Android build system

Nguyen Tran (Nguyen.TranLeHoang@vn.bosch.com)

Jan 06, 2023

Day 5

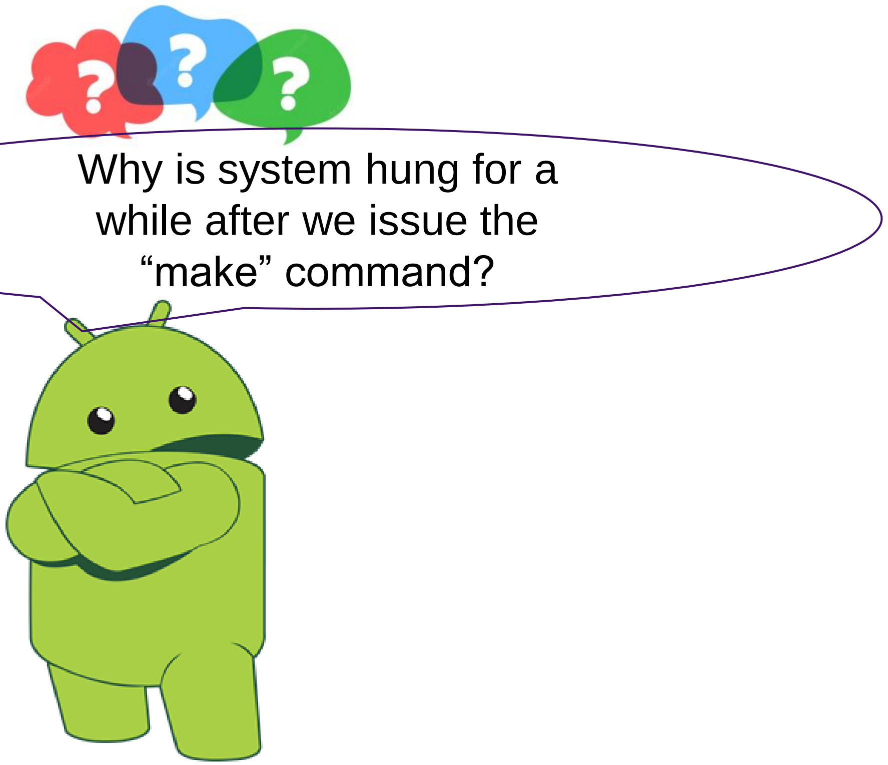
(Android.mk)

QUIZ (1/12)

What is the entry point after we execute “make” command?

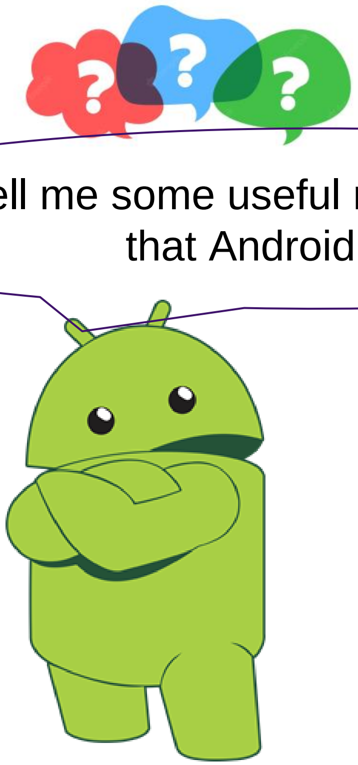
BOSCH

QUIZ (2/12)


?
?
?
Why is system hung for a while after we issue the “make” command?

BOSCH

QUIZ (3/12)

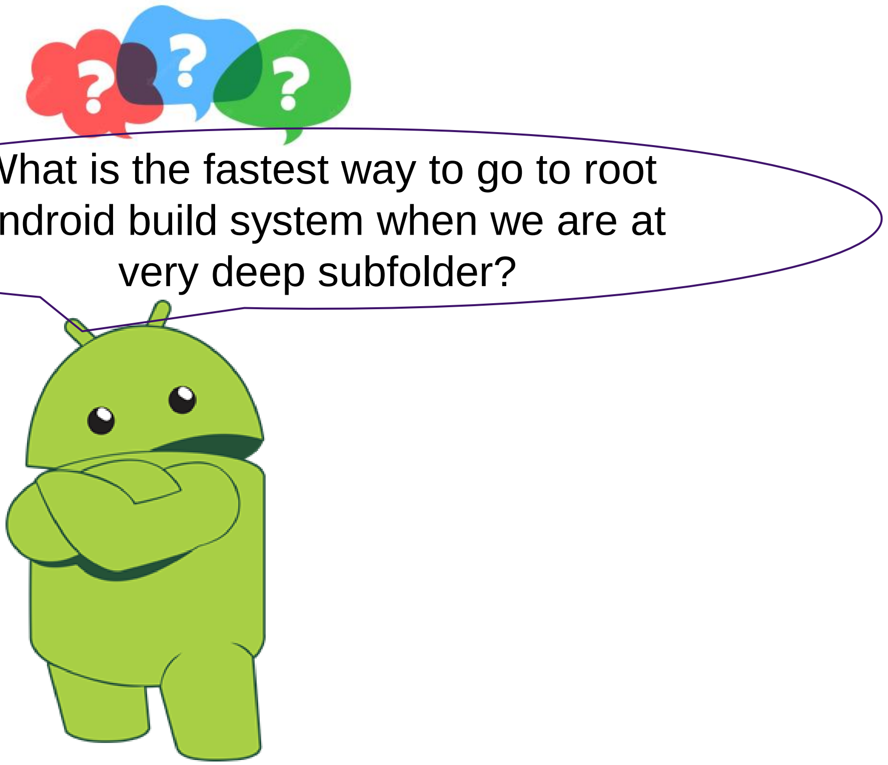


Tell me some useful macros/functions that Android support

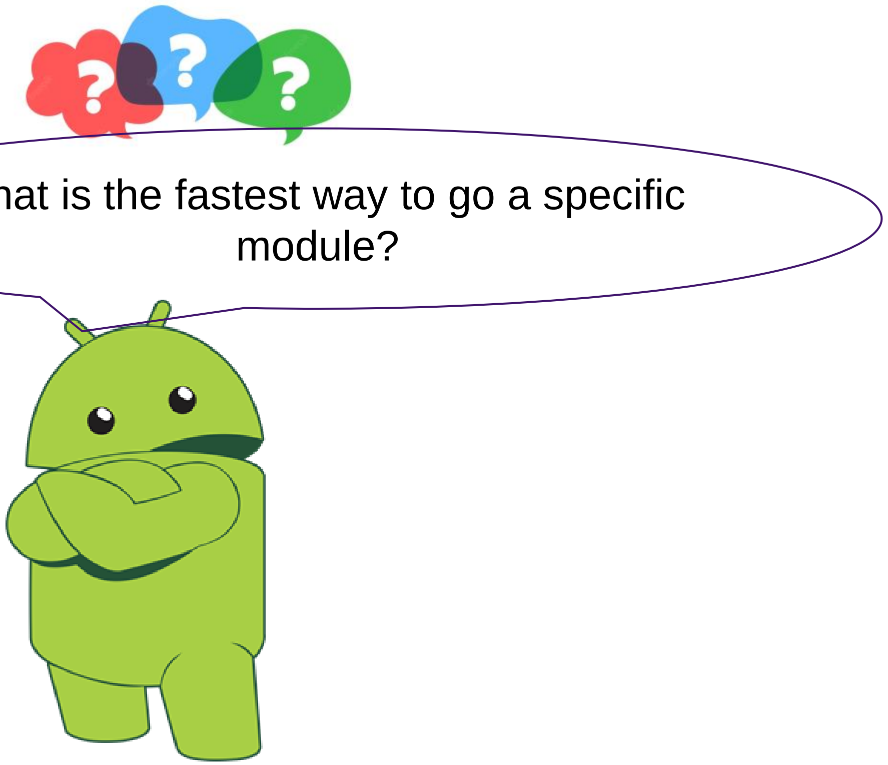
QUIZ (4/12)

How can we activate macros/functions that Android support?

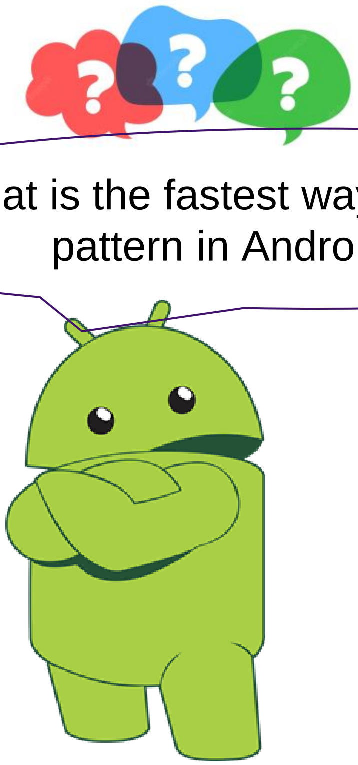
QUIZ (5/12)


What is the fastest way to go to root Android build system when we are at very deep subfolder?

QUIZ (6/12)

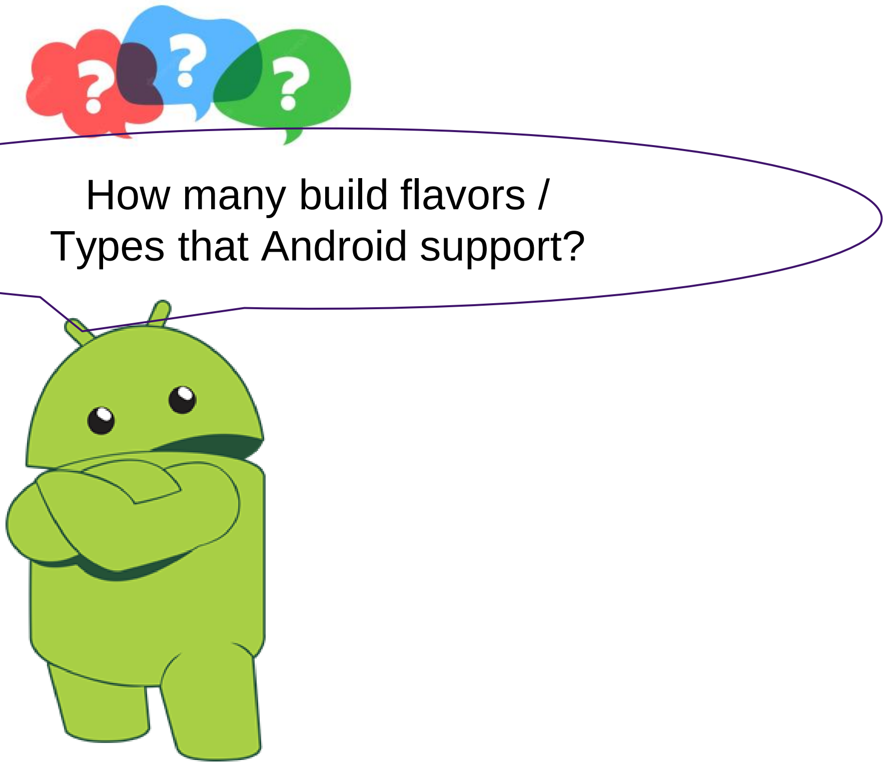


QUIZ (7/12)



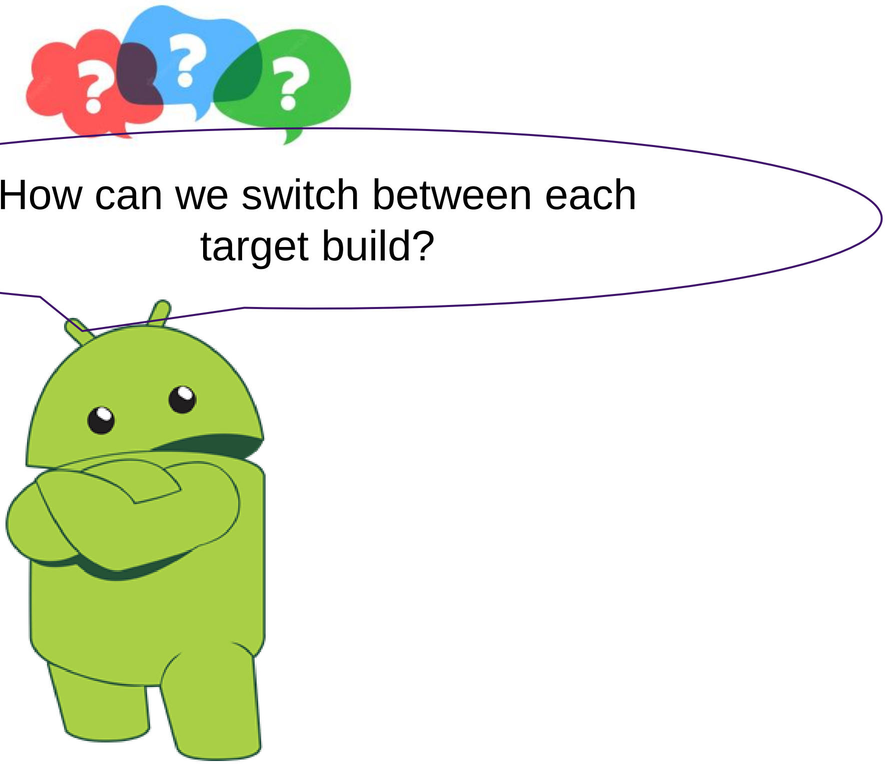
What is the fastest way to find a specific pattern in Android makefile?

