CM 代码编译结构

刘涛

July 4, 2013

Version	Date	Author	Comments
0.01	2013-03-18	刘涛	Draft

表 1: Version history

目录

1 前言

1.1 目的

这是项目研发的总结文档，介绍 CM(CyanogenMod /sa.'æn.o.d□□n.m□d/) 编译脚本和研发人员的关注点。这篇文档的预期读者是 Android 平台软件开发者。特别使用于 ROM 的开发者。

1.2 介绍

本文涉及 CM 编译，编译环境, 编译模块，编译 kernel 等。

1.3 定义

NA

1.4 名词解释

NA

1.5 参考文档

NA

2 CM 介绍

颜色测试

CyanogenMod 是智能手机和平板电脑的增强性的开源的固件，它基于安卓手机的操作系统。提供了一些安卓官方的代码中没有的 feature 和 options. 当然，它是由一个开源组织开发的.CM 所支持的 feature 包括本地主题，无损音频压缩编码，大接入点名称列表，开放的 VPN 客户端，增强型的重启菜单. 硬件方面支持 wifi,usb,cpu 超频和其他增强性能，软键盘和其他平板调节，通知开关，应用的权限管理，和其他特有的接口。总之，CM 就是在 android 源生基础上做的增强的功能。

CM 是自由开放的源码，基于谷歌官方的源代码，增添了第三方的应用。每年谷歌一到两个版本发布 AOSP(Android open source project)。CM 然后基于该代码 porting 到新旧几十个设备上去。同时其他的 CM 开发商开始添加功能，修复和改善一些功能。有些谷歌新开发的功能会从 CM 上借鉴过来。

CM 是一个操作系统，可以替代预装在手机或平板中的系统。如果你已经有了一部老的不能再继续更新系统的手机，或者你的手机运行速度比较慢，再或者你的手机系统存在很多你不能删除的垃圾软件，你都可以选用 CM 来作为你的设备的操作系统。

3 CM 开发流程

以 I9300（GALAXY S III）为例，首先打开 $http://wiki.cyanogenmod.org/w/Devices\#vendor=$ 这个页面，vendor 选择 samsung,type 选择 phone,CM version 我选择 CM10.1，然后页面下面会列

出CM的代码说支持的设备，选择GALAXY S III，也就是在$http://wiki.cyanogenmod.org/w/I9300_Info$这个页面中会详细的讲解对该设备的简单介绍。接着打开详细的链接：How to Build CM for the Galaxy S III (International)，里面有详细的讲解如何进行该设备的开发。

先下载CM的代码$repoinit-ugit://github.com/CyanogenMod/android.git-bcm-10.1$，然后 repo sync 这个代码一般是要下载两天的时间。下载完毕之后，进入 vendor/cm 目录执行./get-prebuilds, 这个脚本的作用就是从网络上下载已经编译好的 APK，并且安装到 android 源码中，该动作只需要做一次，以后不需要再执行这个脚本。第二步就是准备设备相关的代码，主要是设备的配置文件和 kernel 文件，执行命令 source build/envsetup.sh 和 breakfast i9300，就会把 kernel 的代码和关于 i9300 的配置代码下载下来。第三步，用 USB 连接 GALAXY S III，在 device/samsung/i9300 目录下执行./extract-files.sh，这样就把该设备需要的库文件和固件下载下来。接下来就可以像高通平台一样的开发步骤了。

4 CM 的编译流程

4.1 envsetup.sh

解释一下 envsetup.sh 中的函数

```
function help()                          显示帮助信息
function get_abs_build_var()             获取绝对变量
function get_build_var()                 获取绝对变量
function check_product()                 检查product
function check_variant()                 检查变量
function setpaths()                      设置文件路径
function printconfig()                   打印配置
function set_stuff_for_environment()     设置环境变量
function set_sequence_number()           设置序号
function settitle()                      设置标题
function choosetype()                    设置type
function chooseproduct()                 设置product
function choosevariant()                 设置variant
function tapas()                         功能同choosecombo
function choosecombo()                   设置编译参数
function add_lunch_combo()               添加lunch项目
function print_lunch_menu()              打印lunch列表
function lunch()                         配置lunch
function m()                             make from top
function findmakefile()                  查找makefile
function mm()                            make from current directory
function mmm()                           make the supplied directories
function croot()                         回到根目录
function cproj()
function pid()
function systemstack()
function gdbclient()
```

```
function jgrep()                        查找java文件
function cgrep()                        查找c/cpp文件
function resgrep()
function tracedmdump()
function runhat()
function getbugreports()
function startviewserver()
function stopviewserver()
function isviewserverstarted()
function smoketest()
function runtest()
function godir ()                       跳到指定目录

# add the default one here
add_lunch_combo generic-eng

# Execute the contents of any vendorsetup.sh files we can find.
for f in `/bin/ls vendor/*/vendorsetup.sh vendor/*/build/vendorsetup.sh 2> /de
do
    echo "including $f"
    . $f
done
```

