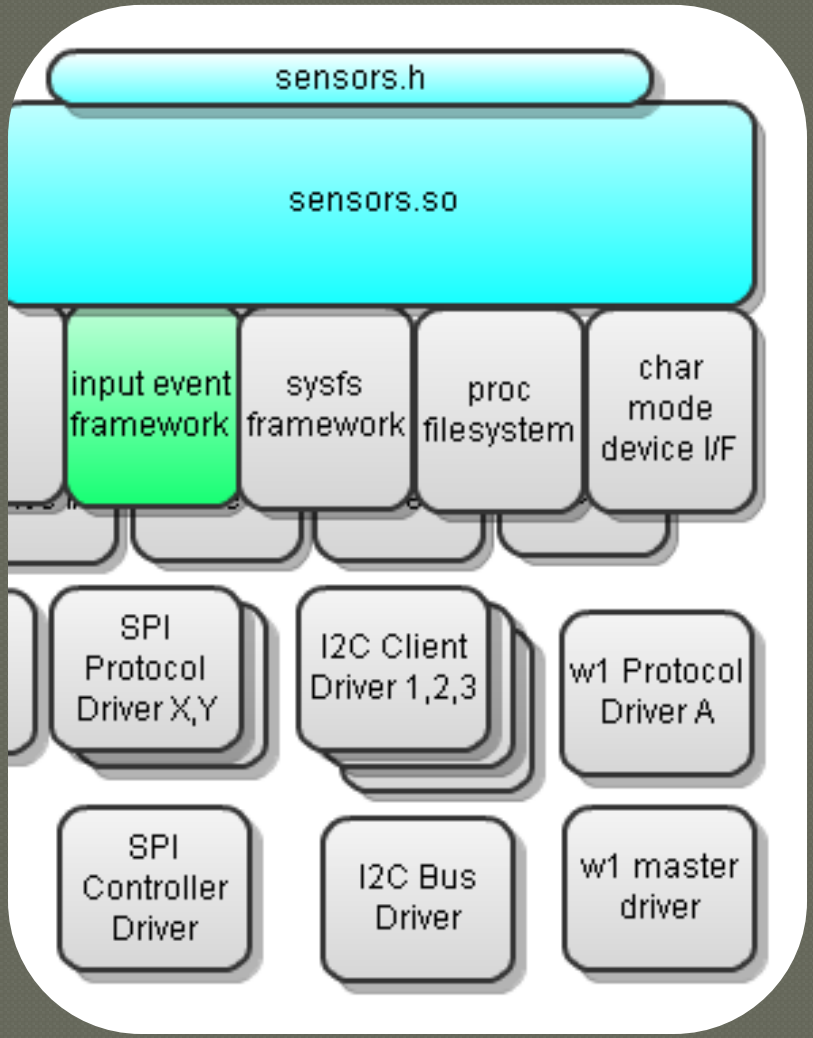
```
dmesg
```

```
adb shell lsmod | grep <your driver>
```