QUIZ (8/12)

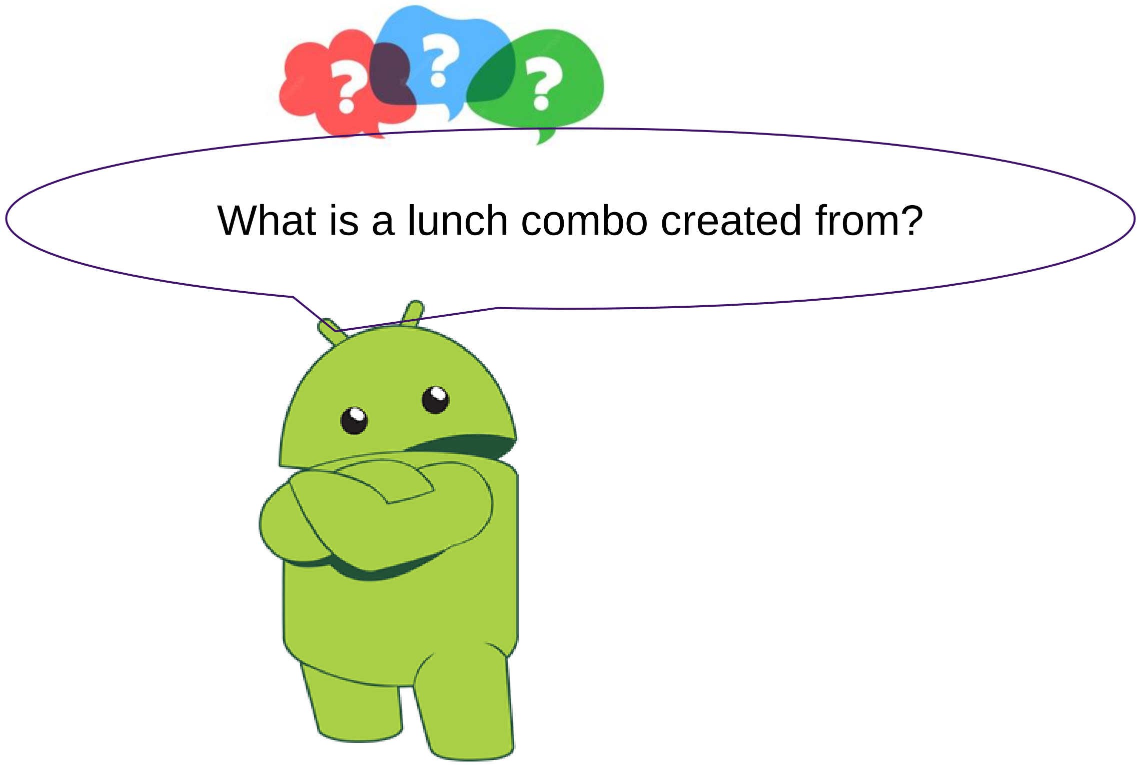


BOSCH

QUIZ (9/12)



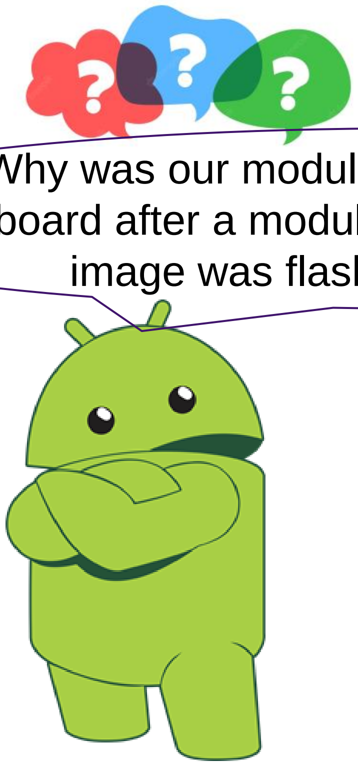
QUIZ (10/12)



QUIZ (11/12)

How can we build/clean a module?

QUIZ (12/12)



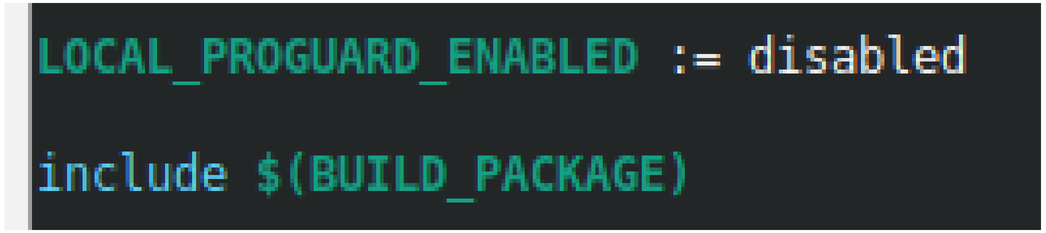
Why was our module not updated in the target board after a module build is finished and the image was flashed into target board?

Introduction

- Android is built from a **top-down perspective** and **Android.mk** is the smallest component.
- The system has been **architected** so that **module build recipes** are pretty much **independent** from the **build system's internals**.
- **Build templates** are provided so that module authors can get their modules built appropriately.
- **The target** is to make **Android.mk** to be fairly **lightweight**.

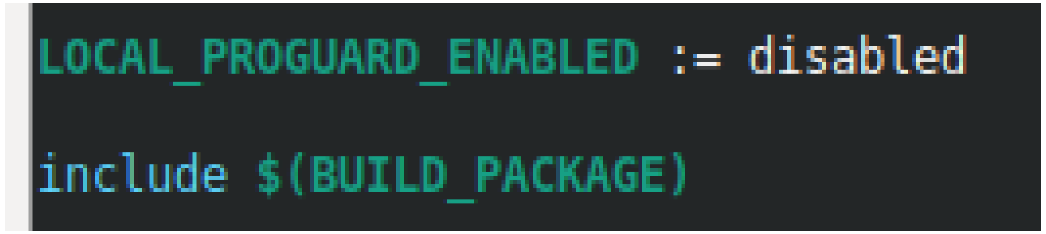
Introduction

- Each template is tailored for a specific type of module, and module authors can use a set of documented variables, all prefixed by **LOCAL_**, to modulate the templates' behavior and output.
- All templates can be found at **build/core/**
- **Android.mk** gets access to them through the **include** directive. Here's an example:



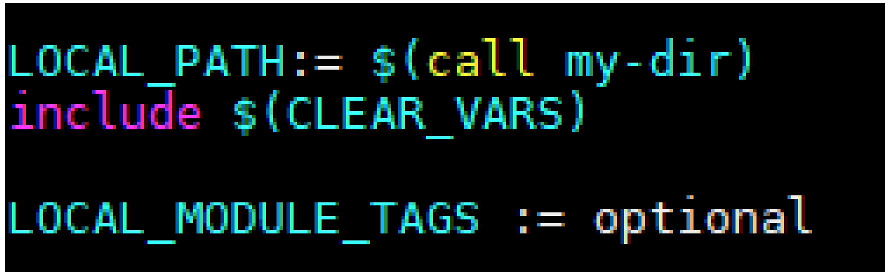
```
LOCAL_PROGUARD_ENABLED := disabled

include $(BUILD_PACKAGE)
```

Introduction



```
LOCAL_PROGUARD_ENABLED := disabled

include $(BUILD_PACKAGE)
```

- **Android.mk** files don’t actually **include the .mk templates by name**.
- Instead, they include a **variable that is set to the corresponding .mk** file.



```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := optional
```

Android build templates list

Variable	**Template**	**What it builds**	**Most notable use**
BUILD_EXECUTABLE	*executable.mk*	Target binaries	Native commands and daemons
BUILD_HOST_EXECUTABLE	*host_executable.mk*	Host binaries	Development tools
BUILD_JAVA_LIBRARY	*java_library.mk*	Target Java libraries	Apache Harmony and Android Framework
BUILD_STATIC_JAVA_LIBRARY	*static_java_library.mk*	Target static Java libraries	N/A
BUILD_HOST_JAVA_LIBRARY	*host_java_library.mk*	Host Java libraries	Development tools
BUILD_SHARED_LIBRARY	*shared_library.mk*	Target shared libraries	A vast number of modules, including many in *external/* and *frameworks/base/*
BUILD_STATIC_LIBRARY	*static_library.mk*	Target static libraries	A vast number of modules, including many in *external/*

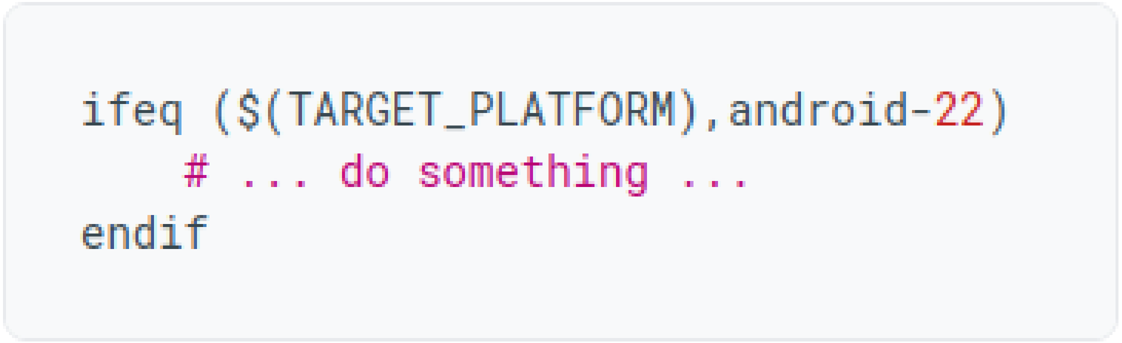
BOSCH

Android build templates list

Variable	Template	What it builds	Most notable use
BUILD_HOST_SHARED_LIBRARY	*host_shared_library.mk*	Host shared libraries	Development tools
BUILD_HOST_STATIC_LIBRARY	*host_static_library.mk*	Host static libraries	Development tools
BUILD_PREBUILT	*prebuilt.mk*	Copies prebuilt target files	Configuration files and binaries
BUILD_HOST_PREBUILT	*host_prebuilt.mk*	Copies prebuilt host files	Tools in *prebuilt/* and configuration files
BUILD_MULTI_PREBUILT	*multi_prebuilt.mk*	Copies prebuilt modules of multiple but known types, like Java libraries or executables	Rarely used
BUILD_PACKAGE	*package.mk*	Built-in AOSP apps (i.e., anything that ends up being an *.apk*)	All apps in the AOSP
CLEAR_VARS	clear_vars.mk	Make sure we won't have anything weird coming from other modules	Always used

Target information variables

- **TARGET_ARCH:**
  - **The CPU family the build system is targeting** as it parses this **Android.mk** file. This variable will be one of: **arm**, **arm64**, **x86**, or **x86_64**.
- **TARGET_PLATFORM:**
  - The **Android API level number** the build system is targeting as it parses this **Android.mk** file

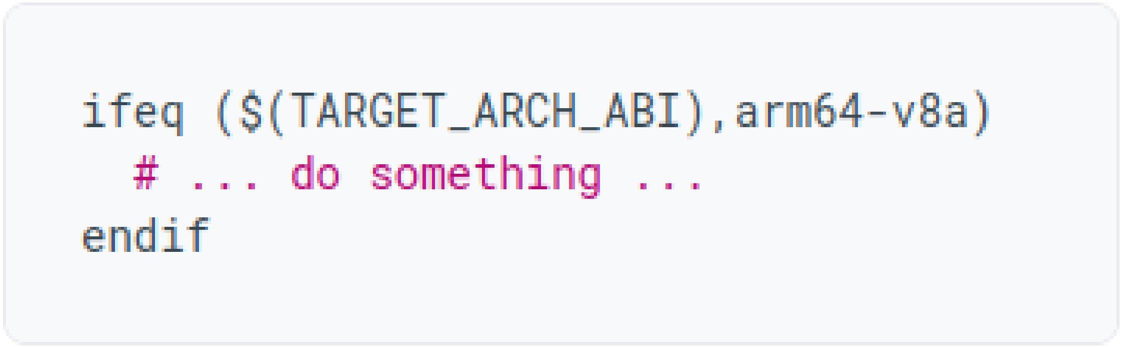


```
ifeq ($(TARGET_PLATFORM),android-22)
    # ... do something ...
endif
```

Target information variables

- **TARGET_ARCH_ABI:**
  - The **ABI build system** is targeting as it parses this **Android.mk** file.

