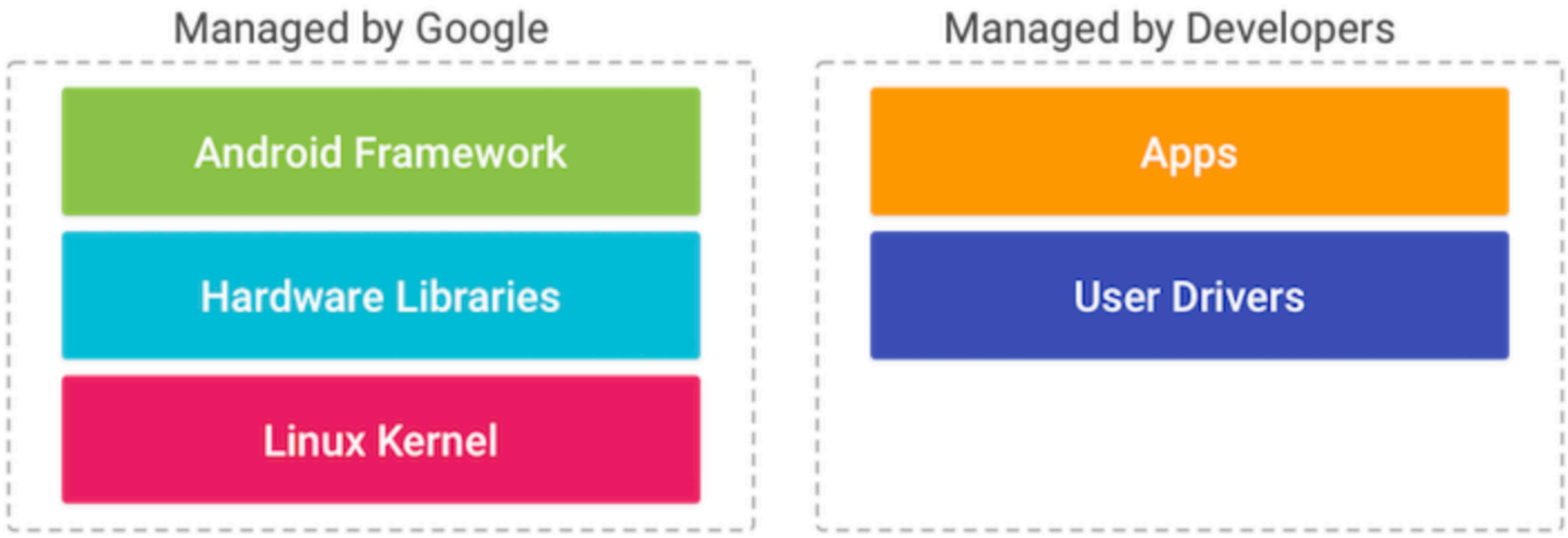
Lecture #6
User-Space Drivers

Android Things 2020

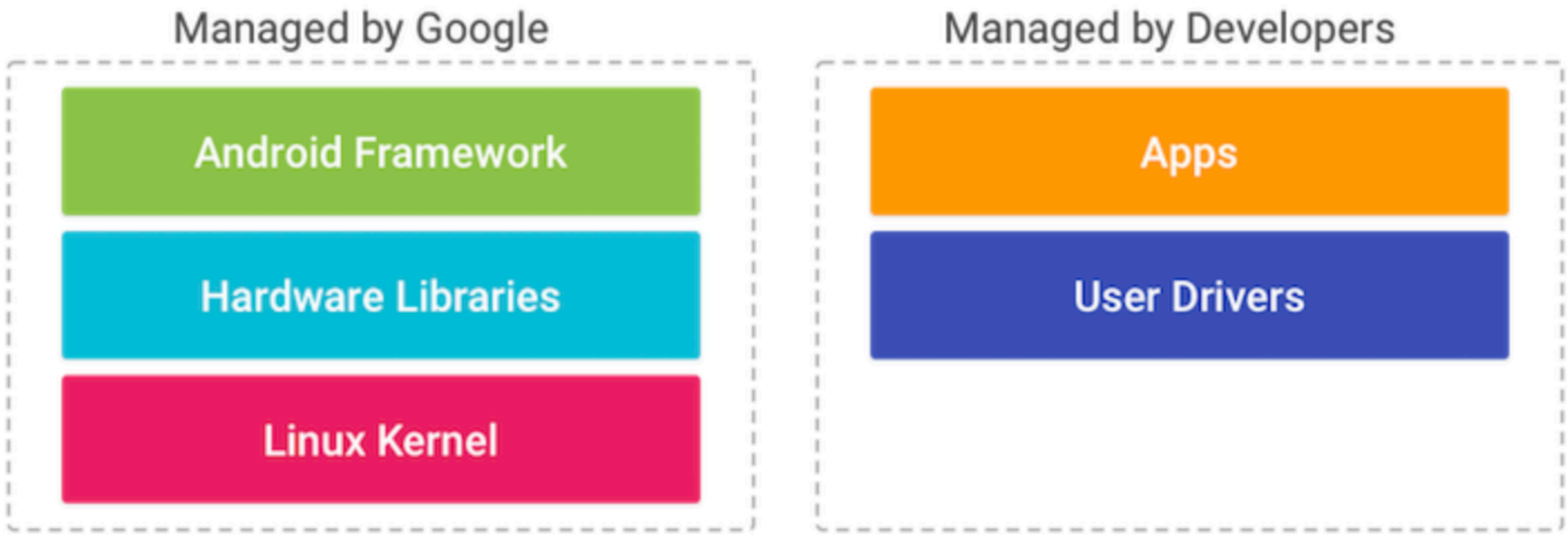
User-Space Drivers

- Components registered from within apps that extend existing Android framework services.