don't write sensors drivers keep them platform agnostic

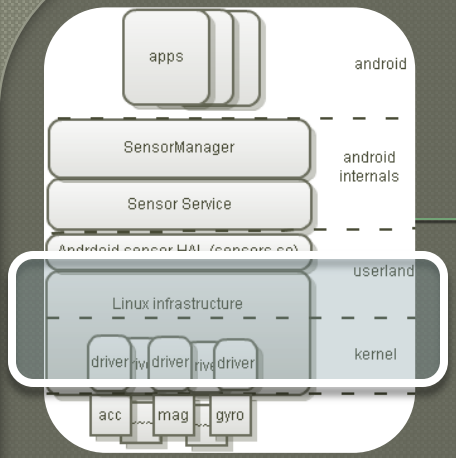
linux infrastructure



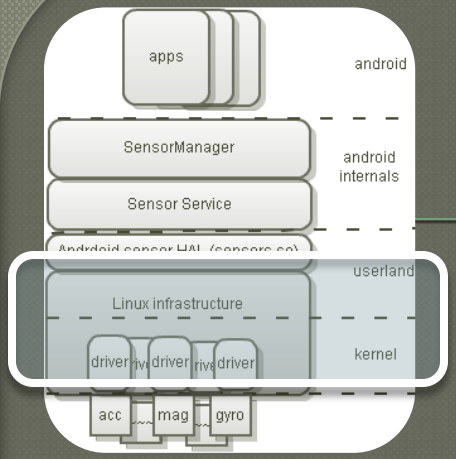


leverage the most appropriate infrastructure

input event framework



presents each sensor as name and a stream of data events



input event processing

```
#include <linux/input.h>
```

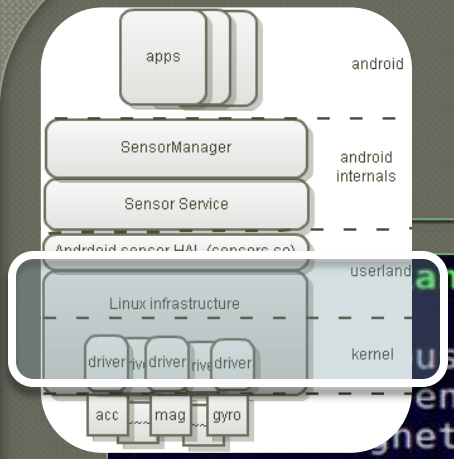
```
struct input_event {
    struct timeval time;
    __u16 type;
    __u16 code;
    __s32 value;
};
```

```
// Event Types
#define EV_SYN   0x00
#define EV_KEY   0x01
#define EV_REL   0x02
#define EV_ABS   0x03
...
// Event Codes
#define REL_X    0x00
#define REL_Y    0x01
#define REL_Z    0x02
...
```

4 event structs per sensor measurement

uses code field to differentiate axis

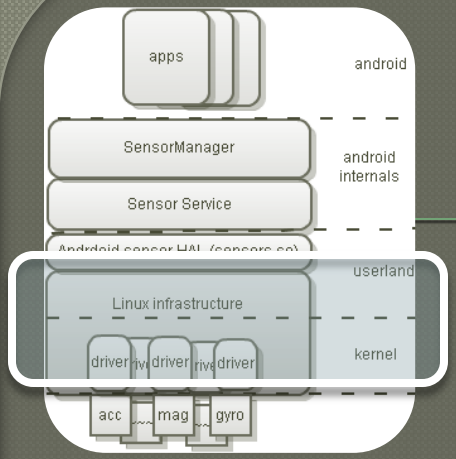
demo: input event framework




```
n-dev:~

s/input/devices
endor=0000 Product=0000 Version=0000
gnetometer"
P: Phys=ami305/input0
S: Sysfs=/devices/platform/i2c_omap.2/i2c-2/2-000e/input/input0
U: Uniq=
H: Handlers=event0
B: EV=d
B: REL=7
B: ABS=7

I: Bus=0019 Vendor=0001 Product=0001
N: Name="gpio-keys"
P: Phys=gpio-keys/input0
S: Sysfs=/devices/platform/gpio-keys
U: Uniq=
H: Handlers=kbd event1
B: EV=3
B: KEY=100000 0 0 0 0 40 10 0 2

: Bus=0000 Vendor=0000 Product=0000
  Name="ADS7846 Touchscreen"
```

```
Bus=0006 Vendor=0591 Product=0001 Version=00
: Name="fm-rotation-vector"
: Phys=FMLIB/fm-rotation-vector
S: Sysfs=/devices/virtual/input/input13
U: Uniq=
H: Handlers=event13
B: EV=d
B: REL=3f
B: ABS=0

# getevent /dev/input/event9
0002 0000 fffffbe4
0002 0001 00000017
0002 0002 00000006
0000 0000 00000000
0002 0000 fffffbe3
0002 0001 0000001a
0002 0002 00000006
0000 0000 00000000
0002 0000 fffffbe3
002 0001 0000001d
02 0002 00000006
  0000 00000000
```

input event framework resources

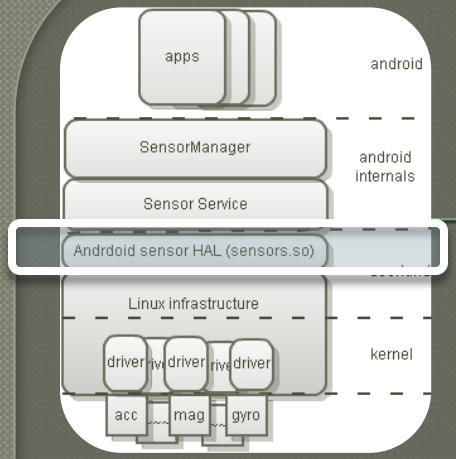


tools for debug:

```
cat /proc/bus/input/devices

getevent /dev/input/event<n>
```

further reading:

