

# Android System Development Day-2

By Team Emertxe



# Table of Content

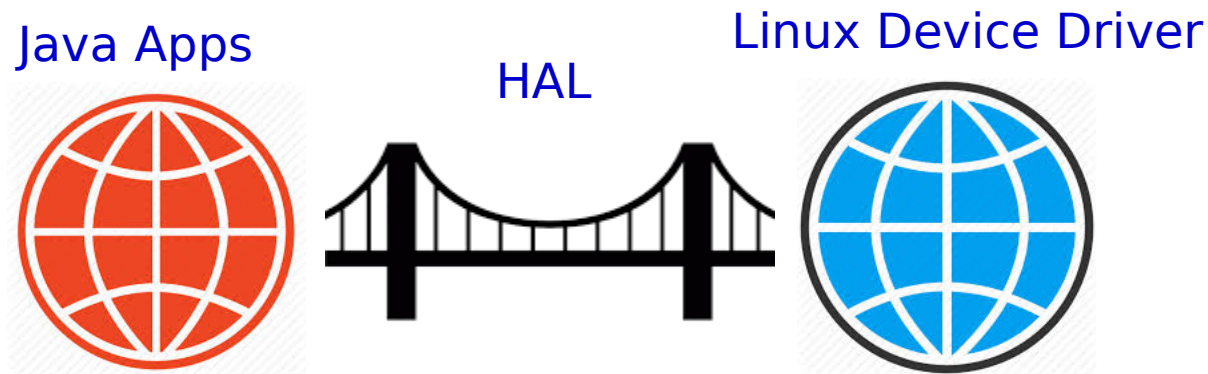


- Introduction to HAL
- Android HAL (sensors)
  - Android Architecture for Sensors
  - Sensors Overview
  - Android Sensor Framework
  - Adding support for a new sensor
    - Enabling IIO for sensors in kernel
    - Verifying with test application for Sensor
  - Compiling sensor HAL
    - Adding permissions for device

# Android HAL Overview

# Why Android HAL?

- Linux is designed to handle only systems calls from application
- Android frameworks communicates with the underlying hardware through Java APIs not by system calls
- Android HAL **bridges** the gap between Android app framework and Linux device driver interface
- The HAL allows you to implement functionality without affecting or modifying higher level system

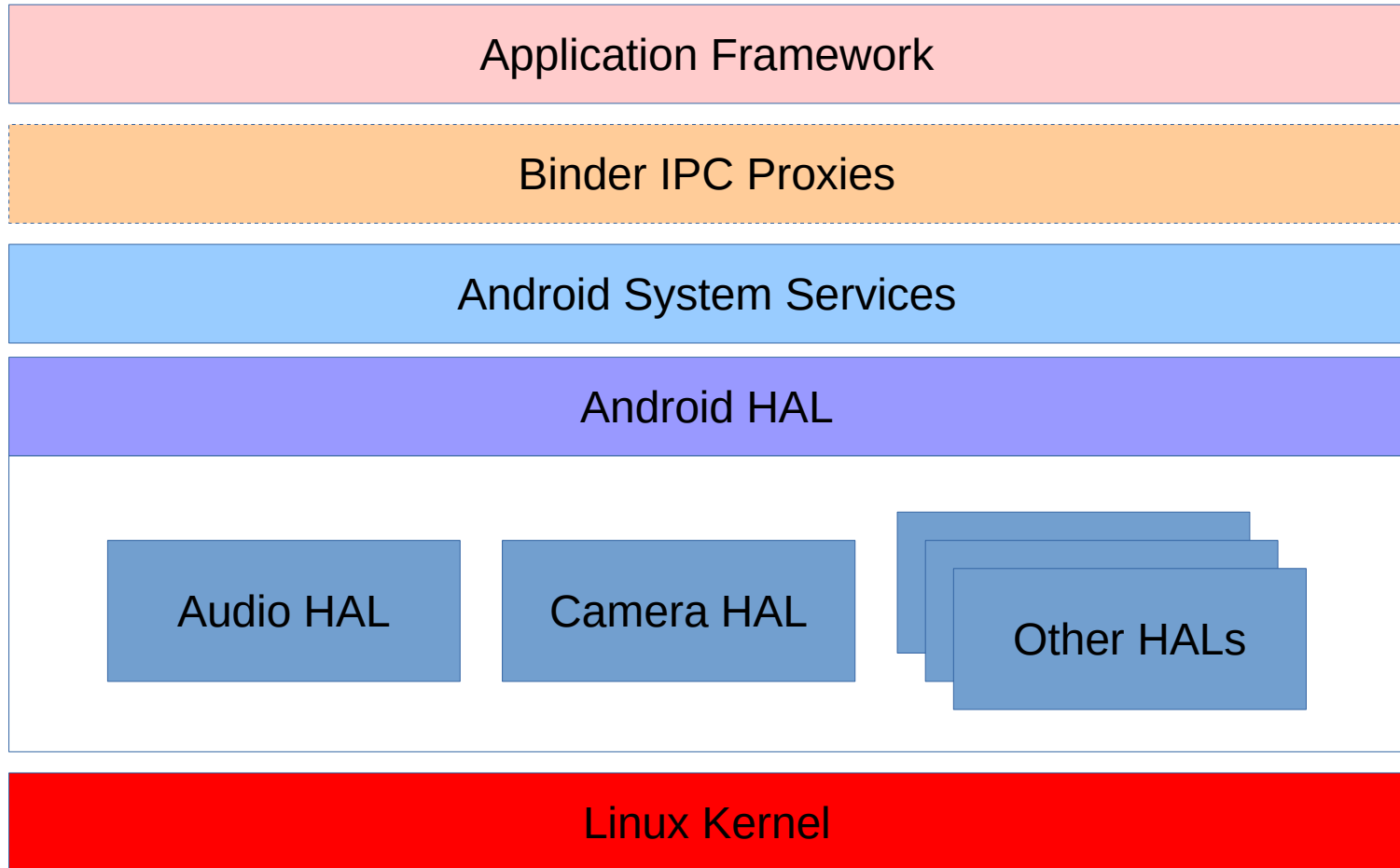


# What is Android HAL?

**“The hardware abstraction layer (HAL) defines a standard interface for hardware vendors to implement and allows Android to be agnostic about lower-level driver implementations”**