4.2 lunch 函数

```
chiplua@chiplua:~/work/cm4.1$ lunch
You're building on Linux
Lunch menu... pick a combo:
     1. full-eng
     2. full_x86-eng
     3. vbox_x86-eng
     4. mini_armv7a_neon-userdebug
     5. mini_armv7a-userdebug
     6. full_panda-userdebug
     7. cm_anzu-userdebug
     8. cm_coconut-userdebug
     9. cm_e610-userdebug
     10. cm_encore-userdebug
     11. cm_haida-userdebug
     12. cm_hallon-userdebug
     13. cm_holiday-userdebug
     14. cm_iyokan-userdebug
     15. cm_mango-userdebug
     16. cm_mint-userdebug
     17. cm_p1c-userdebug
     18. cm_p1l-userdebug
```

```
     19. cm_p1n-userdebug
     20. cm_p1-userdebug
     21. cm_p700-userdebug
     22. cm_p720-userdebug
     23. cm_p920-userdebug
     24. cm_pyramid-userdebug
     25. cm_ruby-userdebug
     26. cm_satsuma-userdebug
     27. cm_smultron-userdebug
     28. cm_t769-userdebug
     29. cm_tf101-userdebug
     30. cm_tf201-userdebug
     31. cm_urushi-userdebug
Which would you like? [full-eng]
```

用户也可以直接输入参数，不使用菜单。例如：

```
chiplua@chiplua:~/work/cm4.1$ lunch
 cm_i9300
```

下面是lunch函数源代码，用蓝色添加了一下注释，便于阅读：

```
function lunch()
{
    local answer

    if [ "$1" ] ; then
       # lunch后面直接带参数
        answer=$1
    else
        # lunch后面不带参数，则打印处所有的target product和variant菜
单提供用户选择
        print_lunch_menu
        echo -n "Which would you like? [generic-eng] "
        read answer
    fi

    local selection=

    if [ -z "$answer" ]
    then
                # 如果用户在菜单中没有选择，直接回车，则为系统缺省的
generic-eng
        selection=generic-eng
    elif [ "$answer" = "simulator" ]
    then
        # 如果是模拟器
        selection=simulator
    elif (echo -n $answer | grep -q -e "^[0-9][0-9]*$")
    then
```

```
        # 如果answer是选择菜单的数字，则获取该数字对应的字符串
        if [ $answer -le ${#LUNCH_MENU_CHOICES[@]} ]
        then
        selection=${LUNCH_MENU_CHOICES[$(($answer-$_arrayoffset))]}
        fi
        # 如果 answer字符串匹配 *-*模式(*的开头不能为-)
    elif (echo -n $answer | grep -q -e "^[^\-][^\-]*-[^\-][^\-]*$")
    then
        selection=$answer
    fi

    if [ -z "$selection" ]
    then
        echo
        echo "Invalid lunch combo: $answer"
        return 1
    fi

    # special case the simulator
    if [ "$selection" = "simulator" ]
    then
        # 模拟器模式
        export TARGET_PRODUCT=sim
        export TARGET_BUILD_VARIANT=eng
        export TARGET_SIMULATOR=true
        export TARGET_BUILD_TYPE=debug
    else

        # 将 product-variant模式种的product分离出来
        local product=$(echo -n $selection | sed -e "s/-.*$//")

     # 检查之，调用关系 check_product()->get_build_var()->build/core/config.m
比较罗嗦，不展开了
        check_product $product
        if [ $? -ne 0 ]
        then
            echo
            echo "** Don't have a product spec for: '$product'"
            echo "** Do you have the right repo manifest?"
            product=
        fi

        # 将 product-variant模式种的variant分离出来
        local variant=$(echo -n $selection | sed -e "s/^[^\-]*-//")

        # 检查之，看看是否在 (user userdebug eng) 范围内
```

```
        check_variant $variant
        if [ $? -ne 0 ]
        then
            echo
            echo "** Invalid variant: '$variant'"
            echo "** Must be one of ${VARIANT_CHOICES[@]}"
            variant=
        fi

        if [ -z "$product" -o -z "$variant" ]
        then
            echo
            return 1
        fi

        export TARGET_PRODUCT=$product
        export TARGET_BUILD_VARIANT=$variant
        export TARGET_SIMULATOR=false
        export TARGET_BUILD_TYPE=release
    fi # !simulator

    echo

   # 设置到环境变量，比较多，不再一一列出，最简单的方法 set >env.txt 可
获得
    set_stuff_for_environment
   # 打印一些主要的变量, 调用关系 printconfig()->get_build_var()->build/core/
较罗嗦，不展开了
    printconfig
}
```

4.3 envsetup.sh 提供的几个 shell 命令

用$. build/envsetup.sh可以引入到shell环境中。下面整理并简述。特别，envsetup.sh
还同时会引入vendor/和device /目录下的vendorsetup.sh脚本。
help

显示帮助，列出提供的命令
get_abs_build_var

列出make脚本中某变量的值，前缀上当前路径。ref dumpvar.mk
使用方法:

```
        get_abs_build_var VAR_NAME
```

VAR_NAME是需要显示的make脚本中的变量。
例如:

```
        get_abs_build_var TARGET_PRODUCT
```

返回
```
        <Your Android Root>\<VAR_NAME Value>
```

get_build_var

列出make脚本中某变量的值。ref dumpvar.mk
Usage：
```
        get_build_var VAR_NAME
```
VAR_NAME是需要显示的make脚本中的变量。
Return：
```
        <VAR_NAME Value>
```
Example:
```
        get_abs_build_var TARGET_PRODUCT
```
check_product