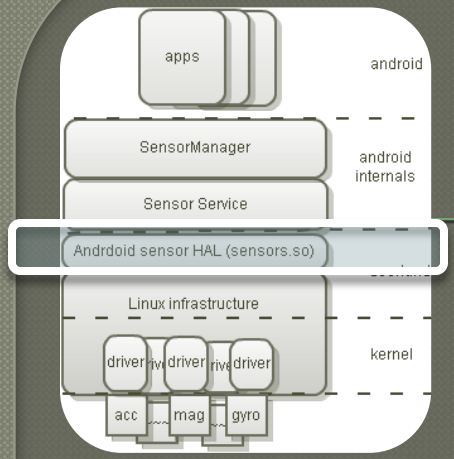
http://www.kernel.org/doc/Documentation/input/
http://www.kernel.org/doc/Documentation/input/event-codes.txt
http://en.wikipedia.org/wiki/Evdev
Internal input event handling in the Linux kernel and the Android userspace



libsensor

- advertises available sensors and makes them available to Sensor Service
- controls sensors and reads data using Linux infrastructure

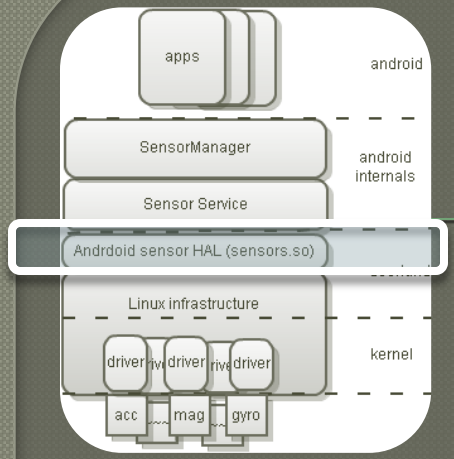
libsensor advertises sensors




```
struct sensors_module_t {
struct hw_module_t common;
int (*get_sensors_list)(struct sensors_module_t* module,
                struct sensor_t const** list);
};
```

`get_sensors_list` *is a function pointer which fills a list of* **`sensor_t`** *strycts describing the sensors on a board*

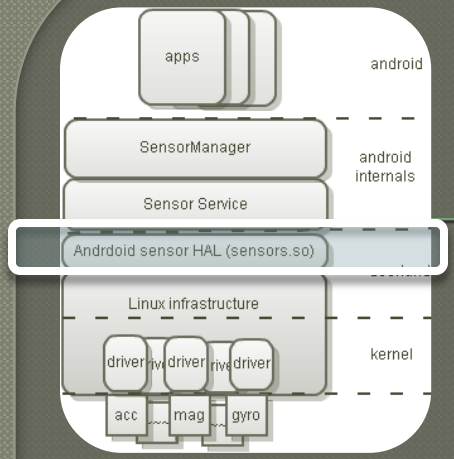
libsensor makes sensors available




```
struct sensors_control_device_t {
    struct hw_device_t common;
    int (*activate)(struct sensors_control_device_t *dev,
            int handle, int enabled);
    int (*set_delay)(struct sensors_control_device_t *dev, int32_t ms);
    int (*wake)(struct sensors_control_device_t *dev);
};
struct sensors_data_device_t {
    struct hw_device_t common;
    int (*data_open)(struct sensors_data_device_t *dev, native_handle_t* nh);
    int (*data_close)(struct sensors_data_device_t *dev);
    int (*poll)(struct sensors_data_device_t *dev,
            sensors_data_t* data);
```

activate(), set_delay(), poll() are the key methods

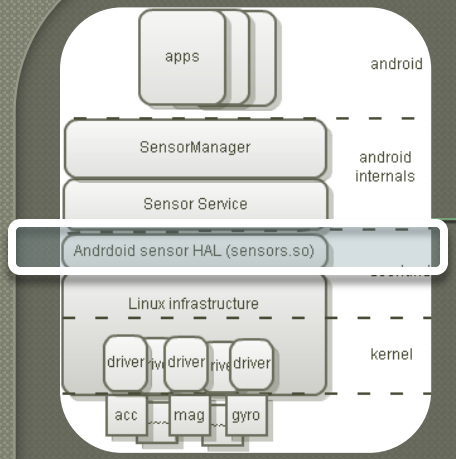
libsensor reads sensor drivers




```
typedef struct {
  union {
    float v[3];
    struct {
      float x;
      float y;
      float z;
    };
...
```

```
struct input_event {
    struct timeval
    time;
    __u16 type;
    __u16 code;
    __s32 value;
};
```

- translate from 4 input events to 1sensors_vect_t
- put into physical units: deg C, m/s^2 ...