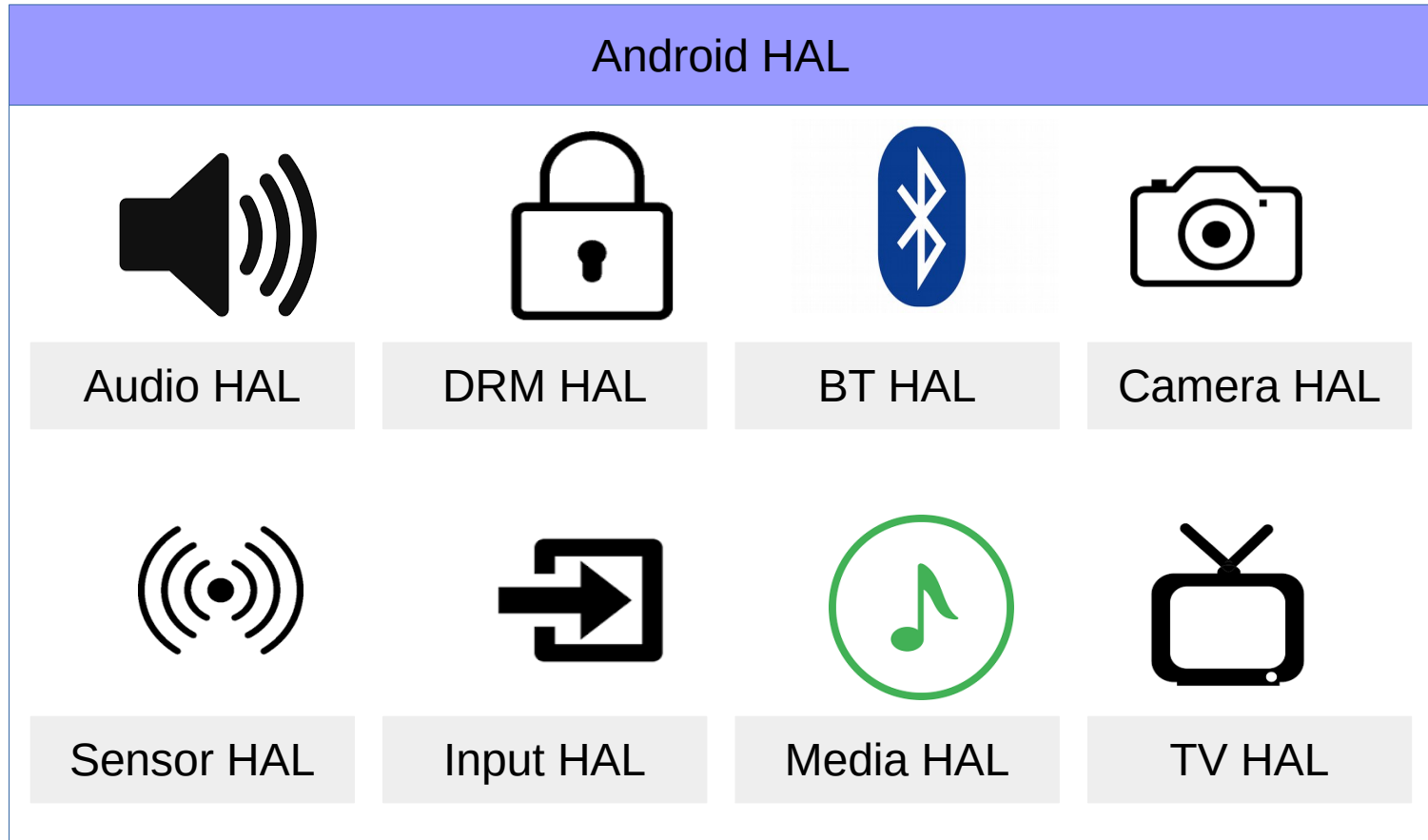
- Provides API's through which Android service can place a request to device
- Uses functions provided by Linux system to service the request from android framework
- A C/C++ layer with purely vendor specific implementation
- Packaged into modules (.so) file & loaded by Android system at appropriate time

# Android System (Architecture)



# Android HAL

(Architecture)



# Android HAL

(Architecture)

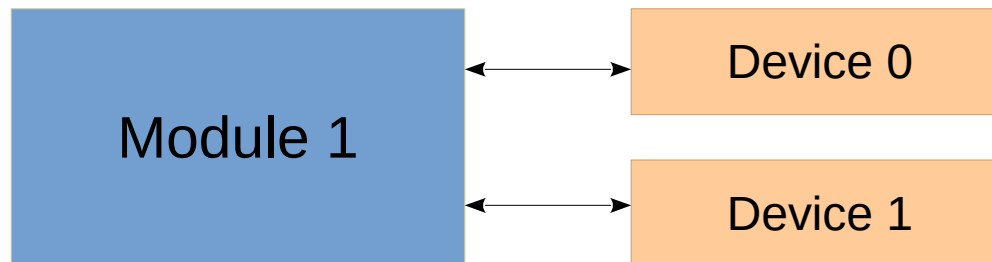
- Each hardware-specific HAL interface has properties that are defined in [hardware/libhardware/include/hardware/hardware.h](#)
- It guarantees that HALs have a predictable structure
- Above interface allows Android system to load correct versions of your HAL modules consistently



# Android HAL

(Architecture)

- HAL interface consists of two general components
  - ✓ Module - Android HAL - automatically loaded by the dynamic linker (sensor.rpi3.so)
  - ✓ Device - Device Specific HAL (provides complete abstraction and control over device vendor) - appropriately loaded at run time



# Android HAL

(Module)

- A module, stored as a shared library (.so file), represents packaged HAL implementation
- Contains metadata such as version, name, and author of the module, which helps Android find and load it correctly
- The “hardware.h” header file defines struct [hw\\_module\\_t](#)
- This structure represents a module & contains information such as module version, author and name
- API are in <aosp>/hardware/libhardware/include/hardware

# Android HAL

(Module...)

- In addition, `hw_module_t` struct contains a pointer to another struct, `hw_module_methods_t`, that contains a pointer to an "open" function for the module
- “open” function is used to initiate communication with the hardware
- Each “hardware-specific HAL” usually extends generic `hw_module_t` struct with additional information for that specific piece of hardware

# Android HAL

(Module hw\_module\_t)



#	Member	Type	Description
1	tag	Integer	HARDWARE_MODULE_TAG
2	module_api_version	Integer	Module interface version (Minor + Major)
3	hal_api_version	Integer	Meant to version module, module methods and device
4	id	String	Ex - "DHT11"
5	name	String	Ex - "Temperature Sensor"
6	author	String	Ex - "Emertxe"
7	methods	Pointer	Open method
8	dso	Pointer	Pointer to Dynamic Shared Objects
9	Reserved	Bytes	Reserved 128 bytes for future use

# Android HAL

(Module...)