User-Space Drivers

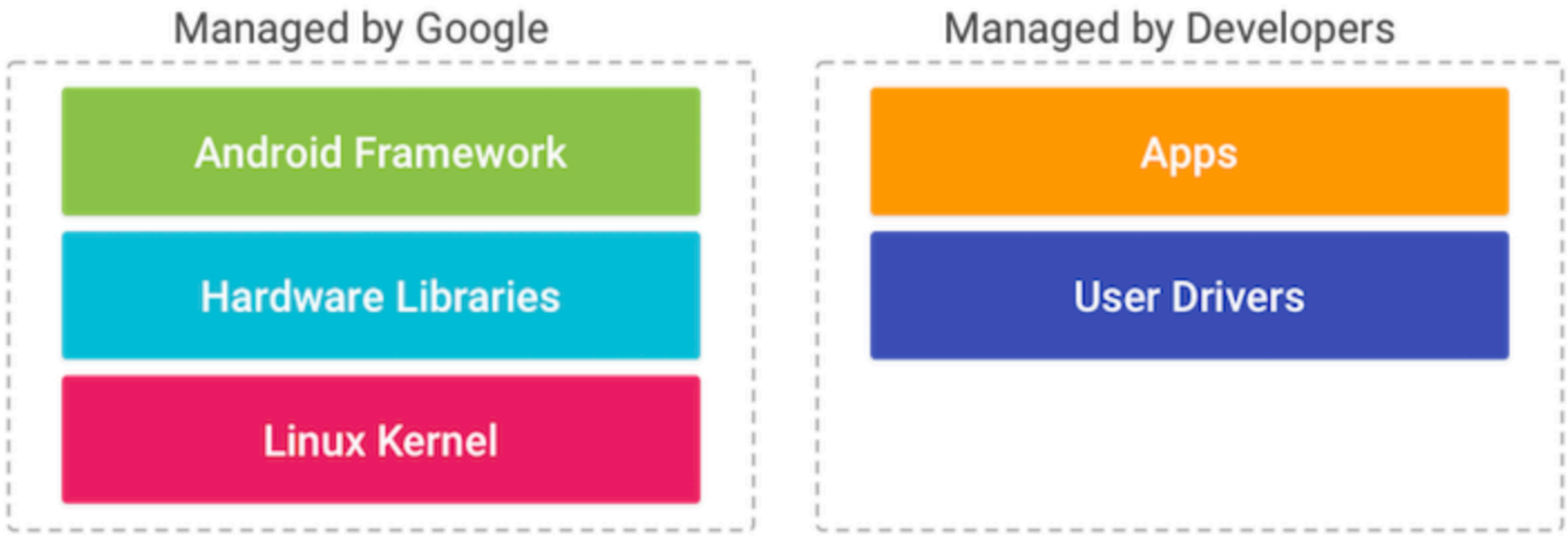
- Components registered from within apps that extend existing Android framework services.



- Inject hardware events into the framework that other apps can process using the standard Android APIs.

User-Space Drivers

- Components registered from within apps that extend existing Android framework services.



- Inject hardware events into the framework that other apps can process using the standard Android APIs.

> ★ **Note:** You cannot customize the behavior of device drivers in the Linux kernel or Hardware Abstraction Layer (HAL) to add new functionality to a device.