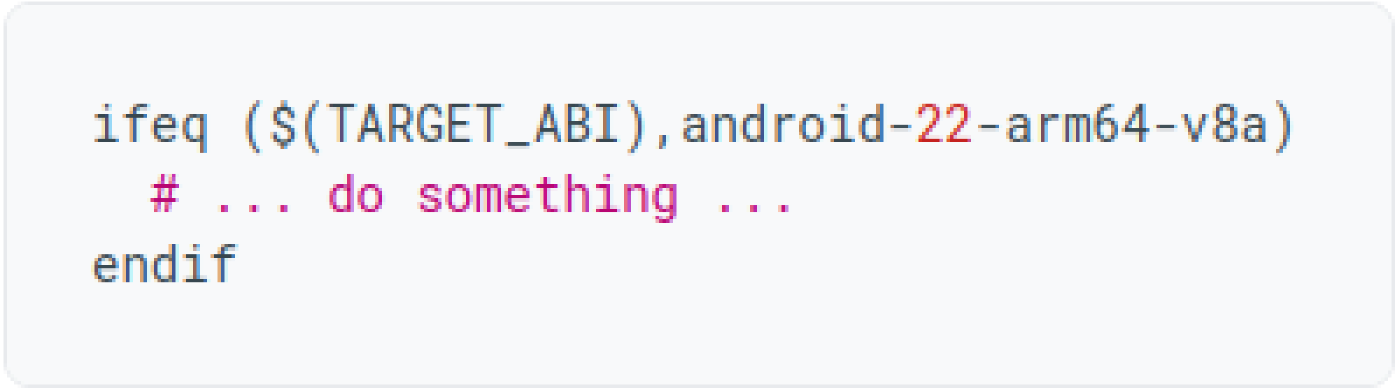
CPU and architecture	Setting
ARMv7	armeabi-v7a
ARMv8 AArch64	arm64-v8a
i686	x86
x86-64	x86_64



```
ifeq ($(TARGET_ARCH_ABI),arm64-v8a)
  # ... do something ...
endif
```

Target information variables

- **TARGET_ABI:**
  - A concatenation of target Android API level and ABI.
  - It is especially useful when you want to test against a specific target system image for a real device.
  - For example, to check for a 64-bit ARM device running on Android API level 22:



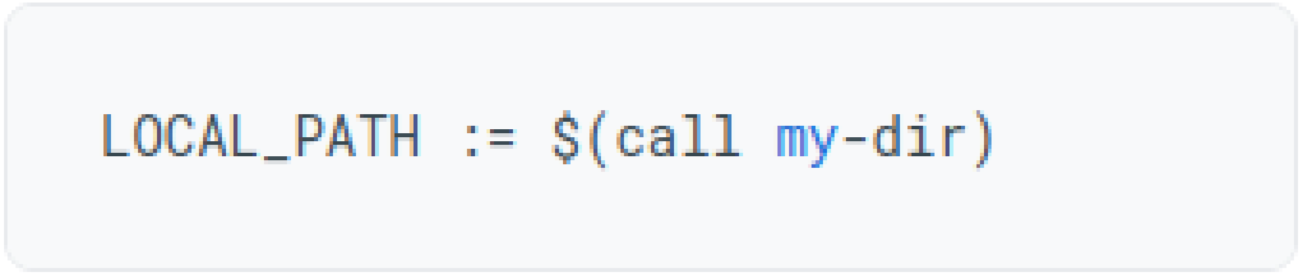
```
ifeq ($(TARGET_ABI),android-22-arm64-v8a)
  # ... do something ...
endif
```

Module-description variables

- Each module description should follow this basic flow:
    - **Initialize or undefine the variables** associated with the module, using the **CLEAR_VARS** variable.
    - **Assign values to the variables** used to describe the module.
    - **Set the build system** to use the appropriate build script for the module, using the **BUILD_XXX** variable.

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_PATH**:
    - The path of the current module's sources, typically provided by invoking **$(call my-dir).**
    - It is not cleared by **CLEAR_VARS**



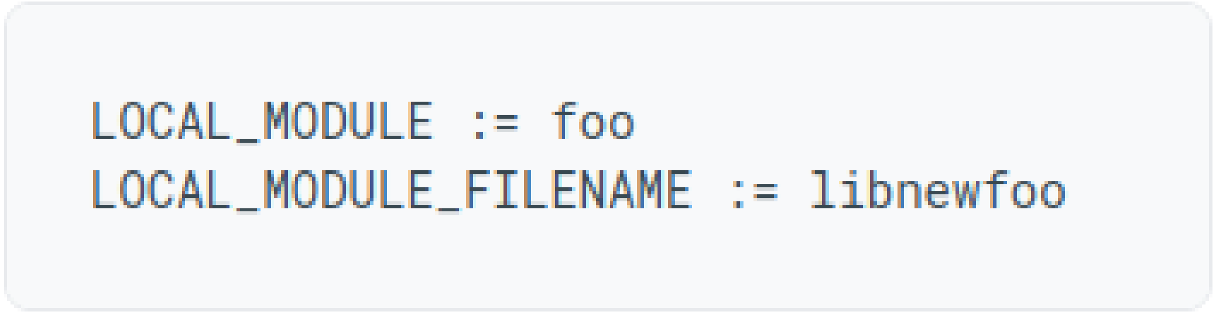
```
LOCAL_PATH := $(call my-dir)
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_MODULE:**
    - This variable stores the **name of our module**.
    - It **must be unique among all module names** and **must not contain any spaces**.
    - We **must define it** before including any scripts (except **CLEAR_VARS**).
    - We do **not need to add** either the **lib prefix or the .so or .a file extension**; the build system makes these modifications automatically.
    - If this is set to **foo**, for example, and we build an **executable**, then the final executable will be a command called **foo** and it will be put in the target's **/system/bin/.**
    - If this is set to **libfoo** and we include **BUILD_SHARED_LIBRARY** instead of **BUILD_EXECUTABLE**, the build system will generate **libfoo.so** and put it in **/system/lib/**.

Module-description variables

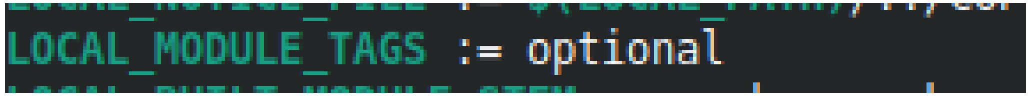
- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_MODULE_FILENAME:**
    - This optional variable allows you to **override the names that the build system uses by default for files** that it generates.
    - For example, if the name of your **LOCAL_MODULE** is **foo**, you can **force** the system to call the file it **generates libnewfoo**



```
LOCAL_MODULE := foo
LOCAL_MODULE_FILENAME := libnewfoo
```

Module-description variables

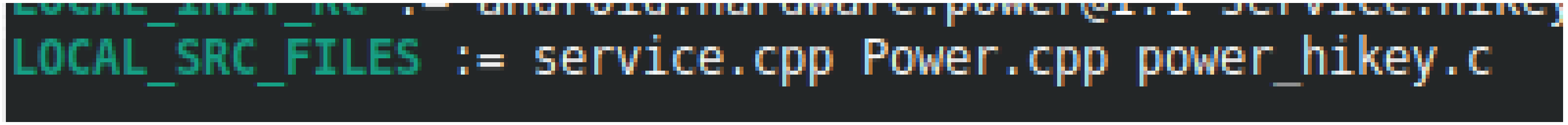
- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_MODULE_TAGS:**
    - This allows us to control under which TARGET_BUILD_VARIANT this module is built.
    - Usually, this should just be set to optional.



```
LOCAL_MODULE_TAGS := optional
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_SRC_FILES**
    - Contain the list of source files that the build system uses to generate the module.
    - Note:
      - Please **list the files** that **the build system passes to the compiler**, since the build system **automatically computes any associated dependencies**.
      - Both **relative** (to **LOCAL_PATH**) and **absolute file paths** can be used.
      - Please **avoiding absolute** file paths; **relative paths** make your **Android.mk** file **more portable**.



```
LOCAL_SRC_FILES := service.cpp Power.cpp power_hikey.c
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_CPP_EXTENSION**
    - We can use this optional variable to indicate a **file extension other than .cpp** for your C++ source files.
    - For example, the following line changes the extension to .cxx. (The setting must include the dot.):

```
LOCAL_CPP_EXTENSION := .cxx
```

    - We can use this variable to specify multiple extensions:

```
LOCAL_CPP_EXTENSION := .cxx .cpp .cc
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_CPP_FEATURES**
    - We can use this optional variable to indicate that **your code relies on specific C++ features**.
    - It enables the right compiler and linker flags **during the build process**.
    - For prebuilt binaries, this variable also declares **which features the binary depends on**, thus helping ensure the final linking works correctly.
    - **This variable should be used instead of enabling C++ feature directly** in your **LOCAL_CPPFLAGS** definition.

```
LOCAL_CPP_FEATURES := rtti
```

```
LOCAL_CPP_FEATURES := rtti features
```

    - **Note:**
      - The **order** in which we describe the values **does not matter**.

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_C_INCLUDES:**
    - We can use this optional variable to specify a list of paths, relative to android root directory, to add to the include search path when compiling all sources (C, C++ and Assembly).

```
LOCAL_C_INCLUDES := sources/foo
```

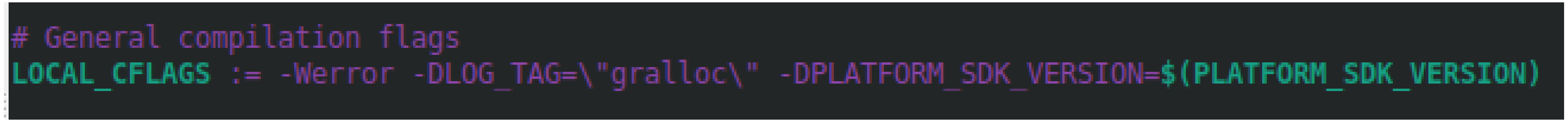
    - Or even:

```
LOCAL_C_INCLUDES := $(LOCAL_PATH)/<subdirectory>/foo
```

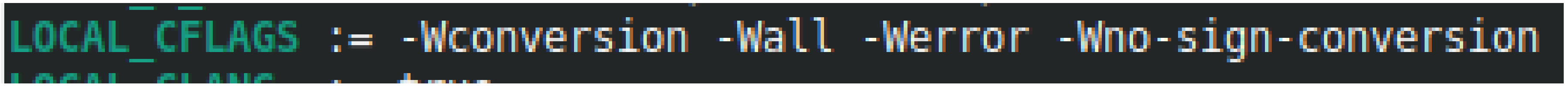
    - Note:
      - **Define this variable before setting** any corresponding inclusion flags via **LOCAL_CFLAGS** or **LOCAL_CPPFLAGS**.

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_CFLAGS:**
    - This optional variable sets compiler flags for the build system to pass when building C and C++ source files.
    - It is very useful for specifying additional macro definitions or compile options.