- For example in the camera HAL, the camera\_module\_t struct contains a hw\_module\_t struct along with other camera-specific function pointers

```
typedef struct camera_module {  
    hw_module_t common;  
    int (*get_number_of_cameras)(void);  
    int (*get_camera_info)(int camera_id, struct camera_info *info);  
} camera_module_t;
```

# Android HAL

(Naming a module)

- Use **HAL\_MODULE\_INFO\_SYM** name while creating module in your HAL
- Example : Audio module

```
struct audio_module HAL_MODULE_INFO_SYM = {  
    .common = {  
        .tag = HARDWARE_MODULE_TAG,  
        .module_api_version = AUDIO_MODULE_API_VERSION_0_1,  
        .hal_api_version = HARDWARE_HAL_API_VERSION,  
        .id = AUDIO_HARDWARE_MODULE_ID,  
        .name = "NVIDIA Tegra Audio HAL",  
        .author = "The Android Open Source Project",  
        .methods = &hal_module_methods,  
    },  
};
```

# Android HAL

(Device)

- A device abstracts the actual hardware of your product
- Example:
  - ✓ An audio module can contain a primary audio device (ear-jack), a USB audio device, or a Bluetooth A2DP audio device
  - ✓ Different printer models from same company
- A device is represented by the `hw_device_t` structure
- APIs are in `<aosp>/hardware/libhardware_legacy/include/hardware_legacy`

# HAL

(Module hw\_device\_t)

#	Member	Type	Description
1	tag	Integer	HARDWARE_DEVICE_TAG
2	version	Integer	Device API version
3	module	Pointer	Reference to module hw_module_t
4	Padding	Integer	Reserved 48 bytes for future use
5	close	Pointer	To close function



# Android HAL

(Device...)

- Like a module, each type of device defines a more-detailed version of the generic `hw_device_t` that contains function **pointers for specific features** of the hardware
- Example : the `audio_hw_device_t` struct type contains function pointers to audio device operations

```
struct audio_hw_device {  
    struct hw_device_t common;  
    ...  
    uint32_t (*get_supported_devices)(const struct audio_hw_device *dev);  
    ...  
};  
typedef struct audio_hw_device audio_hw_device_t;
```

# Android HAL

## (Structure)

- LDD is a HAL for Linux, therefore, Android HAL looks similar to a Linux device driver
- Most of the **Vendor specific** implementations can be done in Android HAL (rather than the driver)
- Therefore, the **license** difference between driver (Open source license GPL) and HAL (Apache License) will give more level of abstraction to vendor
- The driver triggers hardware (say sensor) and deliver the data back to HAL which is passed back to Android application

# Android Sensors Overview



# Sensors Overview

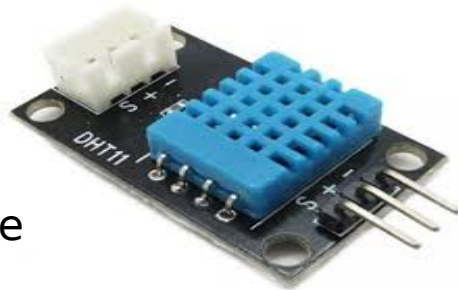
(What?)

- A device that responds to a physical stimulus (as heat, light, sound, pressure, magnetism, or a particular motion) and transmits a resulting impulse (as for measurement or operating a control)
  - Merriam Webster Dictionary

# Sensors Overview

(What?)

- A sensor is a device that detects and responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure, or any one of a great number of other environmental phenomena. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing
- Examples – temperature, motion, light, gravity etc..