检查指定的TARGET_PRODUCT是否允许，默认的有sim和generic。如果不允许，则输出错误信息，允许则无回显。
Usage:
```
        check_product <YourTargetProduct>
```
Example：
```
        check_product generic
```
check_variant

检查variant是否支持，支持则返回0，不支持则返回1。允许的variant列表定义在envsetup.sh中的VARIANT_CHOICES中，默认是user，userdebug，eng。定制android时，可以在VARIANT_CHOICES中添加vairant。
Usage：
```
        check_variant <YourVariant>
```
Example：
```
        check_variant eng
```
setpaths

奇次执行时，将ANDROID_BUILD_PATHS路径加到PATH中。偶次执行时，将ANDROID_BUILD_
路径从PATH中去除。ANDROID_BUILD_PATHS包括android编译中要使用到的路径，例如ANDROID_EABI_TOOLCHAIN，ANDROID_TOOLCHAIN，ANDROID_QTOOLS，ANDROID_JAVA_
ANDROID_PRODUCT_OUT等等。

Usage：
```
        setpaths
```
printconfig

输出类似如下形势的配置信息。
```
============================================
PLATFORM_VERSION_CODENAME=AOSP
PLATFORM_VERSION=AOSP
TARGET_PRODUCT=generic
```

```
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=arm
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=OPENMASTER
============================================
set_stuff_for_environment
```

依次调用settitle， set_java_home，setpaths，set_sequence_number。设置android编译需要的环境变量。

```
set_sequence_number
```

输出环境变量BUILD_ENV_SEQUENCE_NUMBER。

```
settitle
```

设置shell的prompt提示，PROMPT_COMMAND中加入TARGET_PRODUCT，TARGET_BUILD_VARI和TARGET_BUILD_APPS等信息提示。

```
choosesim
```

配置环境变量TARGET_SIMULATOR。linux下会提示用户选择device或simulator。然后调用set_stuff_for_environment设置。

```
choosetype
```

配置环境变量TARGET_BUILD_TYPE_SIMULATOR。会提示用户选择release或debug。然后调用set_stuff_for_environment设置。

```
chooseproduct
```

配置环境变量TARGET_PRODUCT。会提示用户选择release或debug。然后调用set_stuff_设置。

```
choosevariant
```

配置环境变量TARGET_BUILD_VARIANT。会提示用户选择release或debug。

```
choosecombo
```

依次调用choosesim，choosetype，chooseproduct，choosevariant，set_stuff_for_en配置，然后调用printconfig输出。

```
add_lunch_combo
```

向环境变量LUNCH_MENU_CHOICES标识的列表中添加项。envsetup.sh中默认添加了full-eng，full_x86-eng，和simulator。

```
print_lunch_menu
```

列出LUNCH_MENU_CHOICES中的所有选项。
lunch

点菜，用户选择/指定product，variant后，lunch命令设置环境变量TARGET_PRODUCT，TARGET_BUILD_VARIANT，TARGET_SIMULATOR，TARGET_BUILD_TYPE，随后调用set_stuff_
设置，并printconfig显示。
Usage：
lunch [<YourProduct>-<YourBuildVariant>]

不给参数时，将提示用户选择。

Example：
lunch
lunch generic-eng
tapas

用户给定variant和一个或多个app name，就是LOCAL_PACKAGE_NAME的名字。tapas
设定
export TARGET_PRODUCT=generic
export TARGET_BUILD_VARIANT=$variant
export TARGET_SIMULATOR=false
export TARGET_BUILD_TYPE=release
export TARGET_BUILD_APPS=$apps
Usage：
tapas <YourVariant>? <YourAppName>*
?代表可选，*代表0个，1个或多个。YourVariant 和YourAppName的次序可颠倒。

Example：
tapas user Calculator Calender
gettop

返回当前android代码树的顶层路径。前提是当前路径位于android代码树中。
m

等价于在当前android代码树的顶层路径下执行make命令。
findmakefile

查找当前或最接近自己的祖辈路径上的Android.mk，返回Android.mk的路径，假
设当前路径处于android代码树中。
mm

如果当前路径是代码树顶层，则mm相当于make。如果是深层，测mm相当于
ONE_SHOT_MAKEFILE=$M make -C $T files $@

$M是findmakefile发现的Android.mk，$T是代码树顶层路径，files是main.mk中

定义的phony goal，就是完成$M对应目录范围内，所有android需编译的modules以及辅助说明txt文件。
mmm

给定package的路径，则mm会make相应的package。
例如，
mmm package/apps/Calculator

croot

改变当前路径到代码树顶层。
cproj

改变当前路径到最近的还有Android.mk文件的祖父辈路径。
pid

使用adb shell ps命令列出手机上指定名字的进程的pid。
Usage:
pid <YourName>
systemstack

使用kill -3system_server将系统进程中的线程信息写入/data/anr/traces.txt。

gdbclient

建立gdb调试环境，包括两步，手机上运行gdbserver，本机上运行arm-eabi-gdb。

Usage：
gdbclient <EXE> <PORT> <AppName>
EXE: AppName的执行名。

PORT：gdbserver的端口，例如, 192.168.2.102:5039
AppName：手机中ps列出的app名字，据此查pid。
sgrep

查找当前目录及子目录中所有.c，.h，.cpp，.S，.java，.mk，.xml，.sh文件，即源码文件中包含特定单词的行，并颜色显示输出。
Usage：
sgrep <YourWord>
Example:
sgrep Calendar
jgrep