Benefits

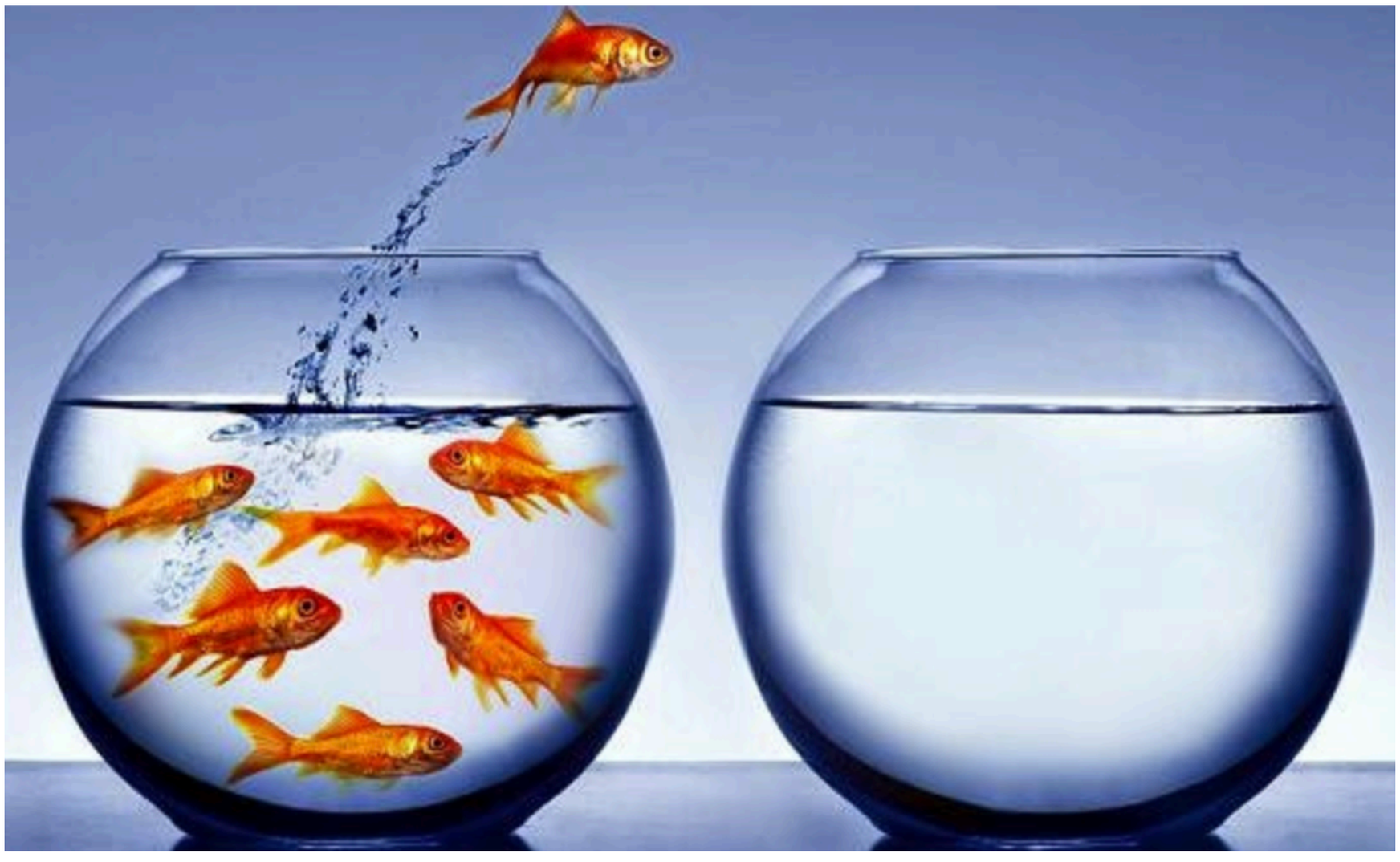


Image Source: https://gyanspot.wordpress.com/2016/05/04/how-to-do-mobile-number-portability/

Benefits

- Portability

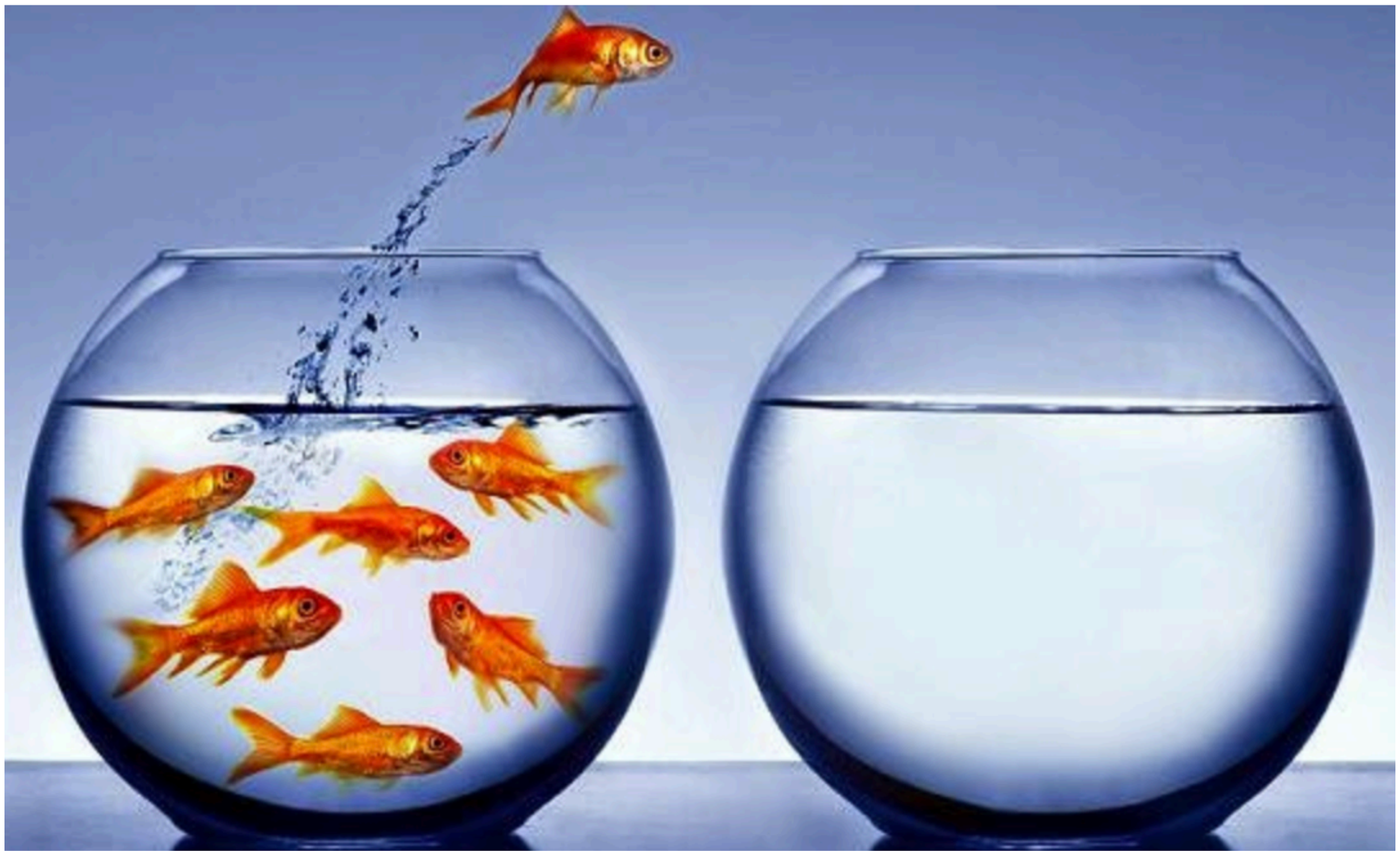


Image Source: https://gyanspot.wordpress.com/2016/05/04/how-to-do-mobile-number-portability/