Temperature  
sensor

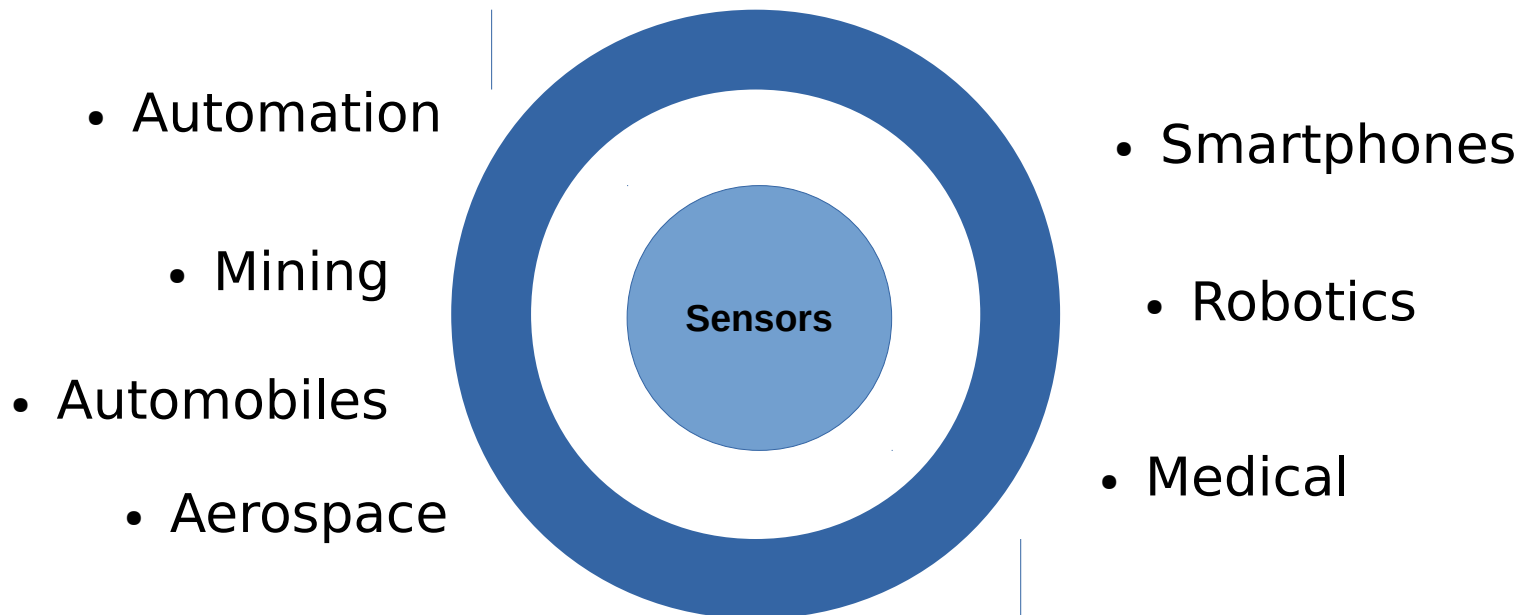


Light  
sensor

# Sensors Overview

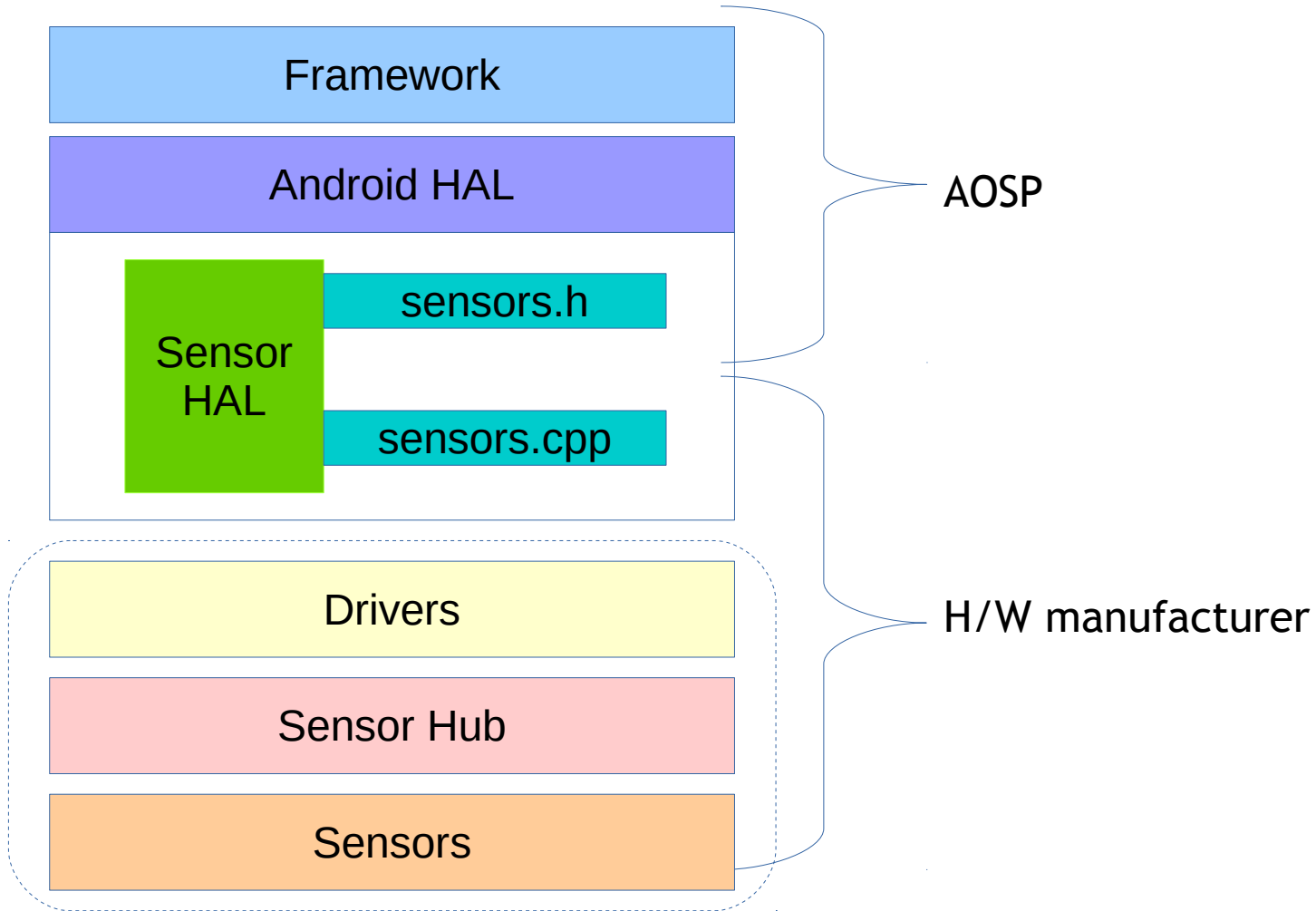
(Why?)

- Sensors are widely used in medical, automation, mining, automobiles, aerospace, robotics, smartphones, houses, farming and more...
- The data is collected, processed and results are used in important decision making and actions



# Sensors overview

(Android Sensor Stack)



# Sensors Overview

## (Application)

- App access sensors through APIs
- App shall register a listener to a sensor
- App specifies its preferred sampling frequency and its latency requirements
- Example :
  - ✓ Register with accelerometer
  - ✓ Request events at 100Hz
  - ✓ Events to be reported with 1-second latency
- The application will receive events from the accelerometer at a rate of at least 100Hz, and possibly delayed up to 1 second



# Sensors Overview

## (Framework)

- Framework links several applications to HAL
- HAL itself is single-client
- Requests are multiplexed by framework to enable access to sensors by many apps
- When first app registers to a sensor, the framework sends a request to the HAL to activate the sensor
- Framework sends updated requested parameters to HAL for additional registration requests from other apps to same sensor
- Framework deactivates the sensor on exit of last app to avoid unwanted power consumption

# Sensors Overview

## (Framework)

- **Final Sampling frequency** - max of all requested sampling frequencies
- Meaning, some applications will receive events at a frequency higher than the one they requested
- **Final maximum reporting latency** – min of all requested ones
- If one app requests one sensor with a maximum reporting latency of 0, all applications will receive the events from this sensor in continuous mode even if some requested the sensor with a non-zero maximum reporting latency

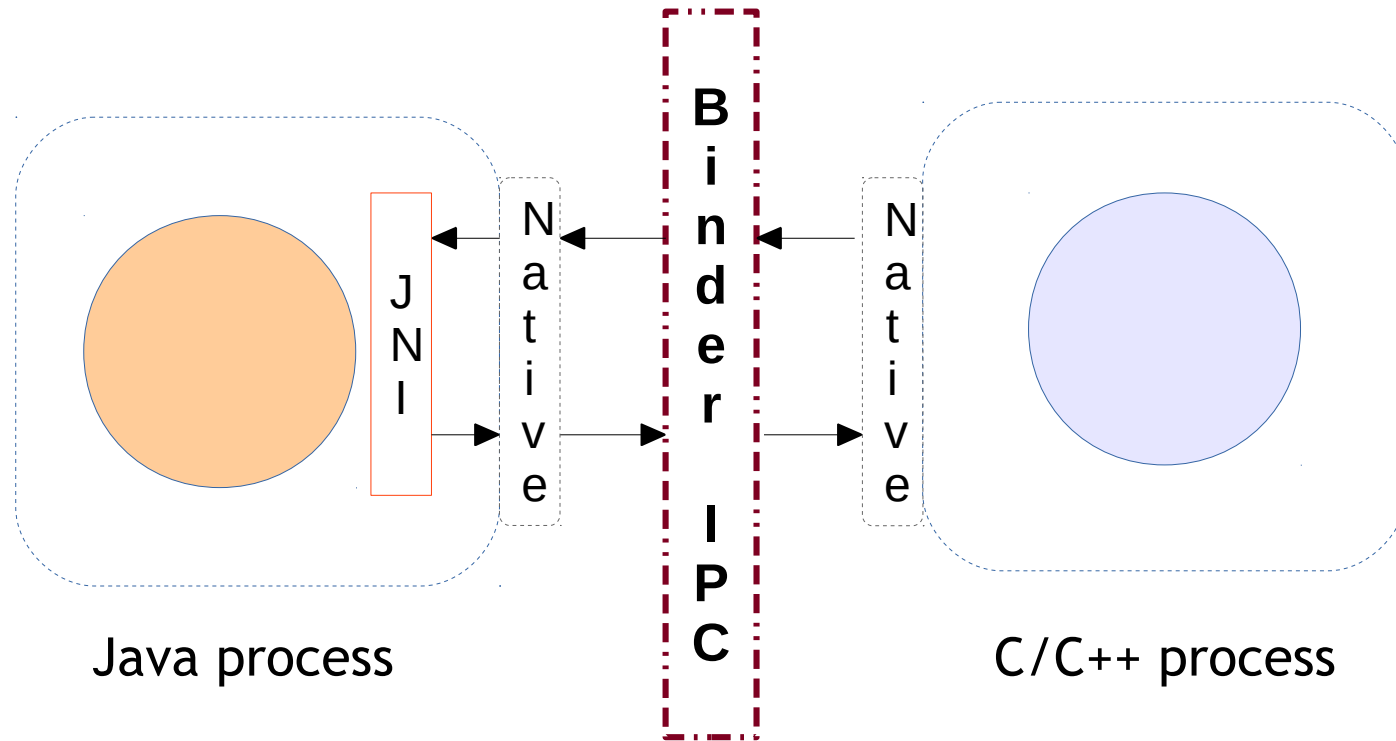
# Sensors Overview

(Implications of multiplexing)