同sgrep，但只查.java文件。
cgrep

同sgrep，但只查c相关的文件，即.c，.cc，.cpp，.h文件。
resgrep

同sgrep，但只查res相关的.xml文件。
mgrep

同sgrep，但只查make相关的脚本文件，包括Makefile文件，Makefile目录下的所有文件，.make文件，.mak文件和.mk文件。
treegrep

查找当前目录及子目录中所有.c，.h，.cpp，.S，.java，.xml文件，即源码文件中包含特定单词的行，并颜色显示输出。
getprebuilt

输出prebuilt的路径。
tracedmdump

生成dexlist文件qtrace.dexlit，dmtrace数据文件dmtrace，和调用dmtracedump工具生成的dmtrace解析文件dmtrace.html，将生成文件放到指定路径。
Usage：
tracedmdump <YourDirName>
如果YourDirName中不含’\’，则将放置的路径是$ANDROID_PRODUCT_OUT/traces/YourD

runhat

貌似使用kill -10的方法得到heap dump并取到本地。使用hat以http方式展现出来。hat可能是个lightweight http server，不曾用过。
getbugreports

将手机/sdcard/bugreports目录下的文件下载到本地并压缩打包。
startviewserver

用指定端口启动viewserver。
Usage：
startviewserver <Port>
不指定端口，则默认4939。
stopviewserver

关闭viewserver。
isviewserverstarted

检查viewserver是否可用。
smoketest

编译smoketest并安装手机运行。
runtest

运行development/testrunner/runtest.py $@

godir

给出一个词，godir会输出一个路径列表供用户选择要进入的路径。路径列表包含的路径满足，路径名中包含这个词，或这路径下的文件有文件名含这个词。out/路径下不考虑。

```
Usage：
        godir <YourKey>
Usage：
        godir Calculator
```