Benefits

- Portability
- Reuse

Image source: http://www.thecolgatemaroonnews.com

Benefits

- Portability
- Reuse
- Integration

Image Source: https://www.eurodiaconia.org

Device driver types

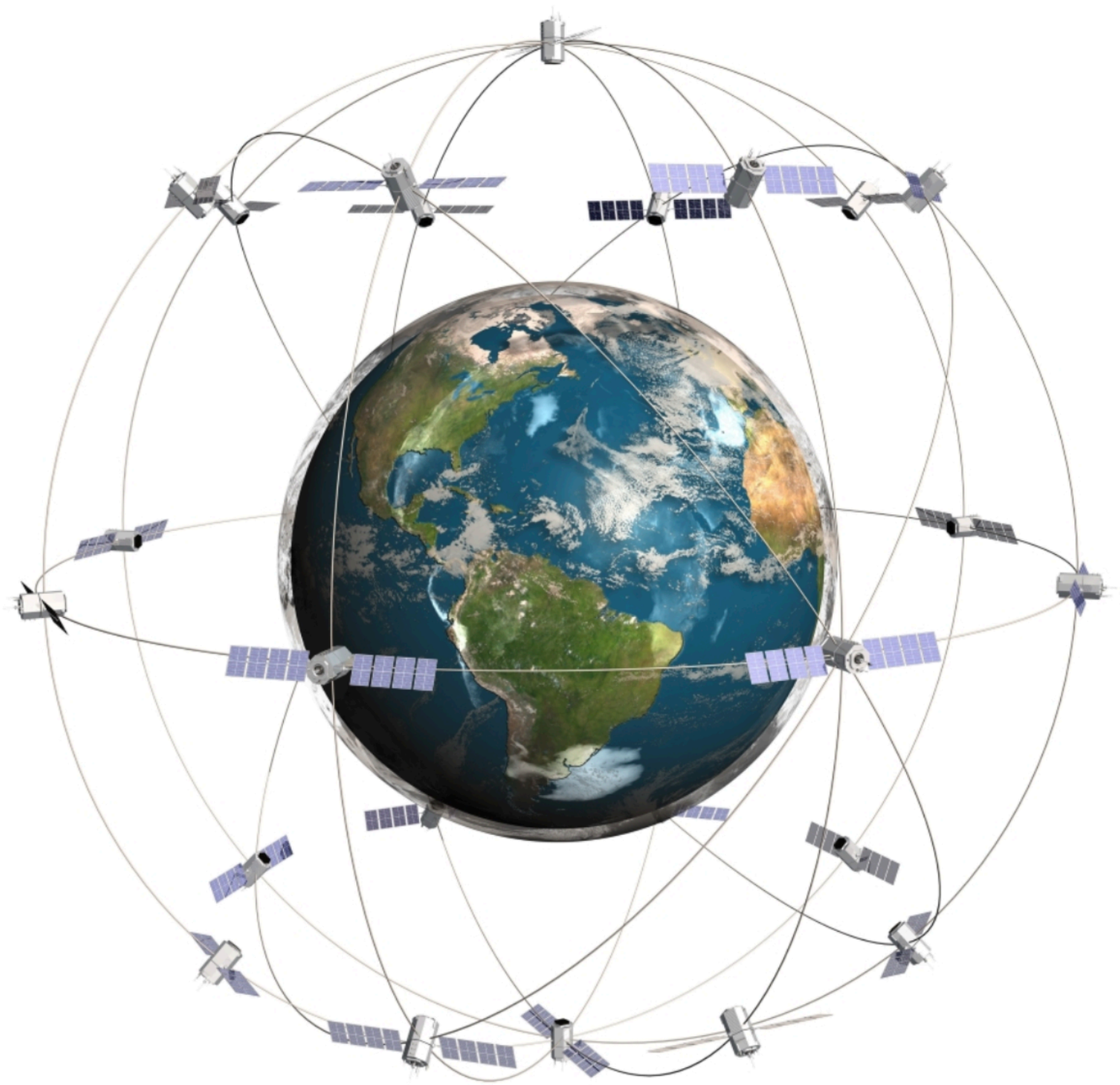


Image source: https://www.geotab.com

Device driver types

- Location

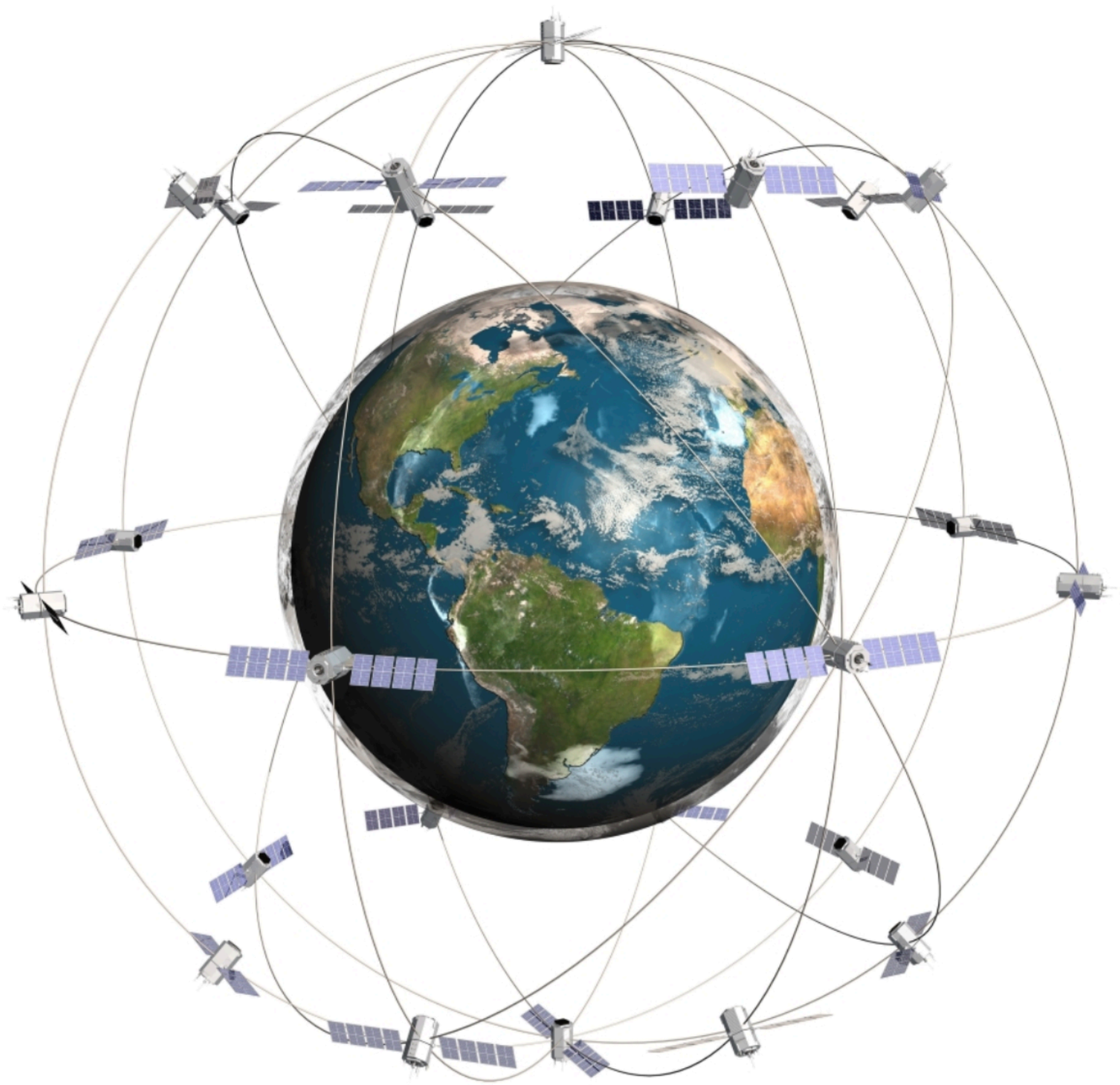