- No guarantee that events won't arrive at a faster rate
- No mechanism to send data down from the apps to sensors or their drivers
- This ensures one app cannot modify the behaviour of sensors and breaking other apps

# Sensors Overview

(The communication)



- \* JNI is located in frameworks/base/core/jni/ directory
- \* Native framework is located in frameworks/native/

# Sensors Overview

## (Types)

### **Motion sensors**

- These sensors measure acceleration forces and rotational forces along three axes
- This category includes [accelerometers](#), [gravity](#) sensors, [gyroscopes](#), and [rotational vector](#) sensors

### **Environmental sensors**

- These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity
- This category includes [barometers](#), [photometers](#), and [thermometers](#)

### **Position sensors**

- These sensors measure the physical position of a device
- This category includes [orientation](#) sensors and [magnetometers](#)

# Sensors Overview

## (Implementation)

### Hardware-based

- Physical components built into a handset or tablet
- Derive their data by directly measuring specific environmental properties such as acceleration, geomagnetic field strength, or angular change

### Software-based

- Are **not** physical devices, although they mimic hardware-based sensors
- Derive their data from one or more of the hardware-based sensors and are sometimes called **virtual** sensors or **synthetic** sensors
- The **linear acceleration** sensor and the **gravity** sensor are examples of software-based sensors

# Sensors Overview

## (Types)

Sensor	Type	Description	Common Uses
Accelerometer	Hardware	Measures the acceleration force in $\text{m/s}^2$ that is applied to a device on all three physical axes (x, y, and z), including the force of gravity	Motion detection (shake, tilt, etc.)
Ambient Temperature	Hardware	Measures the ambient room temperature in degrees Celsius ( $^{\circ}\text{C}$ ). See note below	Monitoring air temperatures
Gravity	Software or Hardware	Measures the force of gravity in $\text{m/s}^2$ that is applied to a device on all three physical axes (x, y, z)	Motion detection (shake, tilt, etc.)
Gyroscope	Hardware	Measures a device's rate of rotation in $\text{rad/s}$ around each of the three physical axes (x, y, and z)	Rotation detection (spin, turn, etc.)

# Sensors Overview

## (Types)

Sensor	Type	Description	Common Uses
Light	Hardware	Measures the ambient light level (illumination) in lx	Controlling screen brightness
Linear Acceleration	Software or Hardware	Measures the acceleration force in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity	Monitoring acceleration along a single axis
Magnetic	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in $\mu T$	Creating a compass
Orientation	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method	Determining device position



# Sensors Overview

## (Types)

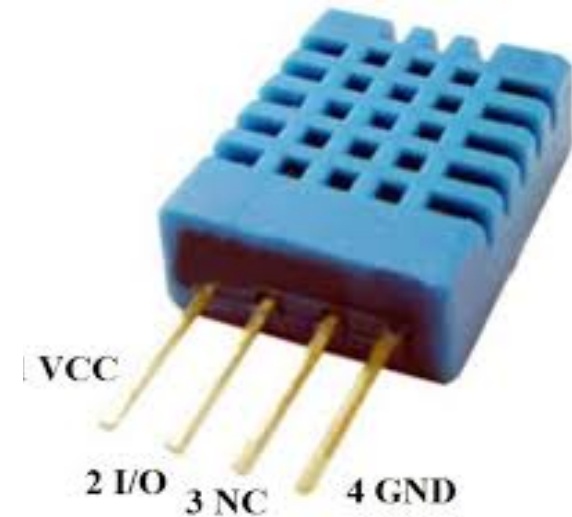
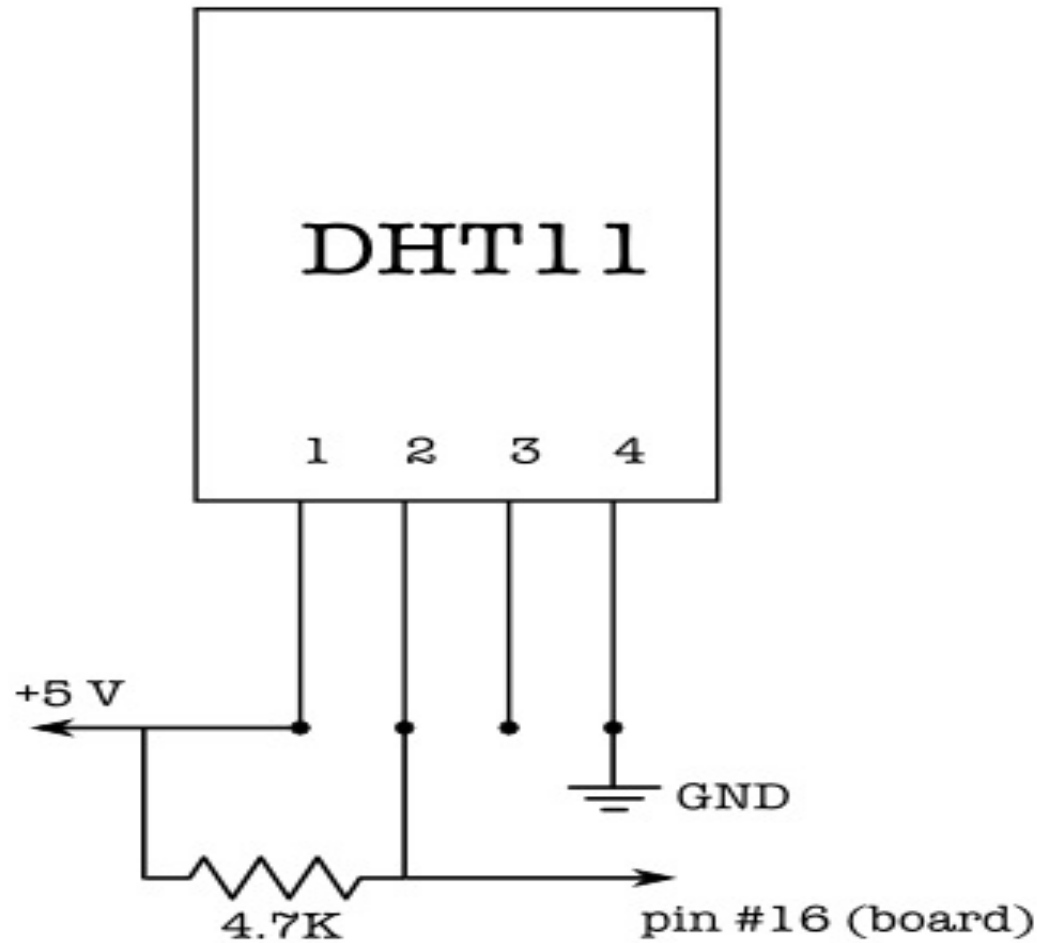
Sensor	Type	Description	Common Uses
Pressure	Hardware	Measures the ambient air pressure in hPa or mbar	Monitoring air pressure changes
Proximity	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear	Phone position during a call
Humidity	Hardware	Measures the relative ambient humidity in percent (%)	Monitoring dewpoint, absolute, and relative humidity
Rotation Vector	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector	Motion detection and rotation detection