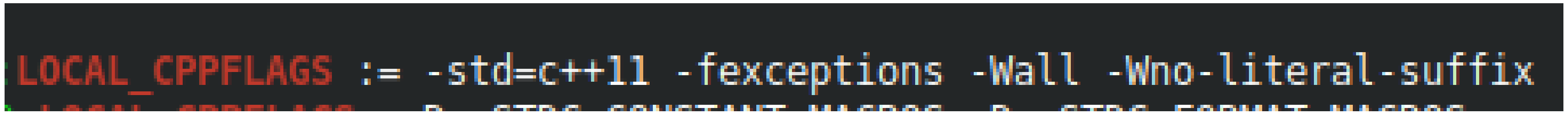
```
# General compilation flags
LOCAL_CFLAGS := -Werror -DLOG_TAG=\"gralloc\" -DPLATFORM_SDK_VERSION=$(PLATFORM_SDK_VERSION)
```



```
LOCAL_CFLAGS := -Wconversion -Wall -Werror -Wno-sign-conversion
```

Module-description variables

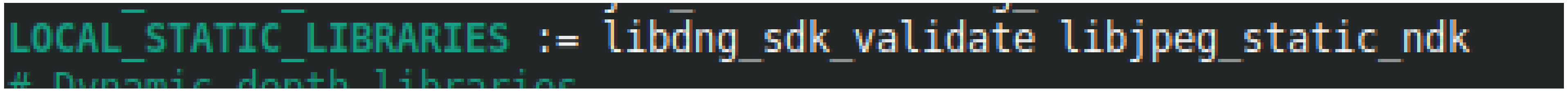
- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_CPPFLAGS:**
    - An optional set of compiler flags that will be passed when building **C++ source files only**.
    - They will **appear after the LOCAL_CFLAGS on the compiler's command-line**.



```
LOCAL_CPPFLAGS := -std=c++11 -fexceptions -Wall -Wno-literal-suffix
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_STATIC_LIBRARIES :**
    - This variable **stores the list of static libraries modules** on which the current module depends.
    - If the current module is **a shared library or an executable**, this variable **will force these libraries to be linked into the resulting binary**.
    - If the current module is **a static library**, this variable simply indicates that **other modules depending on the current one will also depend on the listed libraries**.

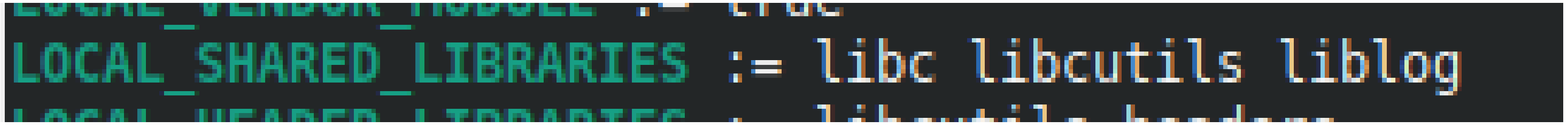


```
LOCAL_STATIC_LIBRARIES := libdng_sdk_validate libjpeg_static_ndk
# Dynamic depth libraries
```

BOSCH

Module-description variables

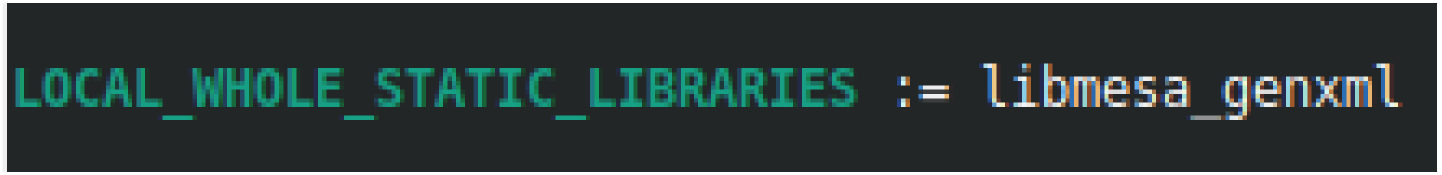
- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_SHARED_LIBRARIES:**
    - This variable is the list of **shared libraries modules on which this module depends at runtime**.
    - This information is **necessary at link time**, and to **embed the corresponding information in the generated file**.



```
LOCAL_SHARED_LIBRARIES := libc libcutils liblog
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_WHOLE_STATIC_LIBRARIES:**
    - This variable is a variant of **LOCAL_STATIC_LIBRARIES** and expresses that the linker **should treat the associated library modules as whole archives**.
    - This variable is **useful when there are circular dependencies among several static libraries**.
    - When we use this variable to **build a shared library**, it will **force the build system to add all object files from your static libraries to the final binary**



```
LOCAL_WHOLE_STATIC_LIBRARIES := libmesa_genxml
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_LDLIBS:**
    - This variable contains the list of **additional linker flags for use** in building our **shared library** or **executable**.
    - It enables us to use the **-l** prefix to **pass the name of specific system libraries**.
    - For example, the following example tells the linker to generate a module that links to **/system/lib/libz.so** at load time:

```
LOCAL_LDLIBS := -lz
```

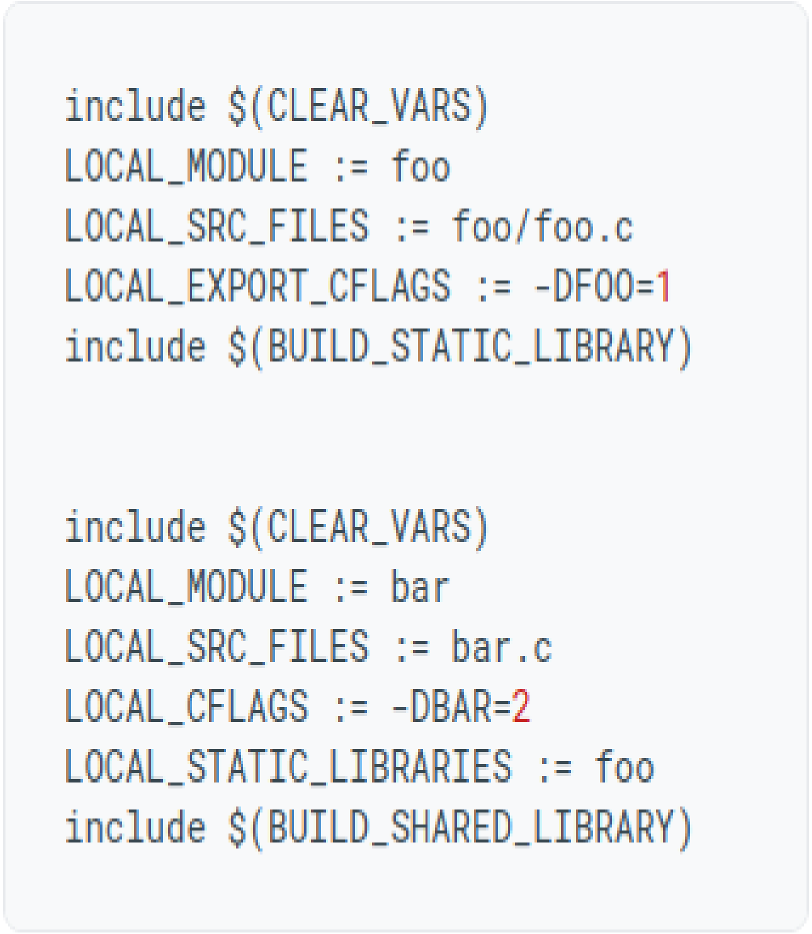
Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_LDFLAGS:**
    - The list of other **linker flags for the build system** to use when building your shared library or executable.
    - For example, to use the **ld.bfd** linker on ARM/X86:

```
LOCAL_LDFLAGS += -fuse-ld=bfd
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_EXPORT_CFLAGS:**
    - This variable **records a set of C/C++ compiler flags** to add to the **LOCAL_CFLAGS** definition of any other module that uses this one via the **LOCAL_STATIC_LIBRARIES** or **LOCAL_SHARED_LIBRARIES** variables.



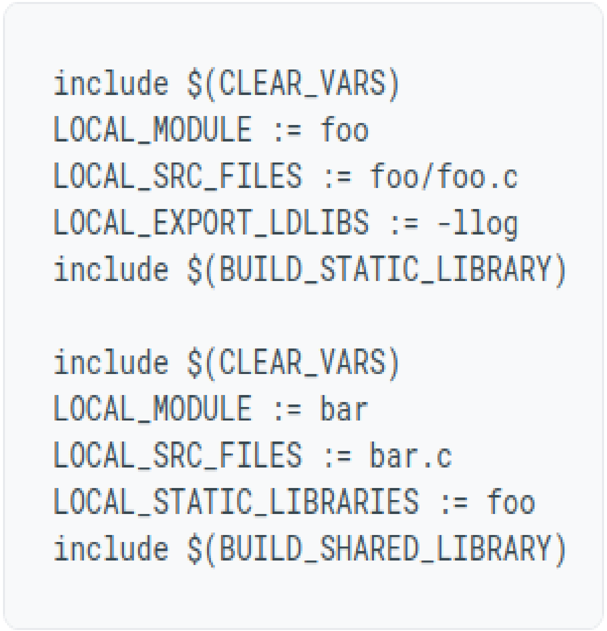
```
include $(CLEAR_VARS)
LOCAL_MODULE := foo
LOCAL_SRC_FILES := foo/foo.c
LOCAL_EXPORT_CFLAGS := -DFOO=1
include $(BUILD_STATIC_LIBRARY)


include $(CLEAR_VARS)
LOCAL_MODULE := bar
LOCAL_SRC_FILES := bar.c
LOCAL_CFLAGS := -DBAR=2
LOCAL_STATIC_LIBRARIES := foo
include $(BUILD_SHARED_LIBRARY)
```

Module-description variables

- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_EXPORT_CPPFLAGS:**
    - This variable is the same as **LOCAL_EXPORT_CFLAGS**, but for **C++ flags** only.
  - **LOCAL_EXPORT_C_INCLUDES:**
    - This variable is the same as **LOCAL_EXPORT_CFLAGS**, but for **C include paths**.
  - **LOCAL_EXPORT_LDFLAGS:**
    - This variable is the same as **LOCAL_EXPORT_CFLAGS**, but for **linker flags**

Module-description variables

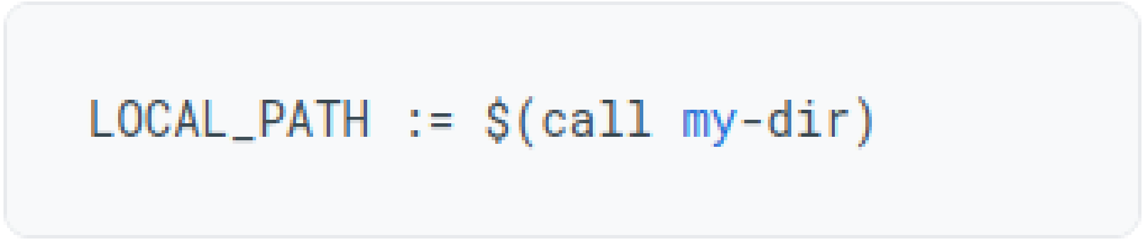
- Most frequently **LOCAL_*** variables to be used:
  - **LOCAL_EXPORT_LDLIBS:**
    - This variable is the same as **LOCAL_EXPORT_CFLAGS**, telling the build system **to pass names of specific system libraries to the compiler**. Prepend **-l** to the name of each library you specify.
    - Note that the build system appends imported linker flags to the value of your module's **LOCAL_LDLIBS** variable.
    - This variable is typically useful when module **foo** is a **static library and has code that depends on a system library**. We can use **LOCAL_EXPORT_LDLIBS** to export the dependency



```
include $(CLEAR_VARS)
LOCAL_MODULE := foo
LOCAL_SRC_FILES := foo/foo.c
LOCAL_EXPORT_LDLIBS := -llog
include $(BUILD_STATIC_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := bar
LOCAL_SRC_FILES := bar.c
LOCAL_STATIC_LIBRARIES := foo
include $(BUILD_SHARED_LIBRARY)
```

Supported function macros

- **my-dir:**
    - This macro returns **the path of the last included makefile**, which typically is the current **Android.mk**'s directory.
    - **my-dir** is useful for defining **LOCAL_PATH** at the start of your **Android.mk** file.



```
LOCAL_PATH := $(call my-dir)
```

- This macro will **return the path of the last makefile** that the build system included when parsing the build scripts.
- Therefore, we **should not call my-dir after including another file**.

Supported function macros

- **my-dir:**

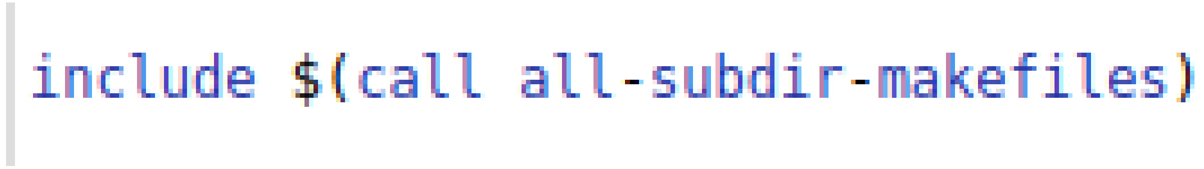
```
LOCAL_PATH := $(call my-dir)

# ... declare one module

include $(LOCAL_PATH)/foo/`Android.mk`

LOCAL_PATH := $(call my-dir)

# ... declare another module
```

```
LOCAL_PATH := $(call my-dir)

# ... declare one module

LOCAL_PATH := $(call my-dir)

# ... declare another module

# extra includes at the end of the Android.mk file
include $(LOCAL_PATH)/foo/Android.mk
```

```
MY_LOCAL_PATH := $(call my-dir)

LOCAL_PATH := $(MY_LOCAL_PATH)

# ... declare one module

include $(LOCAL_PATH)/foo/`Android.mk`

LOCAL_PATH := $(MY_LOCAL_PATH)

# ... declare another module
```

Supported function macros

- **all-subdir-makefiles:**
  - Returns **the list of Android.mk files** located in **all subdirectories** of the current **my-dir** path.
  - We can use this function to provide deep-nested source directory hierarchies to the build system.
  - By default, Android build system will only look for **files in the directory containing the Android.mk file**.

