# [https://source.android.google.cn](https://source.android.google.cn/)

# https://developer.android.google.cn/ndk

# Adding a New Device

Use the information in this page to create the Makefiles for your device and product. Please note, unlike the other pages in this section, the contents here are applicable only when creating an entirely new device type and are intended for company build and product teams only.

## Understand Build Layers

The build hierarchy includes the abstraction layers that correspond to the physical makeup of a device. These layers are described in the table below. Each layer relates to the one above it in a one-to-many relationship. For example, an architecture can have more than one board and each board can have more than one product. You may define an element in a given layer as a specialization of an element in the same layer, thus eliminating copying and simplifying maintenance.

|  |  |  |
| --- | --- | --- |
| **Layer** | **Example** | **Description** |
| Product | myProduct, myProduct\_eu, myProduct\_eu\_fr, j2, sdk | The product layer defines the feature specification of a shipping product such as the modules to build, locales supported, and the configuration for various locales. In other words, this is the name of the overall product. Product-specific variables are defined in product definition Makefiles. A product can inherit from other product definitions, which simplifies maintenance. A common method is to create a base product that contains features that apply for all products, then creating product variants based on that base product. For example, you can have two products that differ only by their radios (CDMA vs GSM) inherit from the same base product that does not define a radio. |
| Board/Device | sardine, trout, goldfish | The device/board layer represents the physical layer of plastic on the device (i.e. the industrial design of the device). For example, North American devices probably include QWERTY keyboards whereas devices sold in France probably include AZERTY keyboards. This layer also represents the bare schematics of a product. These include the peripherals on the board and their configuration. The names used are merely codes for different board/device configurations. |
| Arch | arm, x86, mips, arm64, x86\_64, mips64 | The architecture layer describes the processor configuration and ABI (Application Binary Interface) running on the board. |

## Use Build Variants

When building for a particular product, it's often useful to have minor variations on what is ultimately the final release build. In a module definition, the module can specify tags with **LOCAL\_MODULE\_TAGS**, which can be one or more values of **optional** (default), **debug**, **eng**.

If a module doesn't specify a tag (by **LOCAL\_MODULE\_TAGS**), its tag defaults to **optional**. An optional module is installed only if it is required by product configuration with **PRODUCT\_PACKAGES**.

These are the currently-defined build variants:

|  |  |
| --- | --- |
| **eng** | This is the default flavor.   * Installs modules tagged with: **eng** and/or **debug**. * Installs modules according to the product definition files, in addition to tagged modules. * **ro.secure=0** * **ro.debuggable=1** * **ro.kernel.android.checkjni=1** * **adb** is enabled by default. |
| **user** | This is the flavor intended to be the final release bits.   * Installs modules tagged with **user**. * Installs modules according to the product definition files, in addition to tagged modules. * **ro.secure=1** * **ro.debuggable=0** * **adb** is disabled by default. |
| **userdebug** | The same as **user**, except:   * Also installs modules tagged with **debug**. * **ro.debuggable=1** * **adb** is enabled by default. |

## Customize the Build with Resource Overlays

The Android build system uses resource overlays to customize a product at build time. Resource overlays specify resource files that are applied on top of the defaults. To use resource overlays, modify the project buildfile to set **PRODUCT\_PACKAGE\_OVERLAYS** to a path relative to your top-level directory. That path becomes a shadow root searched along with the current root when the build system searches for resources.

The most commonly customized settings are contained in the file [frameworks/base/core/res/res/config.xml](https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/values/config.xml).

To set up a resource overlay on this file, add the overlay directory to the project buildfile, as follows:

PRODUCT\_PACKAGE\_OVERLAYS := device/DEVICE\_IMPLEMENTER/DEVICE\_NAME/overlay

or

PRODUCT\_PACKAGE\_OVERLAYS := vendor/VENDOR\_NAME/overlay

Then, add an overlay file to the directory, for example:

vendor/foobar/overlay/frameworks/base/core/res/res/config.xml

Any strings or string arrays found in the overlay **config.xml** file replace those found in the original file.

## Build a Product

There are many ways to organize the source files for your device. We'll briefly go over how the Nexus 6 implementation was organized as an example, but you can organize your source files and build the way you see fit.

Nexus 6 was implemented with a main device configuration named **shamu**. From this device configuration, a product is created with a product definition Makefile that declares product-specific information about the device such as the name and model. You can view the **device/moto/shamu** directory to see how all of this is setup.

### Write the Makefiles

The following steps describe how to set up product Makefiles in a way similar to that of the Nexus 6 product line:

1. Create a **device/<company\_name>/<device\_name>** directory for your product. For example, **device/moto/shamu**. This directory will contain source code for your device along with the Makefiles to build them.
2. Create a **device.mk** Makefile that declares the files and modules needed for the device. For an example, see **device/moto/shamu/device.mk**.
3. Create a product definition Makefile to create a specific product based on the device. The following Makefile is taken from **device/moto/shamu/aosp\_shamu.mk** as an example. Notice the product is inheriting from the**device/moto/shamu/device.mk** and **vendor/moto/shamu/device-vendor.mk** files via the Makefile while also declaring the product-specific information such as name, brand, and model.