# Sensors Overview

## (Supported Sensors)

Sensor	Supported	Sensor	Supported
Accelerometer	Yes	Light	Yes
Ambient Temperature	Yes	Linear Acceleration	Yes
Gravity	Yes	Magnetic	Yes
Gyroscope	Yes	Orientation	Yes
Pressure	Yes	Humidity	Yes
Proximity	Yes	Rotation Vector	Yes

# Temperature Sensor (DHT11)



# Android Sensor Framework



# Sensor Framework

- The sensor framework is part of the [android.hardware](#) package and includes the following classes and interfaces
  - ✓ SensorManager
  - ✓ Sensor
  - ✓ SensorEvent
  - ✓ SensorEventListener

# Sensor Framework

## **Android Sensor Framework can be used for -**

- Determine which sensors are available on a device
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements and resolution
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data
- Register and unregister sensor event listeners that monitor sensor changes

# Sensor HAL

## (Integration)

- Step 1 – Make sure sensor is enabled in kernel
- Step 2 – Ensure it is functioning in user space
- Step 3 – Create source files, write Android.mk
- Step 4 – Compile code & copy shared library to target
- Step 5 - Test your library with Java app

# Sensor HAL

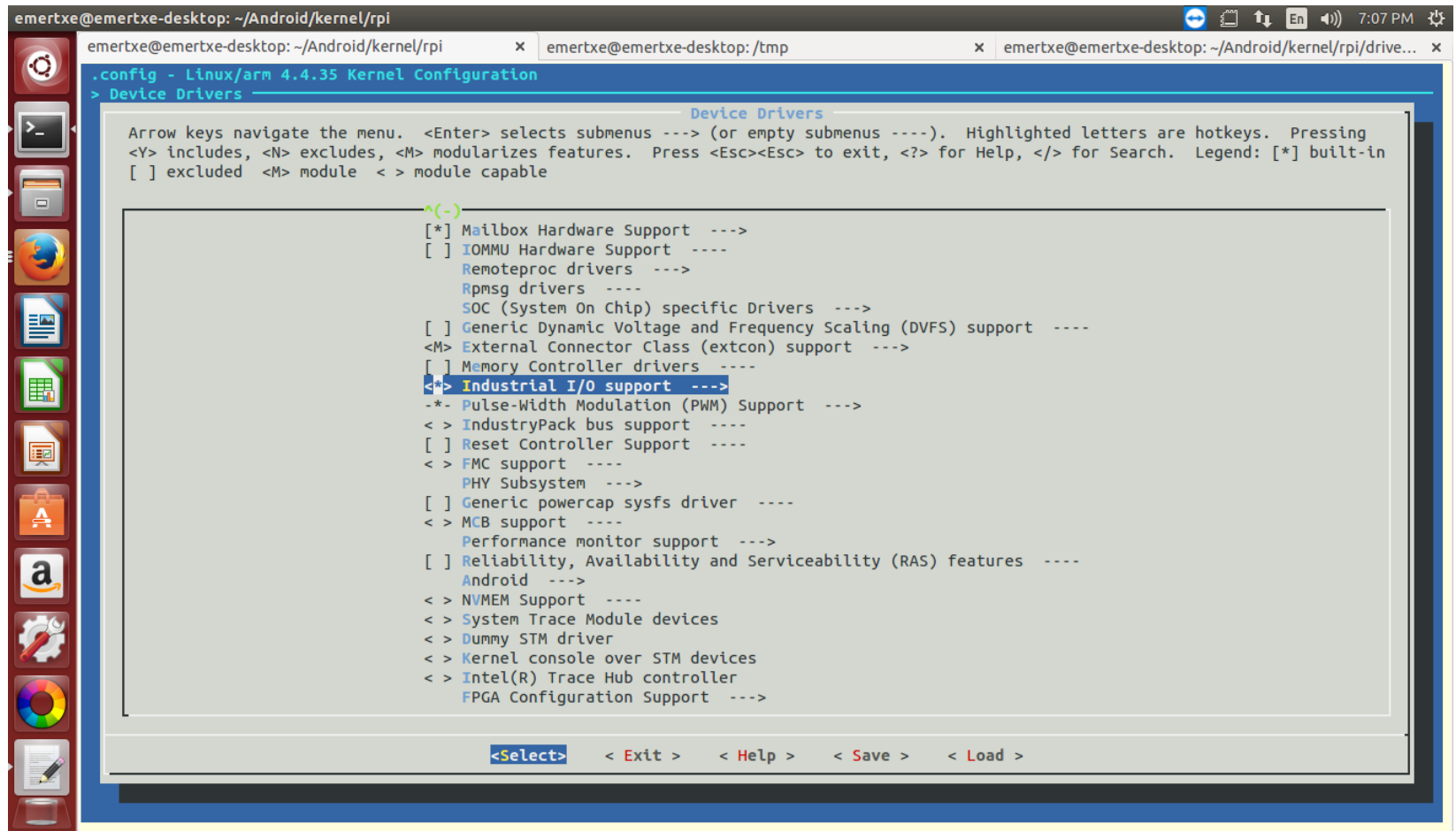
(Enable DHT11 in Kernel)

- make ARCH=arm menuconfig
- Follow instructions as shown in next slides
- Or add CONFIG\_DHT11=y



# Sensor HAL

## (Enable IIO in Kernel)



The screenshot shows a terminal window titled "emertxe@emertxe-desktop: ~/Android/kernel/rpi" running the "make menuconfig" utility. The main window displays the "Device Drivers" menu. At the top, instructions state: "Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [\*] built-in [ ] excluded <M> module < > module capable". The menu items are as follows:

- [\*] Mailbox Hardware Support --->
- [ ] IOMMU Hardware Support ----
- Remoteproc drivers --->
- Rpmsg drivers ----
- SOC (System On Chip) specific Drivers --->
- [ ] Generic Dynamic Voltage and Frequency Scaling (DVFS) support ----
- <M> External Connector Class (extcon) support --->
- [ ] Memory Controller drivers ----
- [\*] **Industrial I/O support --->**
- \* Pulse-Width Modulation (PWM) Support --->
- < > IndustryPack bus support ----
- [ ] Reset Controller Support ----
- < > FMC support ----
- PHY Subsystem --->
- [ ] Generic powercap sysfs driver ----
- < > MCB support ----
- Performance monitor support --->
- [ ] Reliability, Availability and Serviceability (RAS) features ----
- Android --->
- < > NVMEM Support ----
- < > System Trace Module devices
- < > Dummy STM driver
- < > Kernel console over STM devices
- < > Intel(R) Trace Hub controller
- FPGA Configuration Support --->

At the bottom of the window, navigation options are listed: **<Select>**, < Exit >, < Help >, < Save >, and < Load >.