Image source: https://www.geotab.com

Device driver types

- Location
- Human Interface Devices (HID)

Image source: https://www.trendhunter.com

Device driver types

- Location
- Human Interface Devices (HID)

Image source: https://www.maxiaids.com

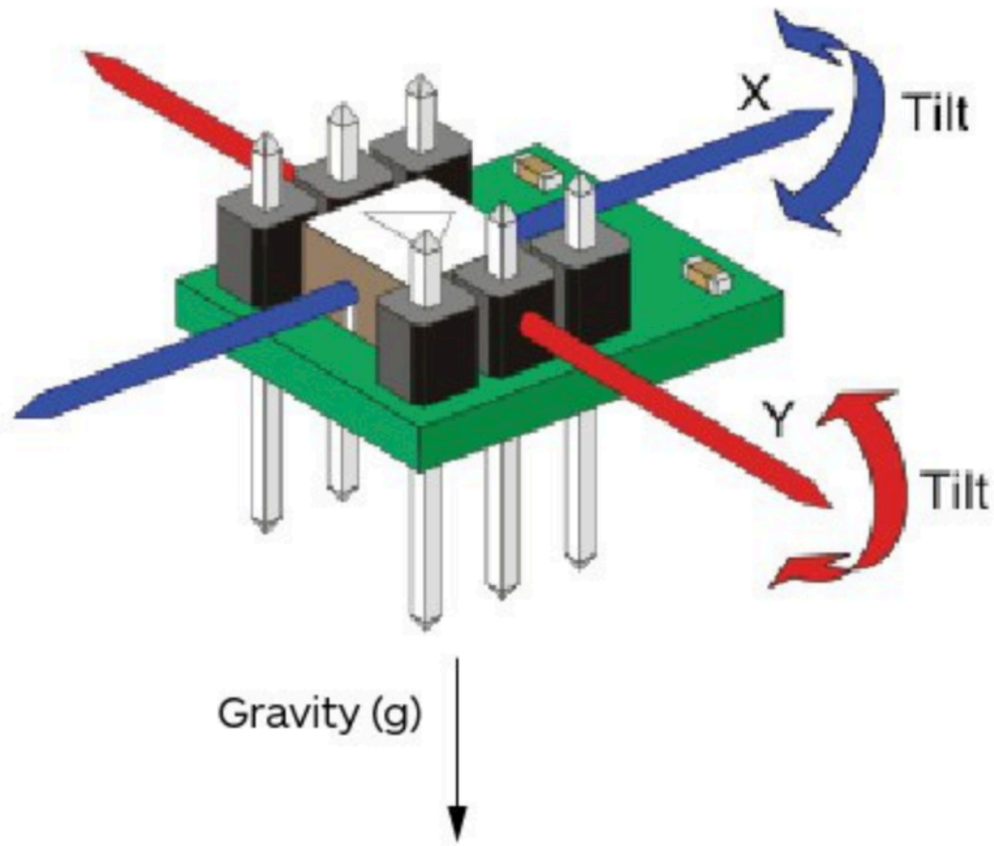
Device driver types

- Location
- Human Interface Devices (HID)