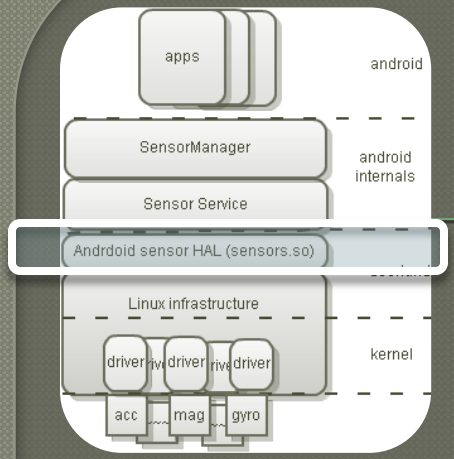


libsensor

```
#include <hardware/sensors.h>
```

- advertises available sensors and makes them available to Sensor Service
- controls sensors and reads data using Linux infrastructure

adapt an existing implementation

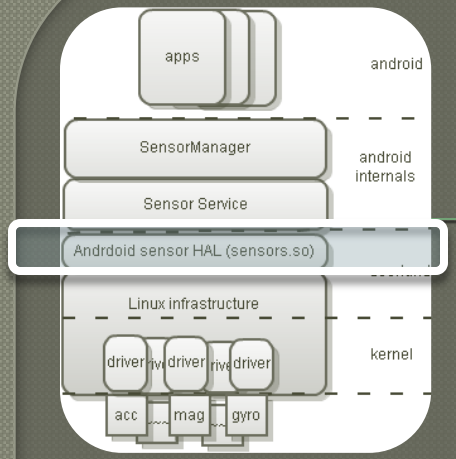


rowboat / hardware-libhardware / include / hardware / sensors.h

rowboat / hardware-ti-omap3 / rowboat-gingerbread / libsensors / sensors.cpp

this is the most common implementation,
and is extendable for input event drivers