# Inherit from the common Open Source product configuration

$(call inherit-product, $(SRC\_TARGET\_DIR)/product/aosp\_base\_telephony.mk)

PRODUCT\_NAME := aosp\_shamu

PRODUCT\_DEVICE := shamu

PRODUCT\_BRAND := Android

PRODUCT\_MODEL := AOSP on Shamu

PRODUCT\_MANUFACTURER := motorola

PRODUCT\_RESTRICT\_VENDOR\_FILES := true

$(call inherit-product, device/moto/shamu/device.mk)

$(call inherit-product-if-exists, vendor/moto/shamu/device-vendor.mk)

PRODUCT\_NAME := aosp\_shamu

PRODUCT\_PACKAGES += \

Launcher3

See [Product Definition Variables](https://source.android.google.cn/setup/add-device" \l "prod-def) for additional product-specific variables you can add to your Makefiles.

1. Create an **AndroidProducts.mk** file that points to the product's Makefiles. In this example, only the product definition Makefile is needed. The example below is from **device/moto/shamu/AndroidProducts.mk**:

#

# This file should set PRODUCT\_MAKEFILES to a list of product makefiles

# to expose to the build system. LOCAL\_DIR will already be set to

# the directory containing this file.

#

# This file may not rely on the value of any variable other than

# LOCAL\_DIR; do not use any conditionals, and do not look up the

# value of any variable that isn't set in this file or in a file that

# it includes.

#

PRODUCT\_MAKEFILES := \

$(LOCAL\_DIR)/aosp\_shamu.mk

1. Create a **BoardConfig.mk** Makefile that contains board-specific configurations. For an example, see **device/moto/shamu/BoardConfig.mk**.
2. Create a **vendorsetup.sh** file to add your product (a "lunch combo") to the build along with a [build variant](https://source.android.google.cn/setup/add-device" \l "build-variants)separated by a dash. For example:

add\_lunch\_combo <PRODUCT\_NAME>-userdebug

1. At this point, you can create more product variants based on the same device.

### Set Product Definition Variables

Product-specific variables are defined in the product's Makefile. Variables maintained in a product definition files include:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Description** | **Example** |
| PRODUCT\_AAPT\_CONFIG | **aapt** configurations to use when creating packages |  |
| PRODUCT\_BRAND | The brand (e.g., carrier) the software is customized for, if any |  |
| PRODUCT\_CHARACTERISTICS | **aapt** characteristics to allow adding variant-specific resources to a package. | tablet,nosdcard |
| PRODUCT\_COPY\_FILES | List of words like **source\_path:destination\_path**. The file at the source path should be copied to the destination path when building this product. The rules for the copy steps are defined in config/Makefile |  |
| PRODUCT\_DEVICE | Name of the industrial design. This is also the board name, and the build system uses it to locate the **BoardConfig.** | **tuna** |
| PRODUCT\_LOCALES | A space-separated list of two-letter language code, two-letter country code pairs that describe several settings for the user, such as the UI language and time, date and currency formatting. The first locale listed in PRODUCT\_LOCALES is used as the product's default locale. | **en\_GB de\_DE es\_ES fr\_CA** |
| PRODUCT\_MANUFACTURER | Name of the manufacturer | **acme** |
| PRODUCT\_MODEL | End-user-visible name for the end product |  |
| PRODUCT\_NAME | End-user-visible name for the overall product. Appears in the Settings > About screen. |  |
| PRODUCT\_OTA\_PUBLIC\_KEYS | List of Over the Air (OTA) public keys for the product |  |
| PRODUCT\_PACKAGES | Lists the APKs and modules to install. | **Calendar Contacts** |
| PRODUCT\_PACKAGE\_OVERLAYS | Indicate whether to use default resources or add any product specific overlays | **vendor/acme/overlay** |
| PRODUCT\_PROPERTY\_OVERRIDES | List of system property assignments in the format "key=value" |  |

### Set ANDROID\_VENDOR\_KEYS to connect over USB

The **ANDROID\_VENDOR\_KEYS** environment variable enables device manufacturers to access production builds over **adb**. Generate a key for each release that every device will accept, store those internally (such as at **vendor/oem-name/security/adb/**), and then use **ANDROID\_VENDOR\_KEYS** to tell **adb** to use these canonical keys rather than random keys.

Use the **ANDROID\_VENDOR\_KEYS** environment variable to point to the directory containing the generated **adb** public and private keys used for encryption. The private key is stored in file. The public key is stored in file.pub. The **ANDROID\_VENDOR\_KEYS** environment variable points to a file or directory where the generated key pairs are stored.

This variable is set to a file or directory that contains 2048-bit RSA authentication key pairs generated with the **adb keygen** file command. These key pairs are in addition to the RSA key pairs generated by the ADB server. An RSA key pair is needed when you use **adb** to connect over USB for the first time.

You must accept the host computer's RSA key to explicitly grant **adb** access to the device. By default key pairs generated by the ADB server are stored in the following key store directories as **adbkey** (private key) and **adbkey.pub**(public key):

For file locations, on MacOS, this will likely be: **$HOME/.android**. On Windows and Linux, this will be:**%USERPOFILE%\.android**. On Windows, RSA authentication keys can also be in **C:\Windows\System32\config\systemprofile\.android** in some cases. When the ADB server needs a key, it first searches the ADB server key store directory. If no keys are found, it then checks the **ANDROID\_VENDOR\_KEYS**environment variable. If no keys are found, the local ADB server generates and saves a new key pair in the ADB server key store directory.

**Note:** You can override the default directory where the ADB server stores RSA keys by setting the **ANDROID\_SDK\_HOME**environment variable. On the device, keys are stored in the **/data/misc/adb/adb\_keys/** file, and new authorized keys are appended to the same file as you accept them.