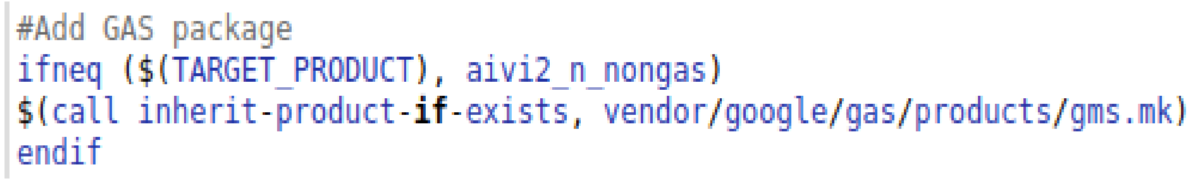


```
include $(call all-subdir-makefiles)
```

- **this-makefile:**
  - Returns the path of the **current makefile**
- **parent-makefile:**
  - Returns the path of **the parent makefile** in the inclusion tree (the path of the makefile that included the current one).

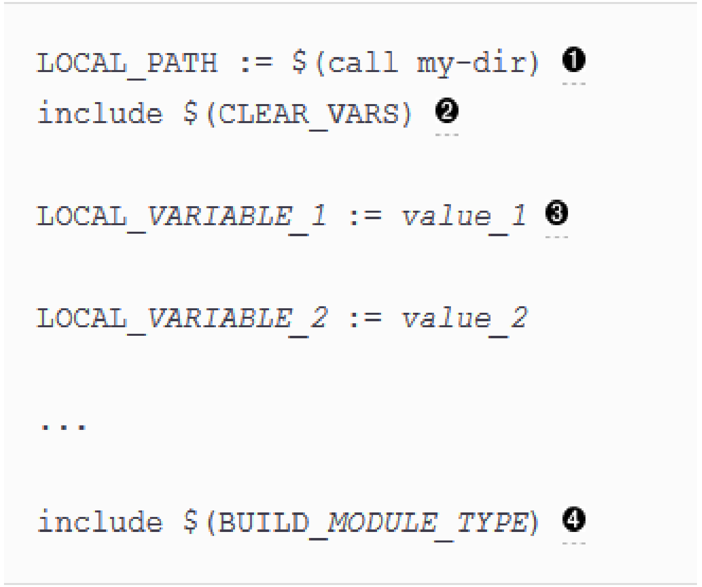
Supported function macros

- **grand-parent-makefile:**
  - Returns the path of the **grandparent makefile** in the inclusion tree (the path of the makefile that included the current one).
- **inherit-product:**
  - Inherits all of the variables from product.
  - Records the inheritance in the **.INHERITS_FROM** variable
  - Records that we've visited this node, in **ALL_PRODUCTS**
- **inherit-product-if-exists:**
  - Perform inherit-product only if product exists



```
#Add GAS package
ifneq ($(TARGET_PRODUCT), aivi2_n_nongas)
$(call inherit-product-if-exists, vendor/google/gas/products/gms.mk)
endif
```

Android.mk example

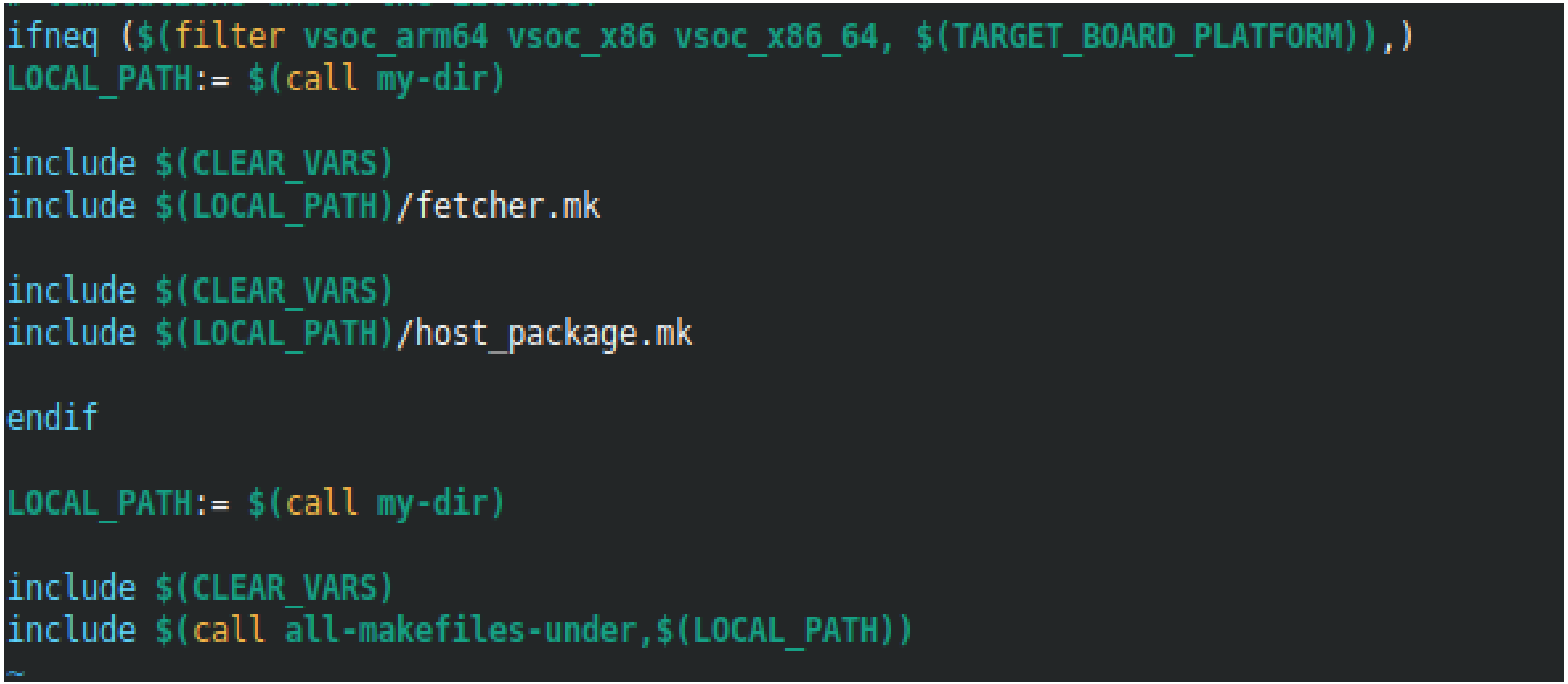


```
LOCAL_PATH := $(call my-dir) ❶
include $(CLEAR_VARS) ❷

LOCAL_VARIABLE_1 := value_1 ❸

LOCAL_VARIABLE_2 := value_2

...

include $(BUILD_MODULE_TYPE) ❹
```

1. Tell the build template where the current module is located. The macro function **my-dir**, provided by the build system, returns the path of the current directory (the directory containing the **Android.mk** file itself).
2. Clear all previously set **LOCAL_*** variables that might have been set for other modules.
3. Set various **LOCAL_*** variables to module-specific values.
4. Invoke the build template that corresponds to the current module's type.

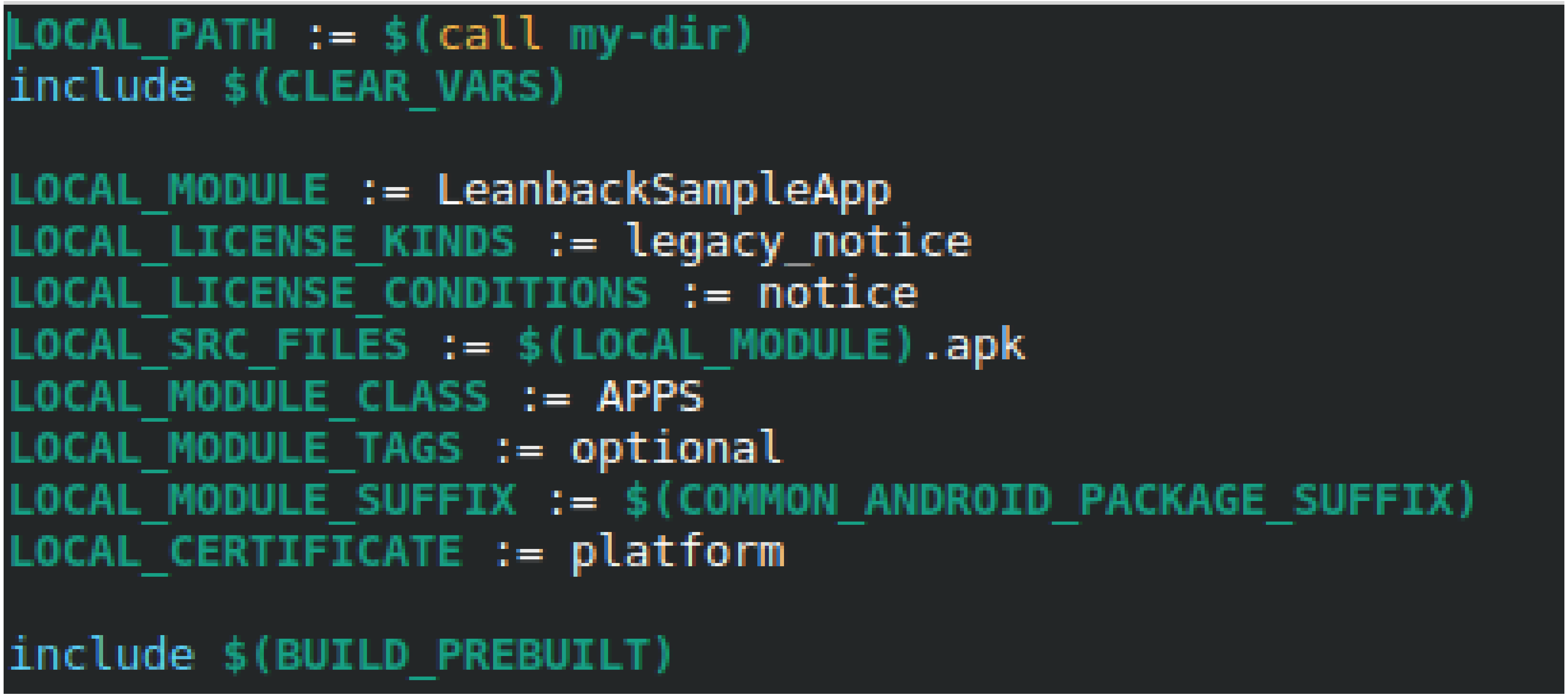
Group working (5 mins)