set_java_home

设置JAVA_HOME环境变量为/usr/lib/jvm/java-6-sun。

5 main.mk

```
# Only use ANDROID_BUILD_SHELL to wrap around bash.
# DO NOT use other shells such as zsh.
ifdef ANDROID_BUILD_SHELL
SHELL := $(ANDROID_BUILD_SHELL)
else
# Use bash, not whatever shell somebody has installed as /bin/sh
# This is repeated in config.mk, since envsetup.sh runs that file
# directly.
SHELL := /bin/bash
endif
#一些规则打开关闭
# this turns off the suffix rules built into make
.SUFFIXES:

# this turns off the RCS / SCCS implicit rules of GNU Make
% : RCS/%,v
% : RCS/%
% : %,v
% : s.%
% : SCCS/s.%

# If a rule fails, delete $@.
.DELETE_ON_ERROR:

# Figure out where we are.
#TOP := $(dir $(word $(words $(MAKEFILE_LIST)),$(MAKEFILE_LIST)))
#TOP := $(patsubst %/,%,$(TOP))
```

```
# TOPDIR is the normal variable you should use, because
# if we are executing relative to the current directory
# it can be "", whereas TOP must be "." which causes
# pattern matching probles when make strips off the
# trailing "./" from paths in various places.
#ifeq ($(TOP),.)
#TOPDIR :=
#else
#TOPDIR := $(TOP)/
#endif

#保证shell的版本在3.8.1或3.8.2
# Check for broken versions of make.
# (Allow any version under Cygwin since we don't actually build the platform the
ifeq (,$(findstring CYGWIN,$(shell uname -sm)))
ifeq (0,$(shell expr $$(echo $(MAKE_VERSION) | sed "s/[^0-9\.].*//") = 3.81))
ifeq (0,$(shell expr $$(echo $(MAKE_VERSION) | sed "s/[^0-9\.].*//") = 3.82))
$(warning ************************************************************
$(warning *  You are using version $(MAKE_VERSION) of make.)
$(warning *  Android is tested to build with versions 3.81 and 3.82.)
$(warning *  see https://source.android.com/source/download.html)
$(warning ************************************************************
endif
endif
endif

#获取工作目录
# Absolute path of the present working direcotry.
# This overrides the shell variable $PWD, which does not necessarily points to
# the top of the source tree, for example when "make -C" is used in m/mm/mmm.
PWD := $(shell pwd)

TOP := .
TOPDIR :=

BUILD_SYSTEM := $(TOPDIR)build/core

# This is the default target.  It must be the first declared target.
.PHONY: droid
DEFAULT_GOAL := droid
$(DEFAULT_GOAL):

# Used to force goals to build.  Only use for conditionally defined goals.
.PHONY: FORCE
FORCE:
```

```
# Targets that provide quick help on the build system.
include $(BUILD_SYSTEM)/help.mk

# Set up various standard variables based on configuration
# and host information.
include $(BUILD_SYSTEM)/config.mk

# This allows us to force a clean build - included after the config.make
# environment setup is done, but before we generate any dependencies.  This
# file does the rm -rf inline so the deps which are all done below will
# be generated correctly
#引入build/core/cleanbuild.mk
include $(BUILD_SYSTEM)/cleanbuild.mk

VERSION_CHECK_SEQUENCE_NUMBER := 2
-include $(OUT_DIR)/versions_checked.mk
ifneq ($(VERSION_CHECK_SEQUENCE_NUMBER),$(VERSIONS_CHECKED))

$(info Checking build tools versions...)

#如果是windows或者power pc则发出警告信息
ifneq ($(HOST_OS),windows)
ifneq ($(HOST_OS)-$(HOST_ARCH),darwin-ppc)
# check for a case sensitive file system
ifneq (a,$(shell mkdir -p $(OUT_DIR) ; \
                echo a > $(OUT_DIR)/casecheck.txt; \
                    echo B > $(OUT_DIR)/CaseCheck.txt; \
                cat $(OUT_DIR)/casecheck.txt))
$(warning ************************************************************)
$(warning You are building on a case-insensitive filesystem.)
$(warning Please move your source tree to a case-sensitive filesystem.)
$(warning ************************************************************)
$(error Case-insensitive filesystems not supported)
endif
endif
endif

#保证路径中没有空格
# Make sure that there are no spaces in the absolute path; the
# build system can't deal with them.
ifneq ($(words $(shell pwd)),1)
$(warning ************************************************************)
$(warning You are building in a directory whose absolute path contains)
$(warning a space character:)
$(warning $(space))
```

```
$(warning "$(shell pwd)")
$(warning $(space))
$(warning Please move your source tree to a path that does not contain)
$(warning any spaces.)
$(warning ************************************************************)
$(error Directory names containing spaces not supported)
endif

#检查java的版本，需要java1.6
# Check for the correct version of java
java_version := $(shell java -version 2>&1 | head -n 1 | grep '^java .*[ "]1\.6[
ifneq ($(shell java -version 2>&1 | grep -i openjdk),)
java_version :=
endif
ifeq ($(strip $(java_version)),)
$(info ************************************************************)
$(info You are attempting to build with an unsupported version)
$(info of java.)
$(info $(space))
$(info Your version is: $(shell java -version 2>&1 | head -n 1).)
$(info The correct version is: Java SE 1.6.)
$(info $(space))
$(info Please follow the machine setup instructions at)
$(info $(space)$(space)$(space)$(space)https://source.android.com/source/dow
$(info ************************************************************)
endif

#检查javac版本
# Check for the correct version of javac
javac_version := $(shell javac -version 2>&1 | head -n 1 | grep '[ "]1\.6[\. "$$
ifeq ($(strip $(javac_version)),)
$(info ************************************************************)
$(info You are attempting to build with the incorrect version)
$(info of javac.)