Image source: https://www.amazon.ca

Device driver types

- Location
- Human Interface Devices (H
- Sensors



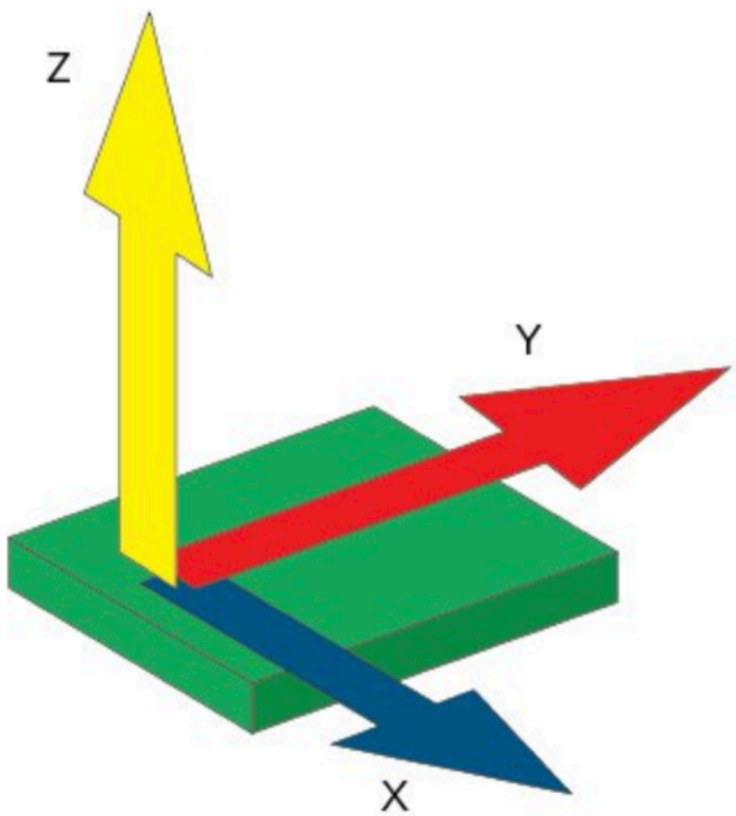


Image source: https://insights.globalspec.com

Device driver types

Device driver types

- Location
- Human Interface Devices (HID)
- Sensors

Image source: https://en.wikipedia.org

Device driver types

- Location
- Human Interface Devices (HID)
- Sensors
- LoWPAN

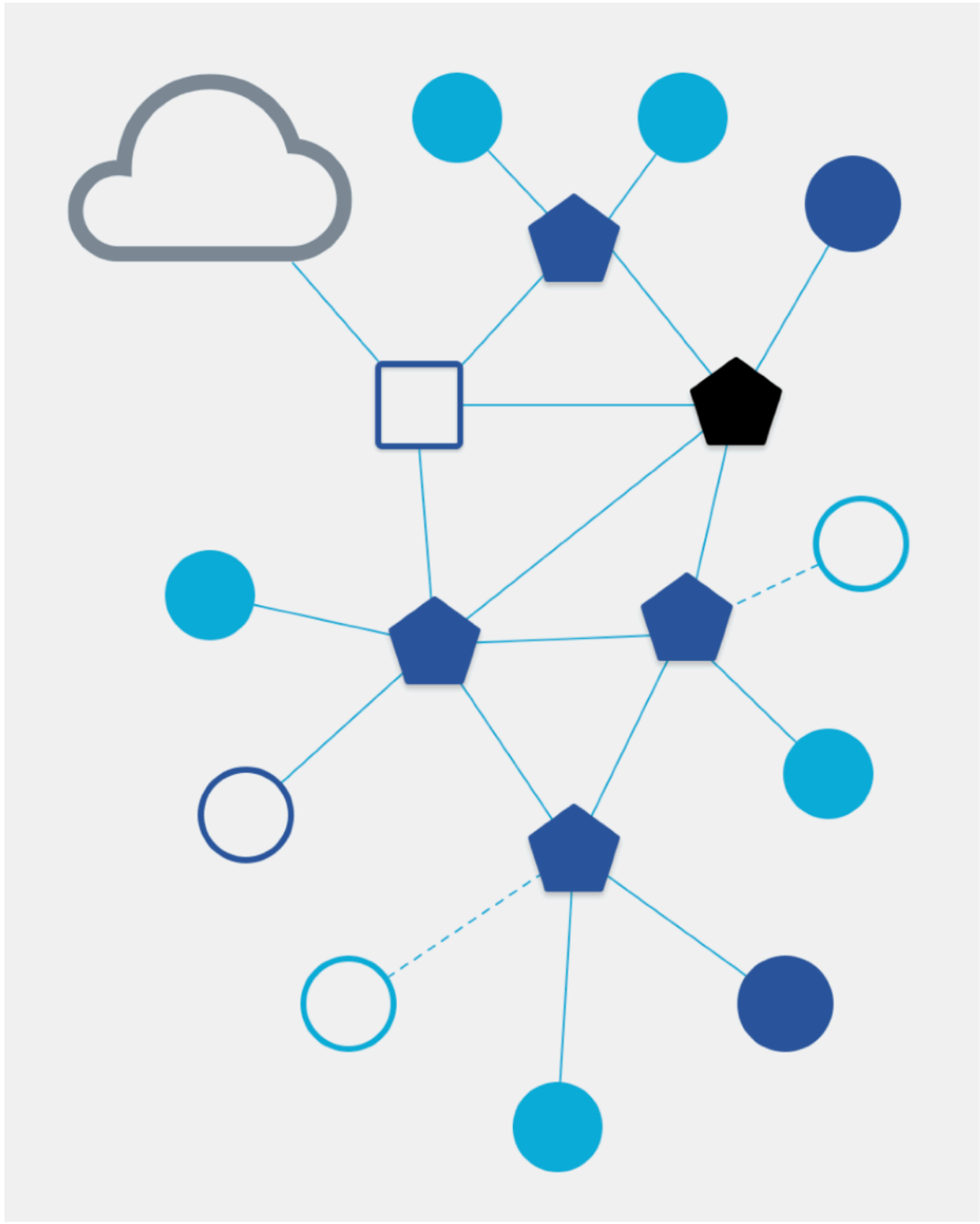


Image source: https://openthread.io/

Global Navigation Satellite System (GNSS)

```
<uses-permission
android:name="com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />
```

Global Navigation Satellite System (GNSS)

```
<uses-permission
android:name="com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />
```

```
import com.google.android.things.userdriver.location.GnssDriver;
import com.google.android.things.userdriver.UserDriverManager;
...
public class LocationDriverService extends Service {
  private GnssDriver mDriver;
  @Override
  public void onCreate() {
    super.onCreate();
    // Create a new driver implementation
    mDriver = new GnssDriver();
    // Register with the framework
    UserDriverManager manager = UserDriverManager.getInstance();
    manager.registerGnssDriver(mDriver);
  }
}
```

Global Navigation Satellite System (GNSS)

```
public class LocationDriverService extends Service {
  ...
  @Override
   protected void onDestroy() {
      super.onDestroy();
      UserDriverManager manager = UserDriverManager.getInstance();
      manager.unregisterGnssDriver();
    }
}
```

GNSS - Location Reporting

```
import android.location.Location;
...
public class LocationDriverService extends Service {
    private GnssDriver mDriver;
    ...
    private Location parseLocationFromString(String data) {
        Location result = new Location(LocationManager.GPS_PROVIDER);
        // ...parse raw GNSS information...
        return result;
    }
    public void handleLocationUpdate(String rawData) {
        // Convert raw data into a location object
        Location location = parseLocationFromString(rawData);
        // Send the location update to the framework
        mDriver.reportLocation(location);
    }
}
```

Location Attributes

Required Attributes	Optional Attributes
Accuracy	Altitude
Timestamp	Bearing
Latitude	Speed
Longitude	

https://developer.android.com/things/sdk/drivers/location.html