# Sensor HAL

## (Enable IIO in Kernel)



```
emertxe@emertxe-desktop: ~/Android/kernel/rpi
emertxe@emertxe-desktop: ~/Android/kernel/rpi  x  emertxe@emertxe-desktop: /tmp  x  emertxe@emertxe-desktop: ~/Android/kernel/rpi/drive...  x

.config - Linux/arm 4.4.35 Kernel Configuration
> Device Drivers > Industrial I/O support > Humidity sensors

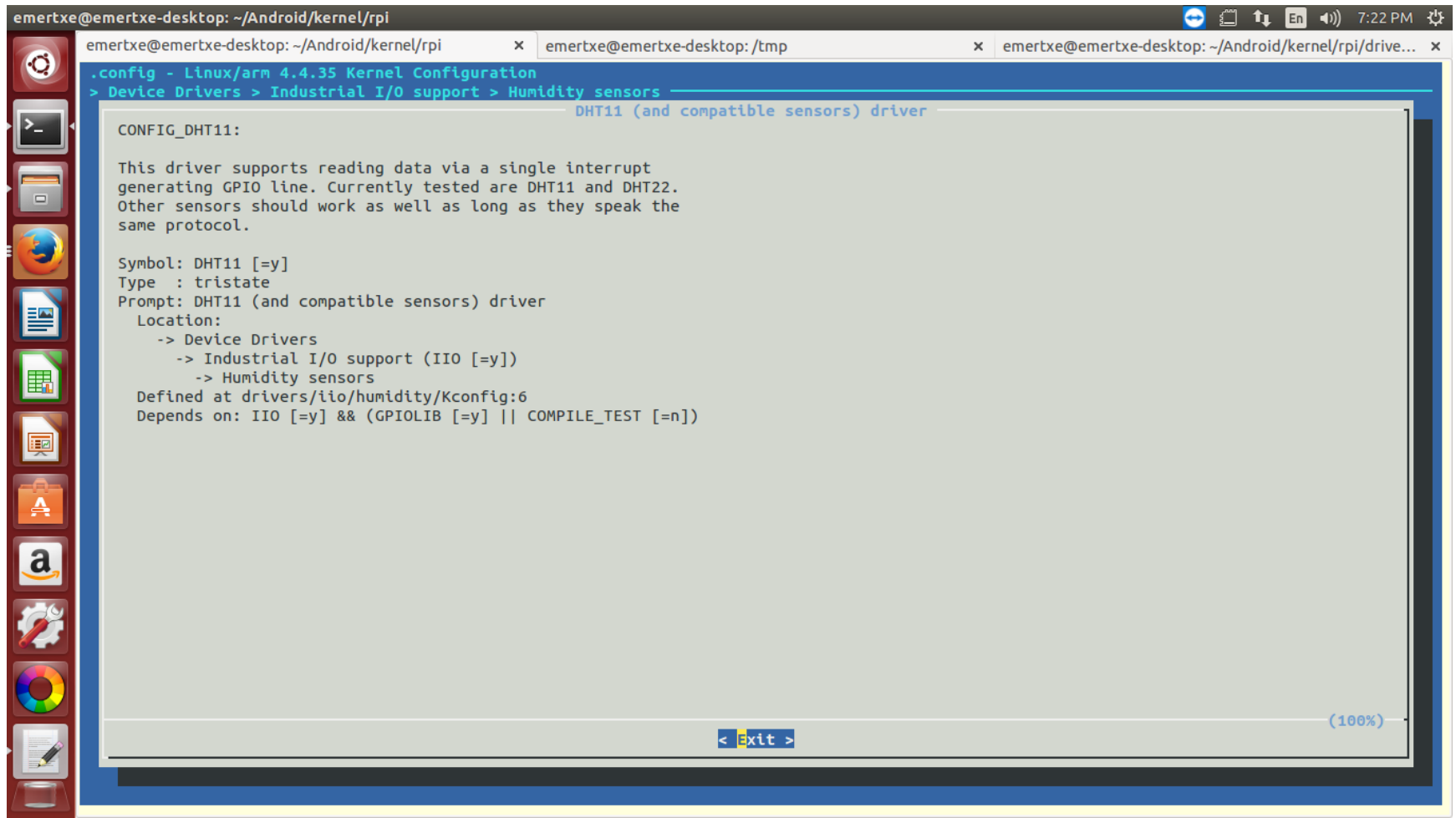
Humidity sensors
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing
<Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in
[ ] excluded <M> module < > module capable

<Y> DHT11 (and compatible sensors) driver
< > TI HDC100x relative humidity and temperature sensor
< > Measurement Specialties HTU21 humidity & temperature sensor
< > SI7005 relative humidity and temperature sensor
< > Si7013/20/21 Relative Humidity and Temperature Sensors

<Select>  < Exit >  < Help >  < Save >  < Load >
```

# Sensor HAL

## (Enable IIO in Kernel)



The screenshot shows a terminal window titled "emertxe@emertxe-desktop: ~/Android/kernel/rpi". The window displays the Linux kernel configuration interface for "Linux/arm 4.4.35 Kernel Configuration". The navigation path is "Device Drivers > Industrial I/O support > Humidity sensors". The current configuration is for the "DHT11 (and compatible sensors) driver".

```
CONFIG_DHT11:
This driver supports reading data via a single interrupt
generating GPIO line. Currently tested are DHT11 and DHT22.
Other sensors should work as well as long as they speak the
same protocol.

Symbol: DHT11 [=y]
Type : tristate
Prompt: DHT11 (and compatible sensors) driver
Location:
-> Device Drivers
-> Industrial I/O support (IIO [=y])
-> Humidity sensors
Defined at drivers/iio/humidity/Kconfig:6
Depends on: IIO [=y] && (GPIOLIB [=y] || COMPILE_TEST [=n])
```

At the bottom right of the configuration window, it says "(100%)" and "< Exit >".