$(info $(space))
$(info Your version is: $(shell javac -version 2>&1 | head -n 1).)
$(info The correct version is: 1.6.)
$(info $(space))
$(info Please follow the machine setup instructions at)
$(info $(space)$(space)$(space)$(space)https://source.android.com/source/dow
$(info ************************************************************)
$(error stop)
endif

$(shell echo 'VERSIONS_CHECKED := $(VERSION_CHECK_SEQUENCE_NUMBER)' \
        > $(OUT_DIR)/versions_checked.mk)
```

```
endif

# These are the modifier targets that don't do anything themselves, but
# change the behavior of the build.
# (must be defined before including definitions.make)
INTERNAL_MODIFIER_TARGETS := showcommands checkbuild all incrementaljavac

.PHONY: incrementaljavac
incrementaljavac: ;

# WARNING:
# ENABLE_INCREMENTALJAVAC should NOT be enabled by default, because change of
# a Java source file won't trigger rebuild of its dependent Java files.
# You can only enable it by adding "incrementaljavac" to your make command line.
# You are responsible for the correctness of the incremental build.
# This may decrease incremental build time dramatically for large Java libraries
# such as core.jar, framework.jar, etc.
ENABLE_INCREMENTALJAVAC :=
ifneq (,$(filter incrementaljavac, $(MAKECMDGOALS)))
ENABLE_INCREMENTALJAVAC := true
MAKECMDGOALS := $(filter-out incrementaljavac, $(MAKECMDGOALS))
endif

#引入标准的系统定义
# Bring in standard build system definitions.
include $(BUILD_SYSTEM)/definitions.mk

#引入高通的帮助宏
# Bring in Qualcomm helper macros
include $(BUILD_SYSTEM)/qcom_utils.mk

#引入build/core/dex_preopt.mk
# Bring in dex_preopt.mk
include $(BUILD_SYSTEM)/dex_preopt.mk

#判断是eng,user还是userdebug
ifneq ($(filter eng user userdebug,$(MAKECMDGOALS)),)
$(info ************************************************************)
$(info ************************************************************)
$(info Don't pass '$(filter eng user userdebug tests,$(MAKECMDGOALS))' on \
the make command line.)
# XXX The single quote on this line fixes gvim's syntax highlighting.
# Without which, the rest of this file is impossible to read.
$(info Set TARGET_BUILD_VARIANT in buildspec.mk, or use lunch or)
$(info choosecombo.)
$(info ************************************************************)
```

```
$(info ************************************************************)
$(error stopping)
endif

ifneq ($(filter-out $(INTERNAL_VALID_VARIANTS),$(TARGET_BUILD_VARIANT)),)
$(info ************************************************************)
$(info ************************************************************)
$(info Invalid variant: $(TARGET_BUILD_VARIANT)
$(info Valid values are: $(INTERNAL_VALID_VARIANTS)
$(info ************************************************************)
$(info ************************************************************)
$(error stopping)
endif

# ---------------------------------------------------------------
# Variable to check java support level inside PDK build.
# Not necessary if the components is not in PDK.
# not defined : not supported
# "sdk" : sdk API only
# "platform" : platform API supproted
TARGET_BUILD_JAVA_SUPPORT_LEVEL := platform

# ---------------------------------------------------------------
# The pdk (Platform Development Kit) build
include build/core/pdk_config.mk

# ---------------------------------------------------------------
###
### In this section we set up the things that are different
### between the build variants
###

is_sdk_build :=
#判断是sdk,sdk_win_sdk还是sdk_addon
ifneq ($(filter sdk win_sdk sdk_addon,$(MAKECMDGOALS)),)
is_sdk_build := true
endif


## user/userdebug ##

user_variant := $(filter userdebug user,$(TARGET_BUILD_VARIANT))
enable_target_debugging := true
ifneq (,$(user_variant))
  # Target is secure in user builds.#如果是user版本，则ro.recure=1
  ADDITIONAL_DEFAULT_PROPERTIES += ro.secure=1
```

```
  tags_to_install := user
  ifeq ($(user_variant),userdebug)
    # Pick up some extra useful tools
    tags_to_install += debug

    # Enable Dalvik lock contention logging for userdebug builds.
    ADDITIONAL_BUILD_PROPERTIES += dalvik.vm.lockprof.threshold=500
  else
    # Disable debugging in plain user builds.
    enable_target_debugging :=
  endif

  # Turn on Dalvik preoptimization for user builds, but only if not
  # explicitly disabled and the build is running on Linux (since host
  # Dalvik isn't built for non-Linux hosts).
  ifneq (true,$(DISABLE_DEXPREOPT))
    ifeq ($(user_variant),user)
      ifeq ($(HOST_OS),linux)
        WITH_DEXPREOPT := true
      endif
    endif
  endif

  # Disallow mock locations by default for user builds
  ADDITIONAL_DEFAULT_PROPERTIES += ro.allow.mock.location=0

else # !user_variant
  # Turn on checkjni for non-user builds.
  ADDITIONAL_BUILD_PROPERTIES += ro.kernel.android.checkjni=1
  # Set device insecure for non-user builds.
  ADDITIONAL_DEFAULT_PROPERTIES += ro.secure=0
  # Allow mock locations by default for non user builds
  ADDITIONAL_DEFAULT_PROPERTIES += ro.allow.mock.location=1
endif # !user_variant

ifeq (true,$(strip $(enable_target_debugging)))
  # Target is more debuggable and adbd is on by default
  ADDITIONAL_DEFAULT_PROPERTIES += ro.debuggable=1
  # Include the debugging/testing OTA keys in this build.
  INCLUDE_TEST_OTA_KEYS := true
else # !enable_target_debugging
  # Target is less debuggable and adbd is off by default
  ADDITIONAL_DEFAULT_PROPERTIES += ro.debuggable=0