Human Interface Devices

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

Human Interface Devices

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
```

```
import com.google.android.things.userdriver.input.InputDriver;
import com.google.android.things.userdriver.UserDriverManager;
...
public class ButtonDriverService extends Service {
// Driver parameters
private static final String DRIVER_NAME = "EscapeButton";
private static final int DRIVER_VERSION = 1;
// Key code for driver to emulate
private static final int KEY_CODE = KeyEvent.KEYCODE_ESCAPE;
private InputDriver mDriver;
@Override
public void onCreate() {
    super.onCreate();

        // Create a new driver instance
        mDriver = new InputDriver.Builder()
                .setName(DRIVER_NAME)
                .setSupportedKeys(new int[] {KEY_CODE})
                .build();
```


```
// Driver parameters
private static final String DRIVER_NAME = "EscapeButton";
private static final int DRIVER_VERSION = 1;
// Key code for driver to emulate
private static final int KEY_CODE = KeyEvent.KEYCODE_ESCAPE;
private InputDriver mDriver;
@Override
public void onCreate() {
    super.onCreate();

        // Create a new driver instance
        mDriver = new InputDriver.Builder()
                .setName(DRIVER_NAME)
                .setSupportedKeys(new int[] {KEY_CODE})
                .build();

        // Register with the framework
        UserDriverManager manager = UserDriverManager.getInstance();
        manager.registerInputDriver(mDriver);
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

https://developer.android.com/things/sdk/drivers/input.html

Sensors

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_SENSOR_DRIVERS" />
```

Sensors

```
<uses-permission android:name="com.google.android.things.permission.MANAGE_SENSOR_DRIVERS" />
```

```
UserSensorDriver driver = new UserSensorDriver() {
    // Sensor data values
    float x, y, z;

    @Override
    public UserSensorReading read() {
        try {
            // ...read the sensor hardware...

            // Return a new reading
            return new UserSensorReading(new float[]{x, y, z});
        } catch (Exception e) {
            // Error occurred reading the sensor hardware
            throw new IOException("Unable to read sensor");
        }
    }
};
```

Low Power Sensor

```
UserSensorDriver driver = new UserSensorDriver() {
    ...