# Sensor HAL

(Boot Configuration)

1. \$ ARCH=arm CROSS\_COMPILE=arm-linux-androideabi- make zImage -j4
  2. \$ ARCH=arm CROSS\_COMPILE=arm-linux-androideabi- make dtbs -j4
- Copy “/arch/arm/boot/dts/overlays/dht11.dtbo” to /boot/overlays folder
  - Add following line in /boot/config.txt
    - ✓ dtoverlay=dht11,gpiopin=4

# Sensor HAL

(Hands-on : Create ramdisk)



- `$ vim /device/brcm/rpi3/ueventd.rpi3.rc`
- Add following line in the file
  - `/dev/iio:device0 0660 system system`
- `$ rm out/target/product/rpi3/ramdisk.img`
- `$ make -j2 bootimage`
- New ramdisk will be generated in out folder

# Sensor HAL

(Industrial IO driver)



- Device path : /dev/iio:device0
- Device File path : /sys/bus/iio/devices/iio:device0
- Temperature input file - in\_temp\_input
- Humidity input file - in\_humidityrelative\_input

# Sensor HAL - Hands-on

(Hands-on : test utility)

- Write Android.mk file for testing IIO DHT11 sensor
- Compile and generate testiio executable and run on target

# Sensor HAL

(Testing stand alone DHT11)

- Use testiio utility program
- Compile and use test-nusensors utility program  
(hardware/libhardware/tests/nusensors/nusensors.cpp)
- Use strace to debug (su mode)

By default system FS is read only (RO). Remount file system as RW to copy test utility in /system/bin

- \$ adb shell
- \$ su
- # mount -o rw, remount /system



# Sensor HAL

(Hands-on : integrating 3<sup>rd</sup> party HAL)

- Integrating open source sensor HAL
  - ✓ Copy files in hardware/rpi/
  - ✓ Write Android.mk
  - ✓ Compile files and copy output in /system/lib/hw
  - ✓ Use test-nusensors to test native library
  - ✓ Use Android Studio to test at app level

# Sensor HAL

(Sensor Module - interface functions)

- Members of struct `sensors_module_t` are as follows

#	Type	Name	Description
1	struct	common	Type of <code>hw_module_t</code>
2	pointer	<code>get_sensor_list</code>	Function pointer
3	pointer	<code>set_operation_mode</code>	Function pointer

**\*See details in `/hardware/libhardware/include/hardware/sensor.h`**

# Sensor HAL

(Sensor Module)

- Interface for sensor that can be polled
- Used with API `SENSOR_DEVICE_API_VERSION_0_1`
- Members of struct `sensors_poll_device_t` are as follows

#	Type	Name	Description
1	struct	common	Type of <code>hw_device_t</code>
2	pointer	activate	Function pointer
3	pointer	setDelay	Function pointer
4	pointer	poll	Function pointer

**\*See details in </hardware/libhardware/include/hardware/sensor.h>**

# Sensor HAL

(Sensor Module)

- Interface for sensor that can be polled
- Used with API `SENSOR_DEVICE_API_VERSION_1_0`
- Members of struct `sensors_poll_device_1_t`

#	Type	Name	Description
1	struct	common	Type of <code>hw_device_t</code>
2	pointer	activate	Function pointer
3	pointer	setDelay	Function pointer
4	pointer	poll	Function pointer
5	pointer	batch	Function pointer
6	pointer	flush	Function pointer

**\*See details in </hardware/libhardware/include/hardware/sensor.h>**

# Sensor HAL

(Sensor Interface functions)



#	function	Description
1	get_sensor_list (list)	Called at boot up to return implemented sensors by HAL
2	activate (sensor, enable)	To activate/deactivate sensor
3	batch (sensor, flags, sampling period, max report latency)	Sets a sensor's parameters, including sampling frequency and maximum report latency.
4	setDelay (sensor, sampling delay)	Deprecated, used by HAL v.01
5	flush (sensor)	Flush hardware FIFO and send flush complete event
6	poll ( )	Returns number of events or error. The function shall never return 0.

# Sensor HAL

## (Sensor Event Mapping)

- Data received from h/w sensor shall be mapped to Android sensor event (`sensors_event_t`)
- Members of `sensors_event_t` are as follows

#	Type	Name	Description
1	Integer	version	Set to sizeof <code>sensors_event_t</code>
2	Integer	sensor	Sensor handle (identifier)
3	Integer	type	Sensor Type
4	Integer	reserved	Reserved for future use
5	Integer	timestamp	Time is nano-seconds
6	Union	Sensor members	Used based on type of sensor
7	Integer	flags	Reserved flags (set to zero)
8	Integer	reserved1[3]	For future use

# Sensor HAL

(Sequence of calls)

- Device boot up : `get_sensors_list` is called
- Sensor activation : batch function will be called with the requested parameters, followed by `activate(..., enable=1)`
  - ✓ In HAL version 0\_1, the order was opposite: `activate` was called first, followed by `set_delay`
- Activated : batch function is called when requested to change characteristics of a sensor
- flush can be called at any time, even on non-activated sensors (in which case it must return `-EINVAL`)
- To deactivate sensor, `activate(..., enable=0)` will be called.
- In parallel to those calls, the poll function will be called repeatedly to request data (poll can be called even when no sensors are activated)

# Sensor HAL

(Wakeup and non-wakeup sensor)



- Wake up sensor will wake up the AP when
  - Their FIFO is full
  - Batch time out expires
- Non-wake up sensors waits for AP to wake up
  - They overwrite the FIFO when full
- Most of the sensors can be implemented as wakeup or non-wake up sensor
- A sensor having both wakeup as well as non-wakeup variant shall maintain separate FIFO for each



# Sensor HAL

(Dynamic sensor)



- A dynamic sensor is runtime pluggable (can be added to and removed from the system at runtime)
- Operations such as batch, activate and setDelay to dynamic sensor shall be treated as no-operation and return successful
- Sensor event of `SENSOR_TYPE_DYNAMIC_SENSOR_META` type shall be delivered, irrespective of activation status, at connection and disconnection of dynamic sensor
- Dynamic sensor shall use unique handle until next boot (if handle 'H' is used by sensor A then same handle can't be assigned to same sensor A or other sensors even after disconnection of A until reboot)
- UUID field is used for identifying sensor in addition to name, vendor, version and type
- UUID shall be unique and persistent for individual sensor unit

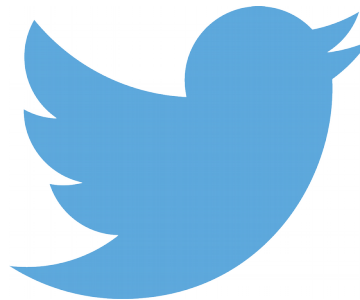
# Stay connected

**About us:** Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies Pvt Ltd  
No. 83, 1st Floor, Farah Towers,  
M.G Road, Bangalore  
Karnataka - 560001  
T: +91 809 555 7 333  
T: +91 80 4128 9576  
E: [training@emertxe.com](mailto:training@emertxe.com)



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



<https://www.slideshare.net/EmertxeSlides>

THANK YOU