endif # !enable_target_debugging
```

```
## eng ##

ifeq ($(TARGET_BUILD_VARIANT),eng)
tags_to_install := user debug eng
ifneq ($(filter ro.setupwizard.mode=ENABLED, $(call collapse-pairs, $(ADDITIO
  # Don't require the setup wizard on eng builds
 ADDITIONAL_BUILD_PROPERTIES := $(filter-out ro.setupwizard.mode=%,\
         $(call collapse-pairs, $(ADDITIONAL_BUILD_PROPERTIES))) \
         ro.setupwizard.mode=OPTIONAL
endif
endif

## tests ##

ifeq ($(TARGET_BUILD_VARIANT),tests)
tags_to_install := user debug eng tests
endif

## sdk ##

ifdef is_sdk_build

# Detect if we want to build a repository for the SDK
sdk_repo_goal := $(strip $(filter sdk_repo,$(MAKECMDGOALS)))
MAKECMDGOALS := $(strip $(filter-out sdk_repo,$(MAKECMDGOALS)))

ifneq ($(words $(filter-out $(INTERNAL_MODIFIER_TARGETS),$(MAKECMDGOALS))),1
$(error The 'sdk' target may not be specified with any other targets)
endif

# TODO: this should be eng I think.  Since the sdk is built from the eng
# variant.
tags_to_install := user debug eng
ADDITIONAL_BUILD_PROPERTIES += xmpp.auto-presence=true
ADDITIONAL_BUILD_PROPERTIES += ro.config.nocheckin=yes
else # !sdk
endif

BUILD_WITHOUT_PV := true

## precise GC ##
#设定GC
ifneq ($(filter dalvik.gc.type-precise,$(PRODUCT_TAGS)),)
 # Enabling type-precise GC results in larger optimized DEX files.  The
 # additional storage requirements for ".odex" files can cause /system
  # to overflow on some devices, so this is configured separately for
```

```
  # each product.
  ADDITIONAL_BUILD_PROPERTIES += dalvik.vm.dexopt-flags=m=y
endif

#设定BT的name属性
ADDITIONAL_BUILD_PROPERTIES += net.bt.name=Android

# enable vm tracing in files for now to help track
# the cause of ANRs in the content process
ADDITIONAL_BUILD_PROPERTIES += dalvik.vm.stack-trace-file=/data/anr/traces.t

# ------------------------------------------------------------
# Define a function that, given a list of module tags, returns
# non-empty if that module should be installed in /system.

#定义不包含tests的tag的目标应该装入system
# For most goals, anything not tagged with the "tests" tag should
# be installed in /system.
define should-install-to-system
$(if $(filter tests,$(1)),,true)
endef

ifdef is_sdk_build
# For the sdk goal, anything with the "samples" tag should be
# installed in /data even if that module also has "eng"/"debug"/"user".
define should-install-to-system
$(if $(filter samples tests,$(1)),,true)
endef
endif


# If they only used the modifier goals (showcommands, checkbuild), we'll actual
# build the default target.
ifeq ($(filter-out $(INTERNAL_MODIFIER_TARGETS),$(MAKECMDGOALS)),)
.PHONY: $(INTERNAL_MODIFIER_TARGETS)
$(INTERNAL_MODIFIER_TARGETS): $(DEFAULT_GOAL)
endif

# These targets are going to delete stuff, don't bother including
# the whole directory tree if that's all we're going to do
ifeq ($(MAKECMDGOALS),clean)
dont_bother := true
endif
ifeq ($(MAKECMDGOALS),clobber)
dont_bother := true
endif
```

```
ifeq ($(MAKECMDGOALS),dataclean)
dont_bother := true
endif
ifeq ($(MAKECMDGOALS),installclean)
dont_bother := true
endif

# Bring in all modules that need to be built.
ifneq ($(dont_bother),true)

ifeq ($(HOST_OS)-$(HOST_ARCH),darwin-ppc)
SDK_ONLY := true
$(info Building the SDK under darwin-ppc is actually obsolete and unsupported.)
$(error stop)
endif

ifeq ($(HOST_OS),windows)
SDK_ONLY := true
endif

ifeq ($(SDK_ONLY),true)
include $(TOPDIR)sdk/build/sdk_only_whitelist.mk
include $(TOPDIR)development/build/sdk_only_whitelist.mk

# Exclude tools/acp when cross-compiling windows under linux
ifeq ($(findstring Linux,$(UNAME)),)
subdirs += build/tools/acp
endif

else # !SDK_ONLY
ifeq ($(BUILD_TINY_ANDROID), true)

# TINY_ANDROID is a super-minimal build configuration, handy for board
# bringup and very low level debugging

subdirs := \
bionic \
system/core \
system/extras/ext4_utils \
system/extras/su \
build/libs \
build/target \
build/tools/acp \
external/gcc-demangle \
external/mksh \
external/yaffs2 \
```

```
external/zlib
else # !BUILD_TINY_ANDROID
#
# Typical build; include any Android.mk files we can find.
#
subdirs := $(TOP)

FULL_BUILD := true

endif # !BUILD_TINY_ANDROID

endif # !SDK_ONLY

# Before we go and include all of the module makefiles, stash away
# the PRODUCT_* values so that later we can verify they are not modified.
stash_product_vars:=true
ifeq ($(stash_product_vars),true)
  $(call stash-product-vars, __STASHED)
endif

ifneq ($(ONE_SHOT_MAKEFILE),)
# We've probably been invoked by the "mm" shell function
# with a subdirectory's makefile.
include $(ONE_SHOT_MAKEFILE)
# Change CUSTOM_MODULES to include only modules that were
# defined by this makefile; this will install all of those
# modules as a side-effect.  Do this after including ONE_SHOT_MAKEFILE
# so that the modules will be installed in the same place they
# would have been with a normal make.
CUSTOM_MODULES := $(sort $(call get-tagged-modules,$(ALL_MODULE_TAGS)))
FULL_BUILD :=
# Stub out the notice targets, which probably aren't defined
# when using ONE_SHOT_MAKEFILE.
NOTICE-HOST-%: ;
NOTICE-TARGET-%: ;

else # ONE_SHOT_MAKEFILE

#
# Include all of the makefiles in the system
#

#找到所有的Android.mk并引入
# Can't use first-makefiles-under here because
# --mindepth=2 makes the prunes not work.
subdir_makefiles := \
```

```
$(shell build/tools/findleaves.py --prune=out --prune=.repo --prune=.git $(su

include $(subdir_makefiles)