    // Called by the framework to toggle low power modes
    @Override
    public void setEnabled(boolean enabled) {
        if (enabled) {
            // Exit low power mode
        } else {
            // Enter low power mode
        }
    }
};
```

Describing the Sensor

```
UserSensor accelerometer = UserSensor.builder()
        .setName("KoolAccelerometerSensor")
        .setVendor(“UBBFactory")
        .setType(Sensor.TYPE_ACCELEROMETER)
        .setDriver(driver)
        .build();
```

Registering the Sensor

```
import com.google.android.things.userdriver.UserDriverManager;
...
public class SensorDriverService extends Service {
    UserSensor accelerometer;
    @Override
    public void onCreate() {
        super.onCreate();
        ...
        UserDriverManager manager = UserDriverManager.getInstance();
        // Create a new driver implementation
        accelerometer = ...;
        // Register the new driver with the framework
        manager.registerSensor(accelerometer);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        ...
        UserDriverManager manager = UserDriverManager.getInstance();
        // Unregister the driver when finished
```

```
import com.google.android.things.userdriver.UserDriverManager;
...
public class SensorDriverService extends Service {
    UserSensor accelerometer;
    @Override
    public void onCreate() {
        super.onCreate();
        ...
        UserDriverManager manager = UserDriverManager.getInstance();
        // Create a new driver implementation
        accelerometer = ...;
        // Register the new driver with the framework
        manager.registerSensor(accelerometer);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        ...
        UserDriverManager manager = UserDriverManager.getInstance();
        // Unregister the driver when finished
        manager.unregisterSensor(accelerometer);
    }
}
```

Access the Sensors

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private SensorCallback callback = new SensorCallback();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sensorManager.registerDynamicSensorCallback(callback);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        ...
        sensorManager.unregisterDynamicSensorCallback(callback);
    }
    @Override
    public void onSensorChanged(SensorEvent event) {
        ...
    }
```

```
    sensorManager.unregisterDynamicSensorCallback(callback);
}
@Override
public void onSensorChanged(SensorEvent event) {
    ...
}
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    ...
}
// Listen for registration events from the sensor driver
private class SensorCallback extends SensorManager.DynamicSensorCallback {
    @Override
    public void onDynamicSensorConnected(Sensor sensor) {
        // Begin listening for sensor readings
        sensorManager.registerListener(SensorActivity.this, sensor,
                SensorManager.SENSOR_DELAY_NORMAL);
    }
    @Override
    public void onDynamicSensorDisconnected(Sensor sensor) {
        // Stop receiving sensor readings
        sensorManager.unregisterListener(SensorActivity.this);
    }
  }
}
```

https://developer.android.com/things/sdk/drivers/sensors

LoWPAN

arm

dialog SEMICONDUCTOR

Microsoft

nest

NORDIC SEMICONDUCTOR

NXP

Particle

qorvo

QUALCOMM TECHNOLOGIES, INC.

STMicroelectronics

SYNOPSYS®

Texas Instruments

Zephyr™ Project

1. Introduction

OpenThread is an open-source implementation of the Thread networking protocol. Nest has released OpenThread to make the technology used in Nest products broadly available to developers to accelerate the development of products for the connected home.

The Thread specification defines an IPv6-based reliable, secure and low-power wireless device-to-device communication protocol for home applications. OpenThread implements all Thread networking layers including IPv6, 6LoWPAN, IEEE 802.15.4 with MAC security, Mesh Link Establishment, and Mesh Routing.

This Codelab will walk you through simulating a Thread network on emulated devices using Docker.

What you'll learn

- ✓ How to set up the OpenThread build toolchain
- ✓ How to simulate a Thread network
- ✓ How to authenticate Thread nodes
- ✓ How to manage a Thread network with `wpantund`

What you'll need

- Docker
- Basic knowledge of Linux, network routing

https://codelabs.developers.google.com/codelabs/openthread-simulation/#0

Create a Driver

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 27
    buildToolsVersion '28.0.3'

    defaultConfig {
        minSdkVersion 27
        targetSdkVersion 27
        // Other default stuff...
    }
}

dependencies {
    provided 'com.google.android.things:androidthings:0.1-devpreview'
}
```

Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mydriver">
    <application>
        <uses-library android:name="com.google.android.things" />
    </application>
</manifest>
```

Driver

```
class LEDSensor : AutoCloseable {
    private lateinit var redPin: Gpio
    private lateinit var greenPin: Gpio
    private lateinit var bluePin: Gpio

    fun init(redGPIOName: String = "BCM17", greenGPIOName: String = "BCM27", blueGPIOName: String = "BCM22") {
        val pioManager = PeripheralManager.getInstance()
        redPin = pioManager.openGpio(redGPIOName)
        greenPin = pioManager.openGpio(greenGPIOName)
        bluePin = pioManager.openGpio(blueGPIOName)
        initGPIO(redPin)
        initGPIO(greenPin)
        initGPIO(bluePin)
    }

    fun write(red: Boolean = false, green: Boolean = false, blue: Boolean = false) {
        redPin.value = red
        greenPin.value = green
        bluePin.value = blue
    }
```


```
    redPin = pioManager.openGpio(redGPIOName)
    greenPin = pioManager.openGpio(greenGPIOName)
    bluePin = pioManager.openGpio(blueGPIOName)
    initGPIO(redPin)
    initGPIO(greenPin)
    initGPIO(bluePin)
}

fun write(red: Boolean = false, green: Boolean = false, blue: Boolean = false) {
    redPin.value = red
    greenPin.value = green
    bluePin.value = blue
    }

    private fun initGPIO(pin: Gpio) {
        pin.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW)
        pin.setActiveType(Gpio.ACTIVE_HIGH)
    }

    override fun close() {
        redPin.close()
        greenPin.close()
        bluePin.close()
    }
}
```

Lecture outcomes

- How to create a user driver.
- How to use the existing user-defined drivers.