root of the implementation

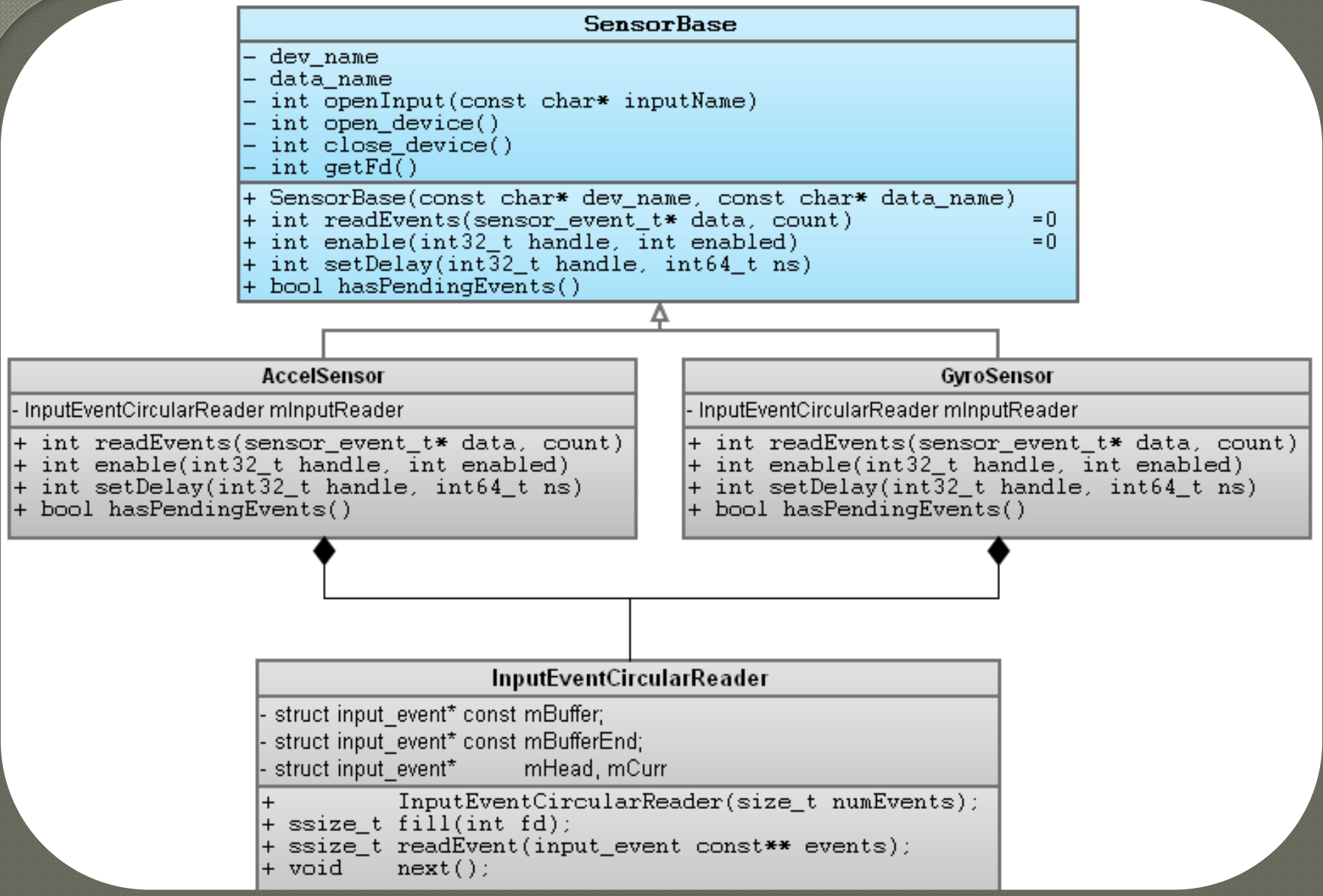


SensorBase

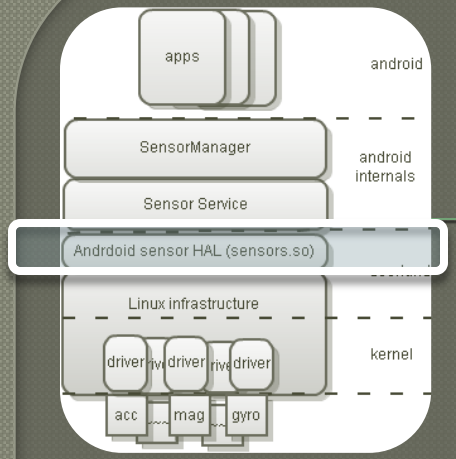
```
- dev_name
- data_name
- int openInput(const char* inputName)
- int open_device()
- int close_device()
- int getFd()
```

```
+ SensorBase(const char* dev_name, const char* data_name)
+ int readEvents(sensor_event_t* data, count)              =0
+ int enable(int32_t handle, int enabled)                  =0
+ int setDelay(int32_t handle, int64_t ns)
+ bool hasPendingEvents()
```

best thing about it is that it works



copy/paste if you have an input event driver



libsensor

source code

rowboat / hardware-ti-omap3 / rowboat-gingerbread / libsensors

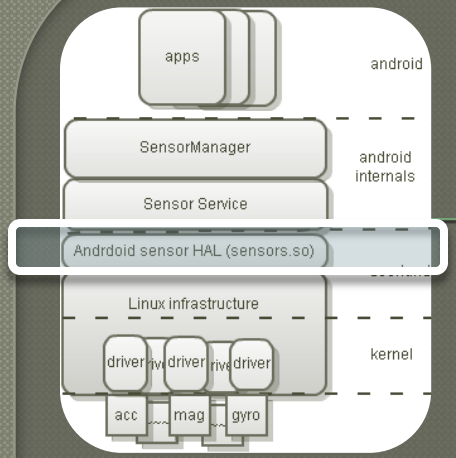
root / device / samsung / crespo / libsensors