Group 1



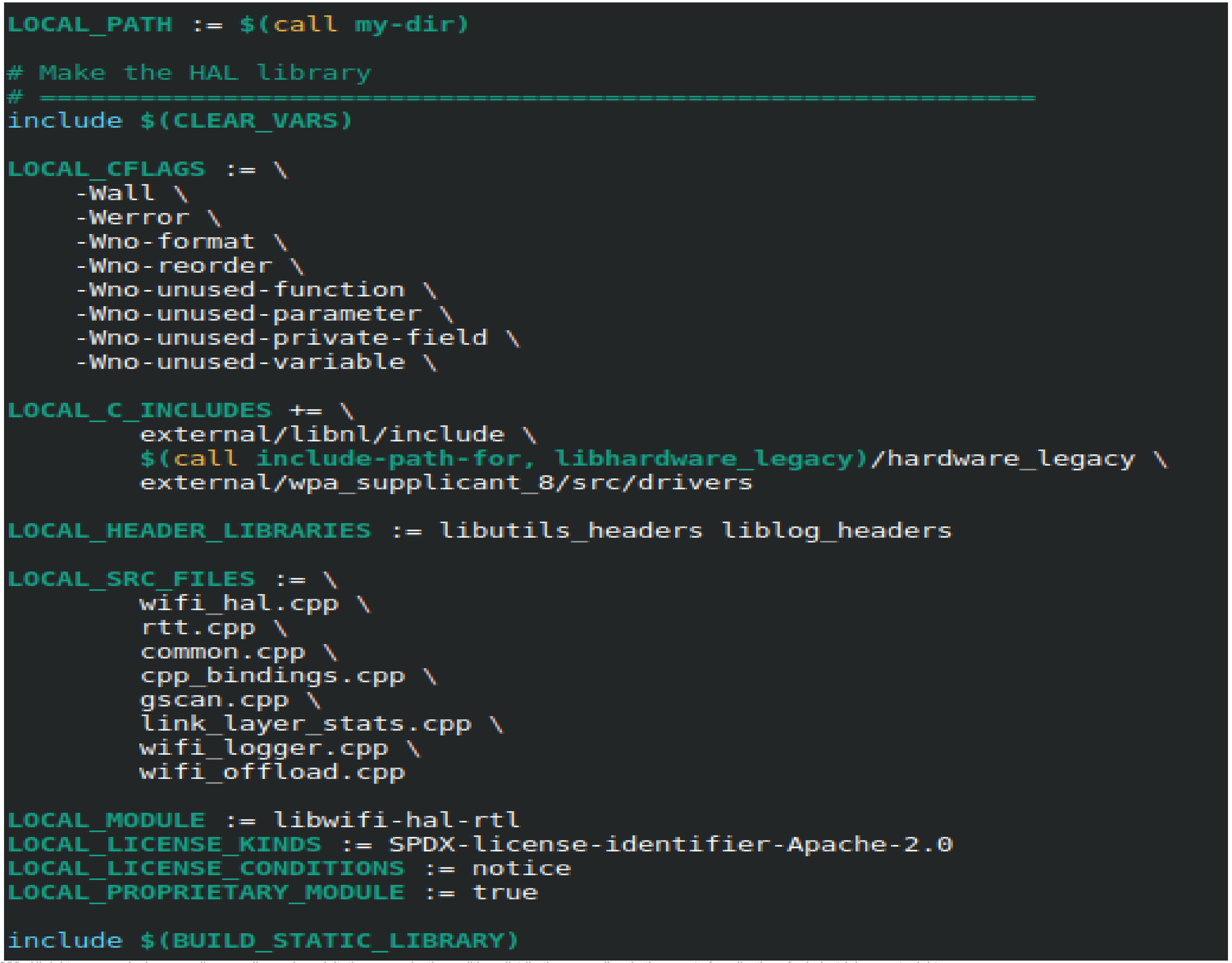
```
ifneq ($(filter vsoc_arm64 vsoc_x86 vsoc_x86_64, $(TARGET_BOARD_PLATFORM)),)
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
include $(LOCAL_PATH)/fetcher.mk

include $(CLEAR_VARS)
include $(LOCAL_PATH)/host_package.mk

endif

LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
include $(call all-makefiles-under,$(LOCAL_PATH))
~
```

Group 2



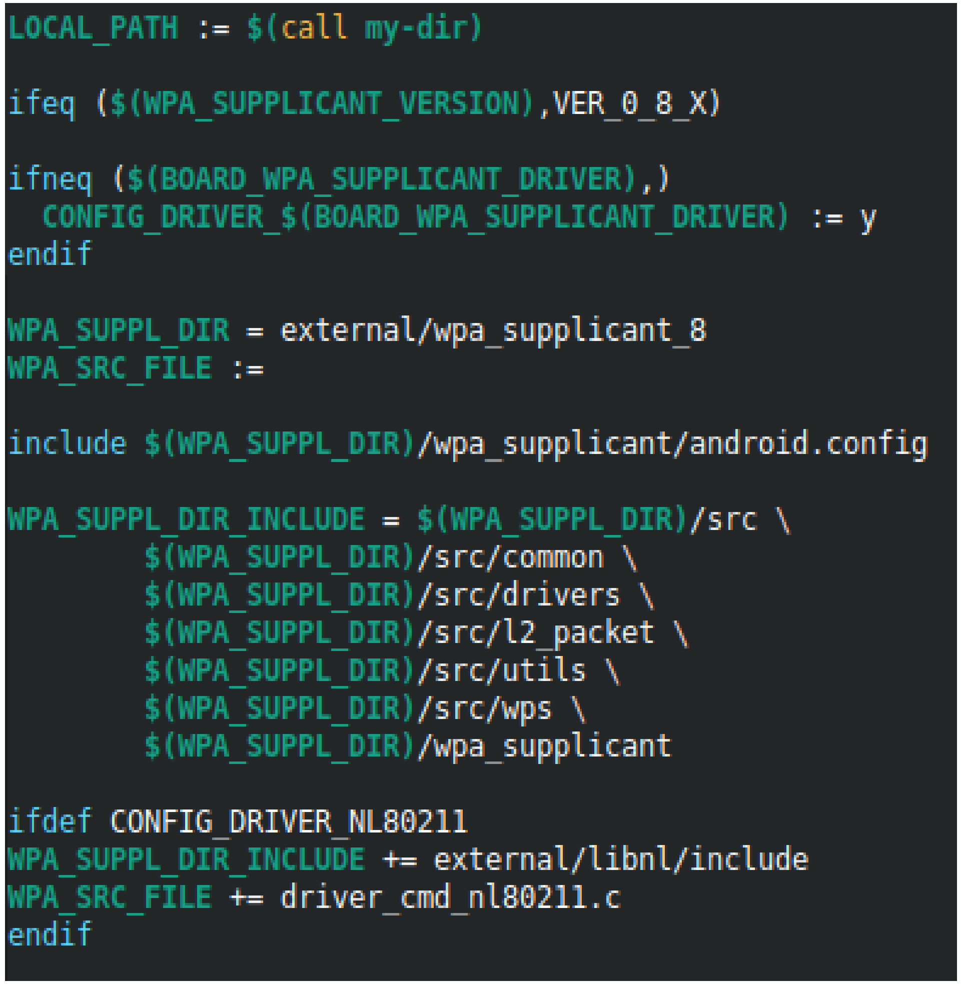
```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE := LeanbackSampleApp
LOCAL_LICENSE_KINDS := legacy_notice
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := platform

include $(BUILD_PREBUILT)
```

Group 3

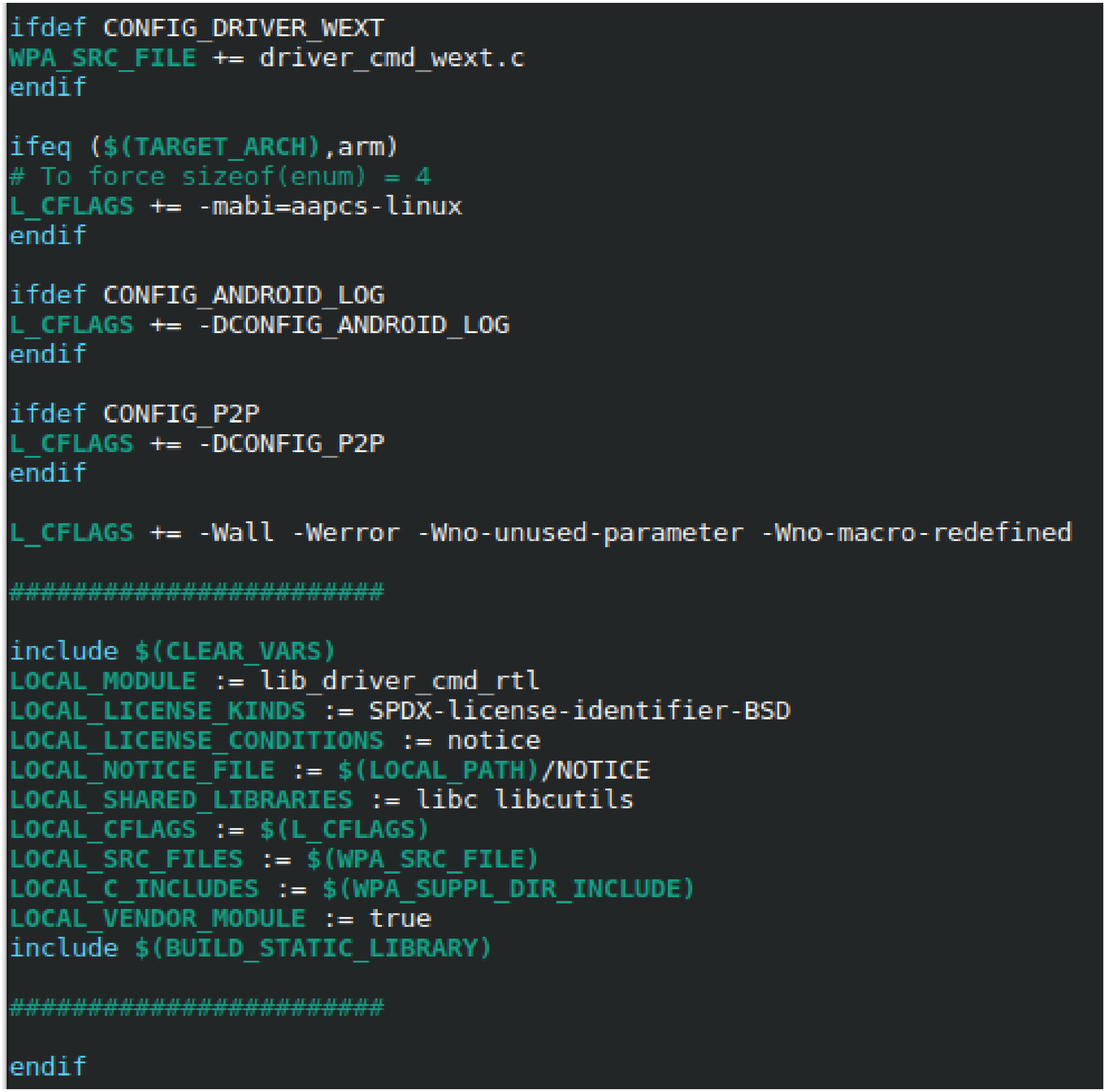


```
LOCAL_PATH := $(call my-dir)

# Make the HAL library
# ============================================================
include $(CLEAR_VARS)

LOCAL_CFLAGS := \
    -Wall \
    -Werror \
    -Wno-format \
    -Wno-reorder \
    -Wno-unused-function \
    -Wno-unused-parameter \
    -Wno-unused-private-field \
    -Wno-unused-variable \

LOCAL_C_INCLUDES += \
        external/libnl/include \
        $(call include-path-for, libhardware_legacy)/hardware_legacy \
        external/wpa_supplicant_8/src/drivers

LOCAL_HEADER_LIBRARIES := libutils_headers liblog_headers

LOCAL_SRC_FILES := \
        wifi_hal.cpp \
        rtt.cpp \
        common.cpp \
        cpp_bindings.cpp \
        gscan.cpp \
        link_layer_stats.cpp \
        wifi_logger.cpp \
        wifi_offload.cpp

LOCAL_MODULE := libwifi-hal-rtl
LOCAL_LICENSE_KINDS := SPDX-license-identifier-Apache-2.0
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_PROPRIETARY_MODULE := true

include $(BUILD_STATIC_LIBRARY)
```

