Using Hardware Devices

When building a mobile application, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device.

You can use any Android-powered device as an environment for running, debugging, and testing your applications. The tools included in the SDK make it easy to install and run your application on the device each time you compile. You can install your application on the device directly from Eclipse or from the command line with ADB. If you don't yet have a device, check with the service providers in your area to determine which Android-powered devices are available.

If you want a SIM-unlocked phone, then you might consider the Google Nexus S. To find a place to purchase the Nexus S and other Android-powered devices, visit google.com/phone.

> **Note:** When developing on a device, keep in mind that you should still use the Android emulator to test your application on configurations that are not equivalent to those of your real device. Although the emulator does not allow you to test every device feature (such as the accelerometer), it does allow you to verify that your application functions properly on different versions of the Android platform, in different screen sizes and orientations, and more.

Setting up a Device for Development

With an Android-powered device, you can develop and debug your Android applications just as you would on the emulator. Before you can start, there are just a few things to do:

1. Declare your application as "debuggable" in your Android Manifest.

   When using Eclipse, you can skip this step, because running your app directly from the Eclipse IDE automatically enables debugging.

   In the `AndroidManifest.xml` file, add `android:debuggable="true"` to the `<application>` element.

   > **Note:** If you manually enable debugging in the manifest file, be sure to disable it before you build for release (your published application should usually *not* be debuggable).

2. Turn on "USB Debugging" on your device.

   On the device, go to **Settings > Applications > Development** and enable **USB debugging** (on an Android 4.0 device, the setting is located in **Settings > Developer options**).

3. Set up your system to detect your device.
   - If you're developing on Windows, you need to install a USB driver for adb. For an installation guide and links to OEM drivers, see the OEM USB Drivers document.
   - If you're developing on Mac OS X, it just works. Skip this step.
   - If you're developing on Ubuntu Linux, you need to add a `udev` rules file that contains a USB configuration for each type of device you want to use for development. In the rules file, each device manufacturer is identified by a unique vendor ID, as specified by the `ATTR{idVendor}` property. For a list of vendor IDs, see USB Vendor IDs, below. To set up device detection on Ubuntu Linux:
     a. Log in as root and create this file: `/etc/udev/rules.d/51-android.rules`.

        Use this format to add each vendor to the file:
        ```
        SUBSYSTEM=="usb", ATTR{idVendor}=="0bb4", MODE="0666", GROUP="plugdev"
        ```

        In this example, the vendor ID is for HTC. The `MODE` assignment specifies read/write permissions, and `GROUP` defines which Unix group owns the device node.

        > **Note:** The rule syntax may vary slightly depending on your environment. Consult the `udev` documentation for your system as needed. For an overview of rule syntax, see this guide to writing udev rules.

     b. Now execute:
        ```
        chmod a+r /etc/udev/rules.d/51-android.rules
        ```

When plugged in over USB, can verify that your device is connected by executing `adb devices` from your SDK `platform-tools/` directory. If connected, you'll see the device name listed as a "device."

If using Eclipse, run or debug your application as usual. You will be presented with a **Device Chooser** dialog that lists the available emulator(s) and connected device(s). Select the device upon which you want to install and run the application.

If using the Android Debug Bridge (adb), you can issue commands with the `-d` flag to target your connected device.

USB Vendor IDs

This table provides a reference to the vendor IDs needed in order to add USB device support on Linux. The USB Vendor ID is the value given to the `ATTR{idVendor}` property in the rules file, as described above.

Company	USB Vendor ID
Acer	`0502`
ASUS	`0b05`
Dell	`413c`
Foxconn	`0489`
Fujitsu	`04c5`
Fujitsu Toshiba	`04c5`
Garmin-Asus	`091e`
Google	`18d1`
Hisense	`109b`
HTC	`0bb4`
Huawei	`12d1`
K-Touch	`24e3`
KT Tech	`2116`

Kyocera	`0482`
Lenovo	`17ef`
LG	`1004`
Motorola	`22b8`
NEC	`0409`
Nook	`2080`
Nvidia	`0955`
OTGV	`2257`
Pantech	`10a9`
Pegatron	`1d4d`
Philips	`0471`
PMC-Sierra	`04da`
Qualcomm	`05c6`
SK Telesys	`1f53`
Samsung	`04e8`
Sharp	`04dd`
Sony	`054c`
Sony Ericsson	`0fce`
Teleepoch	`2340`
Toshiba	`0930`
ZTE	`19d2`

+1 125