OpenEtna / android_device_lg_eve / android_device_lg_eve / libsensors

further reading

http://www.kandroid.org/online-pdk/guide/sensors.html

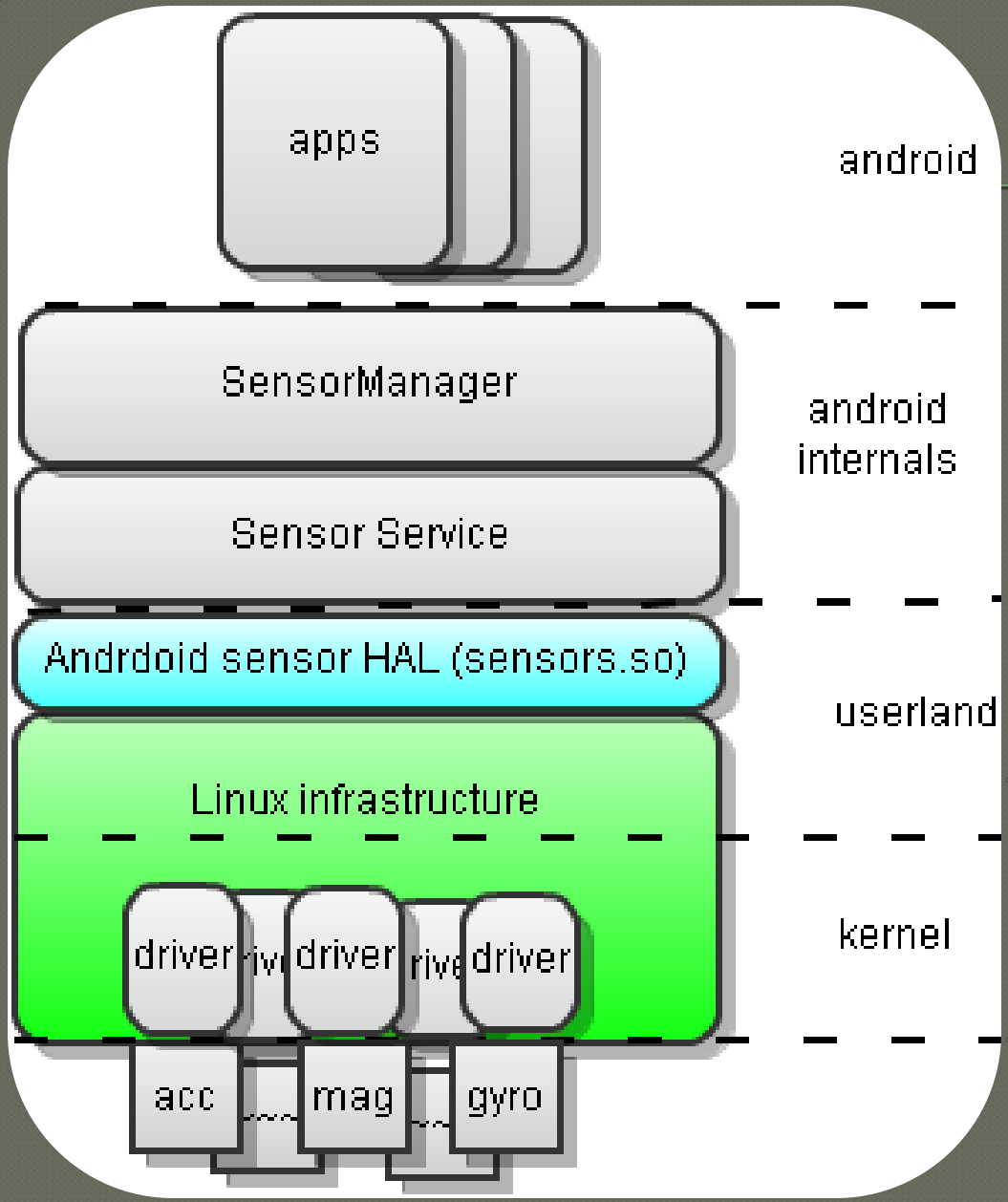
linux infrastructure links



further reading

- Android Sensor PortingGuide
- Linux Industrial I/O Subsystem
- kernel.org/.../w1/w1.generic
- kernel.org/.../spi/spi-summary
- kernel.org/.../i2c/summary
- Getting Started With UInput

remember



implement the glue between Android and Linux

leverage existing Linux infrastructure

keep drivers platform agnostic

what's next

now

- Key Lime Pie, Ice Cream Sandwich, Gingerbread – same HAL API
- Non input-framework based drivers: IIO
- Sensor Fusion daemons
- Dedicated Sensor Processors

future

- Open Sensor Processing standards
- Sensor Fusion going beyond just orientation

questions?

rsanderson@sensorplatforms.com