Group 4

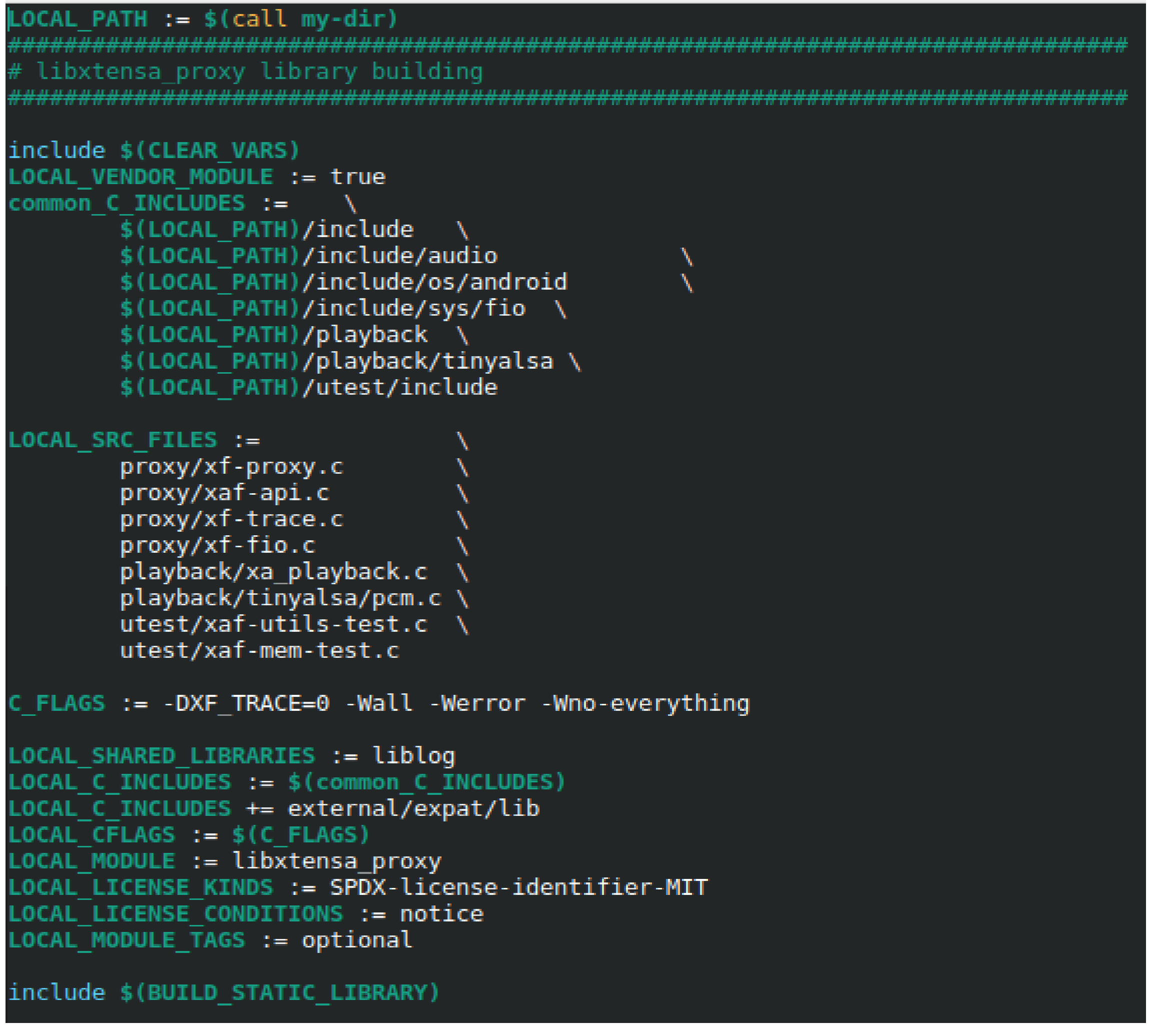


```
LOCAL_PATH := $(call my-dir)

ifeq ($(WPA_SUPPLICANT_VERSION),VER_0_8_X)

ifneq ($(BOARD_WPA_SUPPLICANT_DRIVER),)
  CONFIG_DRIVER_$(BOARD_WPA_SUPPLICANT_DRIVER) := y
endif

WPA_SUPPL_DIR = external/wpa_supplicant_8
WPA_SRC_FILE :=

include $(WPA_SUPPL_DIR)/wpa_supplicant/android.config

WPA_SUPPL_DIR_INCLUDE = $(WPA_SUPPL_DIR)/src \
        $(WPA_SUPPL_DIR)/src/common \
        $(WPA_SUPPL_DIR)/src/drivers \
        $(WPA_SUPPL_DIR)/src/l2_packet \
        $(WPA_SUPPL_DIR)/src/utils \
        $(WPA_SUPPL_DIR)/src/wps \
        $(WPA_SUPPL_DIR)/wpa_supplicant

ifdef CONFIG_DRIVER_NL80211
WPA_SUPPL_DIR_INCLUDE += external/libnl/include
WPA_SRC_FILE += driver_cmd_nl80211.c
endif
```

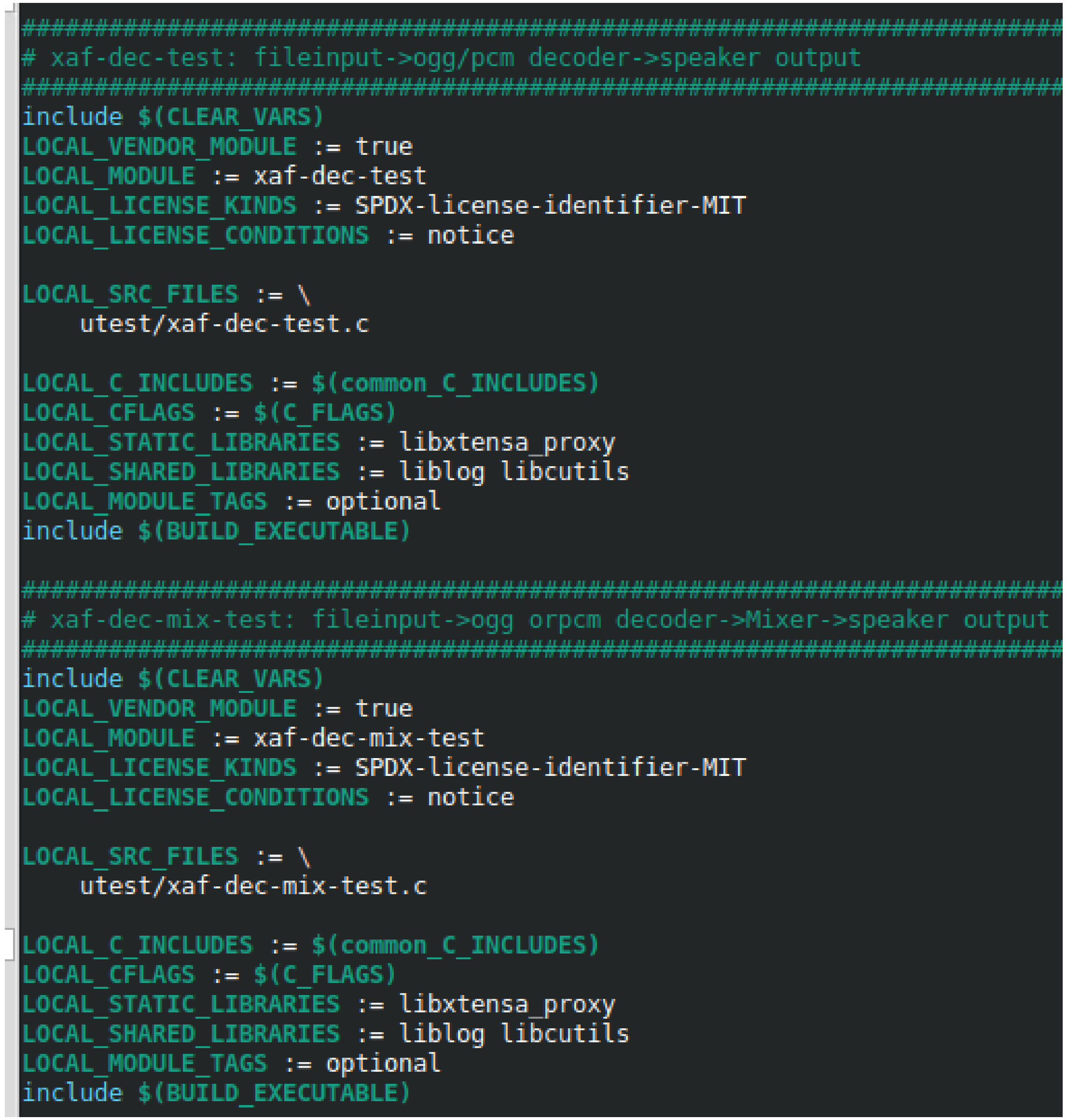


```
ifdef CONFIG_DRIVER_WEXT
WPA_SRC_FILE += driver_cmd_wext.c
endif

ifeq ($(TARGET_ARCH),arm)
# To force sizeof(enum) = 4
L_CFLAGS += -mabi=aapcs-linux
endif

ifdef CONFIG_ANDROID_LOG
L_CFLAGS += -DCONFIG_ANDROID_LOG
endif

ifdef CONFIG_P2P
L_CFLAGS += -DCONFIG_P2P
endif

L_CFLAGS += -Wall -Werror -Wno-unused-parameter -Wno-macro-redefined

########################

include $(CLEAR_VARS)
LOCAL_MODULE := lib_driver_cmd_rtl
LOCAL_LICENSE_KINDS := SPDX-license-identifier-BSD
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_NOTICE_FILE := $(LOCAL_PATH)/NOTICE
LOCAL_SHARED_LIBRARIES := libc libcutils
LOCAL_CFLAGS := $(L_CFLAGS)
LOCAL_SRC_FILES := $(WPA_SRC_FILE)
LOCAL_C_INCLUDES := $(WPA_SUPPL_DIR_INCLUDE)
LOCAL_VENDOR_MODULE := true
include $(BUILD_STATIC_LIBRARY)

########################

endif
```

Group 5

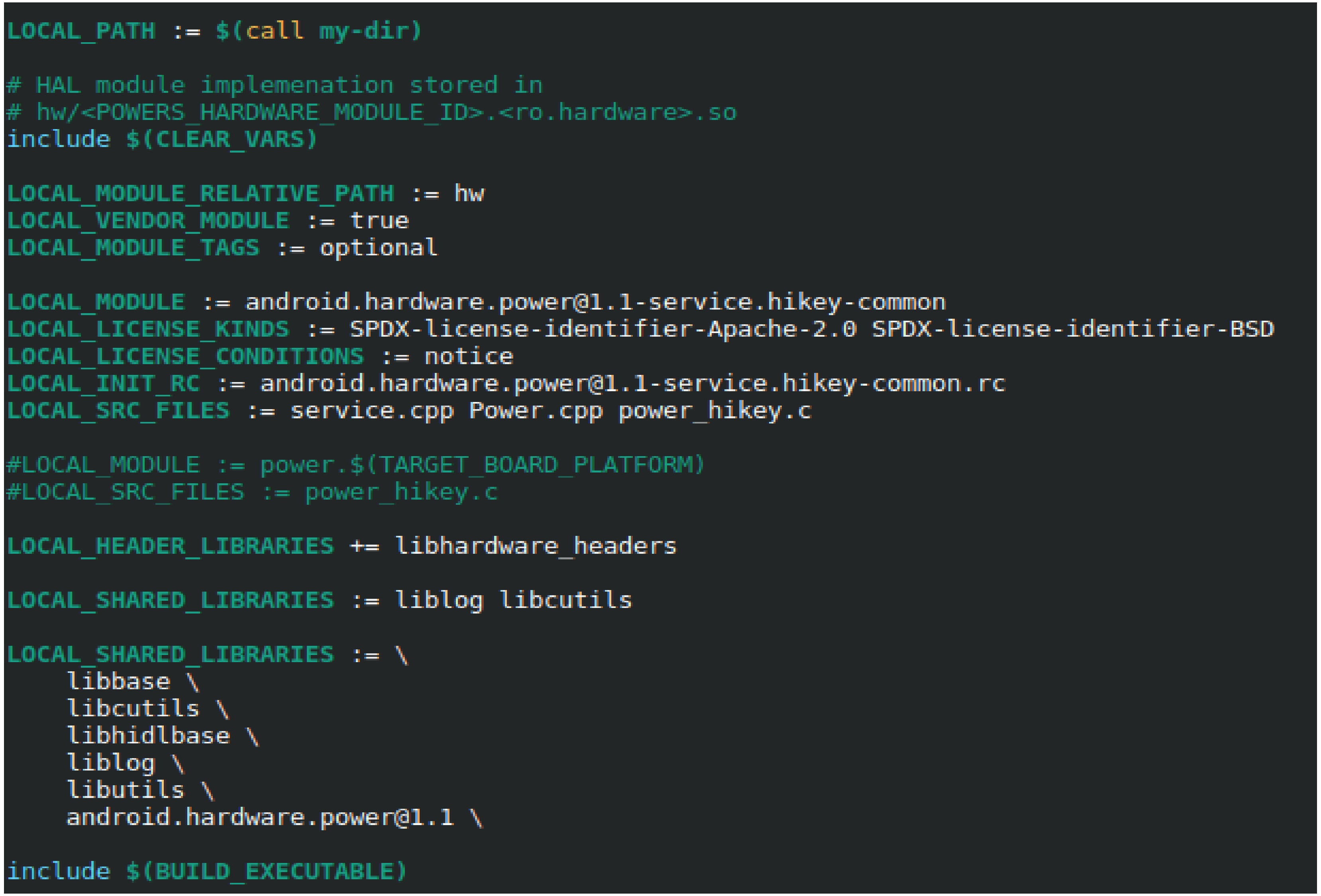


```
LOCAL_PATH := $(call my-dir)
################################################################################
# libxtensa_proxy library building
################################################################################

include $(CLEAR_VARS)
LOCAL_VENDOR_MODULE := true
common_C_INCLUDES :=    \
        $(LOCAL_PATH)/include   \
        $(LOCAL_PATH)/include/audio             \
        $(LOCAL_PATH)/include/os/android         \
        $(LOCAL_PATH)/include/sys/fio  \
        $(LOCAL_PATH)/playback  \
        $(LOCAL_PATH)/playback/tinyalsa \
        $(LOCAL_PATH)/utest/include

LOCAL_SRC_FILES :=              \
        proxy/xf-proxy.c        \
        proxy/xaf-api.c         \
        proxy/xf-trace.c        \
        proxy/xf-fio.c          \
        playback/xa_playback.c  \
        playback/tinyalsa/pcm.c \
        utest/xaf-utils-test.c  \
        utest/xaf-mem-test.c

C_FLAGS := -DXF_TRACE=0 -Wall -Werror -Wno-everything

LOCAL_SHARED_LIBRARIES := liblog
LOCAL_C_INCLUDES := $(common_C_INCLUDES)
LOCAL_C_INCLUDES += external/expat/lib
LOCAL_CFLAGS := $(C_FLAGS)
LOCAL_MODULE := libxtensa_proxy
LOCAL_LICENSE_KINDS := SPDX-license-identifier-MIT
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_MODULE_TAGS := optional

include $(BUILD_STATIC_LIBRARY)
```



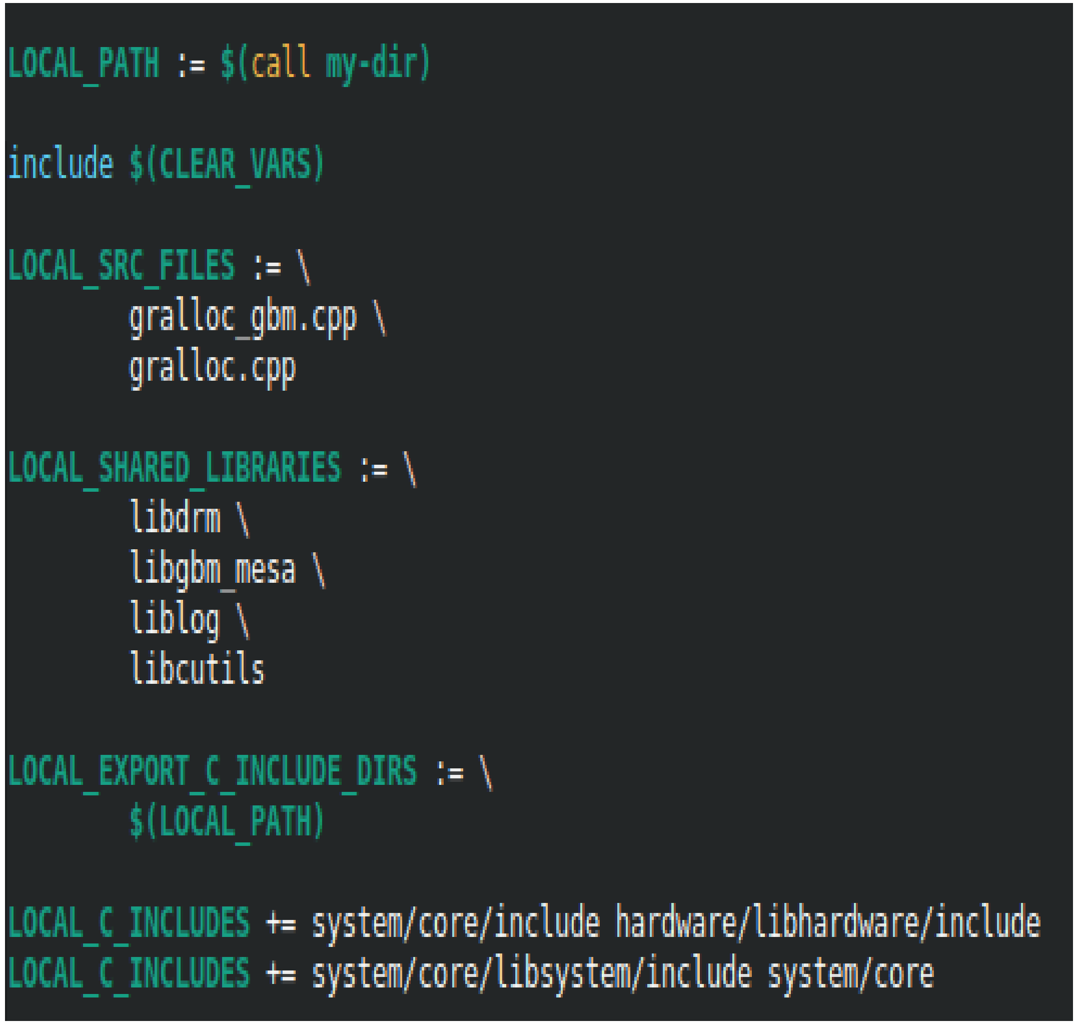
```
########################################################################
# xaf-dec-test: fileinput->ogg/pcm decoder->speaker output
########################################################################
include $(CLEAR_VARS)
LOCAL_VENDOR_MODULE := true
LOCAL_MODULE := xaf-dec-test
LOCAL_LICENSE_KINDS := SPDX-license-identifier-MIT
LOCAL_LICENSE_CONDITIONS := notice

LOCAL_SRC_FILES := \
    utest/xaf-dec-test.c

LOCAL_C_INCLUDES := $(common_C_INCLUDES)
LOCAL_CFLAGS := $(C_FLAGS)
LOCAL_STATIC_LIBRARIES := libxtensa_proxy
LOCAL_SHARED_LIBRARIES := liblog libcutils
LOCAL_MODULE_TAGS := optional
include $(BUILD_EXECUTABLE)

########################################################################
# xaf-dec-mix-test: fileinput->ogg orpcm decoder->Mixer->speaker output
########################################################################
include $(CLEAR_VARS)
LOCAL_VENDOR_MODULE := true
LOCAL_MODULE := xaf-dec-mix-test
LOCAL_LICENSE_KINDS := SPDX-license-identifier-MIT
LOCAL_LICENSE_CONDITIONS := notice

LOCAL_SRC_FILES := \
    utest/xaf-dec-mix-test.c

LOCAL_C_INCLUDES := $(common_C_INCLUDES)
LOCAL_CFLAGS := $(C_FLAGS)
LOCAL_STATIC_LIBRARIES := libxtensa_proxy
LOCAL_SHARED_LIBRARIES := liblog libcutils
LOCAL_MODULE_TAGS := optional
include $(BUILD_EXECUTABLE)
```

Group 6

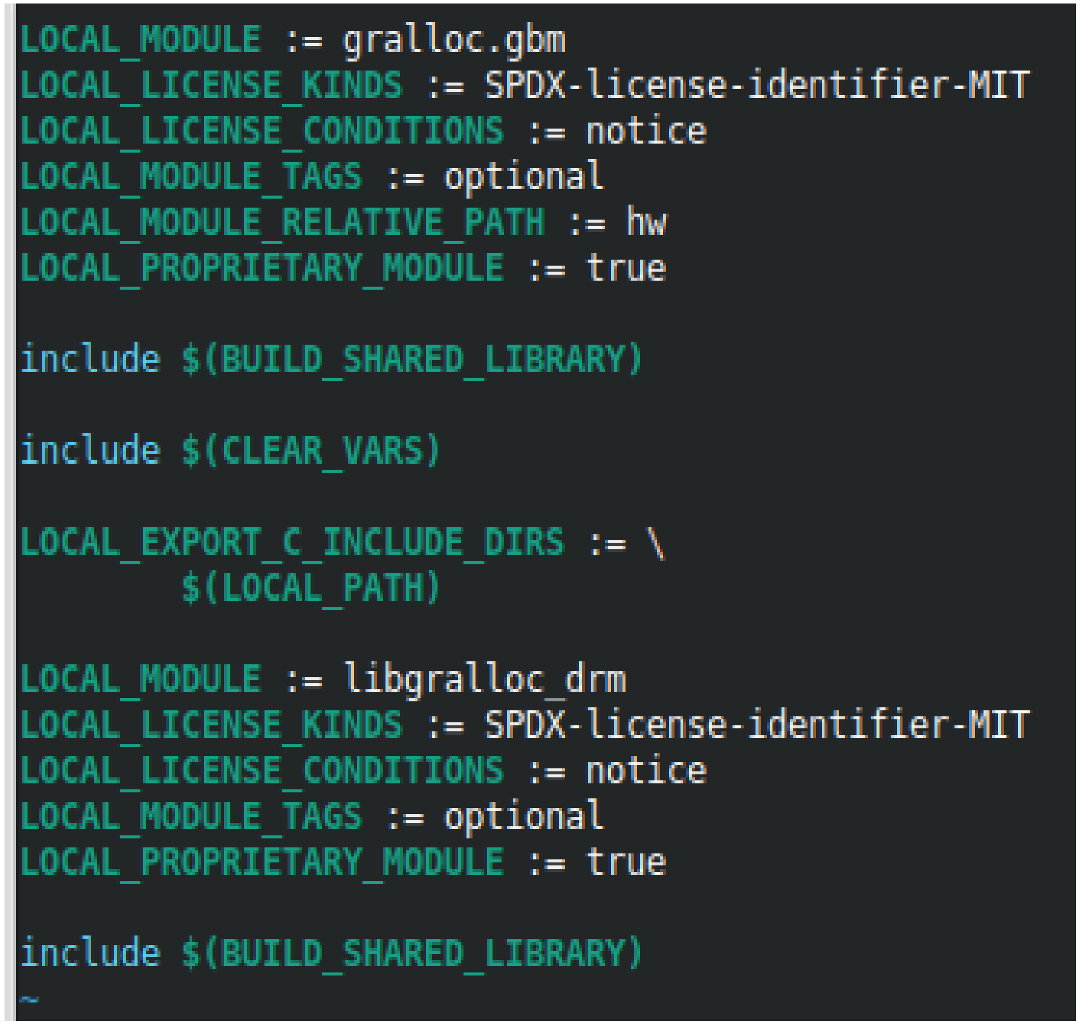


```
LOCAL_PATH := $(call my-dir)

# HAL module implemenation stored in
# hw/<POWERS_HARDWARE_MODULE_ID>.<ro.hardware>.so
include $(CLEAR_VARS)

LOCAL_MODULE_RELATIVE_PATH := hw
LOCAL_VENDOR_MODULE := true
LOCAL_MODULE_TAGS := optional

LOCAL_MODULE := android.hardware.power@1.1-service.hikey-common
LOCAL_LICENSE_KINDS := SPDX-license-identifier-Apache-2.0 SPDX-license-identifier-BSD
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_INIT_RC := android.hardware.power@1.1-service.hikey-common.rc
LOCAL_SRC_FILES := service.cpp Power.cpp power_hikey.c

#LOCAL_MODULE := power.$(TARGET_BOARD_PLATFORM)
#LOCAL_SRC_FILES := power_hikey.c

LOCAL_HEADER_LIBRARIES += libhardware_headers

LOCAL_SHARED_LIBRARIES := liblog libcutils

LOCAL_SHARED_LIBRARIES := \
    libbase \
    libcutils \
    libhidlbase \
    liblog \
    libutils \
    android.hardware.power@1.1 \

include $(BUILD_EXECUTABLE)
```

Group 7



```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_SRC_FILES := \
        gralloc_gbm.cpp \
        gralloc.cpp

LOCAL_SHARED_LIBRARIES := \
        libdrm \
        libgbm_mesa \
        liblog \
        libcutils

LOCAL_EXPORT_C_INCLUDE_DIRS := \
        $(LOCAL_PATH)

LOCAL_C_INCLUDES += system/core/include hardware/libhardware/include
LOCAL_C_INCLUDES += system/core/libsystem/include system/core
```



```
LOCAL_MODULE := gralloc.gbm
LOCAL_LICENSE_KINDS := SPDX-license-identifier-MIT
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_RELATIVE_PATH := hw
LOCAL_PROPRIETARY_MODULE := true

include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)

LOCAL_EXPORT_C_INCLUDE_DIRS := \
        $(LOCAL_PATH)

LOCAL_MODULE := libgralloc_drm
LOCAL_LICENSE_KINDS := SPDX-license-identifier-MIT
LOCAL_LICENSE_CONDITIONS := notice
LOCAL_MODULE_TAGS := optional
LOCAL_PROPRIETARY_MODULE := true

include $(BUILD_SHARED_LIBRARY)
~
```

Exercises

Exercises

Write an **Android.mk** file to build C/C++ source code to:

a) An executable file

b) A static library

c) A share libraries

Thank for your listening!

BOSCH