endif # ONE_SHOT_MAKEFILE

# Now with all Android.mks loaded we can do post cleaning steps.
include $(BUILD_SYSTEM)/post_clean.mk

ifeq ($(stash_product_vars),true)
  $(call assert-product-vars, __STASHED)
endif

include $(BUILD_SYSTEM)/legacy_prebuilts.mk
ifneq ($(filter-out $(GRANDFATHERED_ALL_PREBUILT),$(strip $(notdir $(ALL_PREB
  $(warning *** Some files have been added to ALL_PREBUILT.)
  $(warning *)
  $(warning * ALL_PREBUILT is a deprecated mechanism that)
  $(warning * should not be used for new files.)
  $(warning * As an alternative, use PRODUCT_COPY_FILES in)
  $(warning * the appropriate product definition.)
  $(warning * build/target/product/core.mk is the product)
  $(warning * definition used in all products.)
  $(warning *)
 $(foreach bad_prebuilt,$(filter-out $(GRANDFATHERED_ALL_PREBUILT),$(strip $
  $(warning *)
  $(error ALL_PREBUILT contains unexpected files)
endif

# -------------------------------------------------------------------
# All module makefiles have been included at this point.
# -------------------------------------------------------------------

# -------------------------------------------------------------------
# Include any makefiles that must happen after the module makefiles
# have been included.
# TODO: have these files register themselves via a global var rather
# than hard-coding the list here.
ifdef FULL_BUILD
  # Only include this during a full build, otherwise we can't be
  # guaranteed that any policies were included.
  -include frameworks/policies/base/PolicyConfig.mk
endif

# -------------------------------------------------------------------
# Fix up CUSTOM_MODULES to refer to installed files rather than
```

```
# just bare module names.  Leave unknown modules alone in case
# they're actually full paths to a particular file.
known_custom_modules := $(filter $(ALL_MODULES),$(CUSTOM_MODULES))
unknown_custom_modules := $(filter-out $(ALL_MODULES),$(CUSTOM_MODULES))
CUSTOM_MODULES := \
$(call module-installed-files,$(known_custom_modules)) \
$(unknown_custom_modules)

# ------------------------------------------------------------------
# Define dependencies for modules that require other modules.
# This can only happen now, after we've read in all module makefiles.
#
# TODO: deal with the fact that a bare module name isn't
# unambiguous enough.  Maybe declare short targets like
# APPS:Quake or HOST:SHARED_LIBRARIES:libutils.
# BUG: the system image won't know to depend on modules that are
# brought in as requirements of other modules.
define add-required-deps
$(1): $(2)
endef
$(foreach m,$(ALL_MODULES), \
  $(eval r := $(ALL_MODULES.$(m).REQUIRED)) \
  $(if $(r), \
    $(eval r := $(call module-installed-files,$(r))) \
   $(eval $(call add-required-deps,$(ALL_MODULES.$(m).INSTALLED),$(r))) \
    ) \
 )
m :=
r :=
i :=
add-required-deps :=

# ------------------------------------------------------------------
# Figure out our module sets.

# Of the modules defined by the component makefiles,
# determine what we actually want to build.
Default_MODULES := $(sort $(ALL_DEFAULT_INSTALLED_MODULES) \
                          $(CUSTOM_MODULES))
# TODO: Remove the 3 places in the tree that use
# ALL_DEFAULT_INSTALLED_MODULES and get rid of it from this list.

ifdef FULL_BUILD
  # The base list of modules to build for this product is specified
  # by the appropriate product definition file, which was included
  # by product_config.make.
```

```
  user_PACKAGES := $(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_PACKAGES)
  $(call expand-required-modules,user_PACKAGES,$(user_PACKAGES))
  user_PACKAGES := $(call module-installed-files, $(user_PACKAGES))
else
  # We're not doing a full build, and are probably only including
  # a subset of the module makefiles.  Don't try to build any modules
  # requested by the product, because we probably won't have rules
  # to build them.
  user_PACKAGES :=
endif
# Use tags to get the non-APPS user modules.  Use the product
# definition files to get the APPS user modules.
user_MODULES := $(sort $(call get-tagged-modules,user shell_$(TARGET_SHELL)))
user_MODULES := $(user_MODULES) $(user_PACKAGES)

eng_MODULES := $(sort $(call get-tagged-modules,eng))
debug_MODULES := $(sort $(call get-tagged-modules,debug))
tests_MODULES := $(sort $(call get-tagged-modules,tests))

ifeq ($(strip $(tags_to_install)),)
$(error ASSERTION FAILED: tags_to_install should not be empty)
endif
modules_to_install := $(sort $(Default_MODULES) \
          $(foreach tag,$(tags_to_install),$($(tag)_MODULES)))

# Some packages may override others using LOCAL_OVERRIDES_PACKAGES.
# Filter out (do not install) any overridden packages.
overridden_packages := $(call get-package-overrides,$(modules_to_install))
ifdef overridden_packages
#  old_modules_to_install := $(modules_to_install)
  modules_to_install := \
    $(filter-out $(foreach p,$(overridden_packages),$(p) %/$(p).apk), \
          $(modules_to_install))
endif
#$(error filtered out
#       $(filter-out $(modules_to_install),$(old_modules_to_install)))

# Don't include any GNU targets in the SDK.  It's ok (and necessary)
# to build the host tools, but nothing that's going to be installed
# on the target (including static libraries).
ifdef is_sdk_build
  target_gnu_MODULES := \
              $(filter \
                      $(TARGET_OUT_INTERMEDIATES)/% \
                      $(TARGET_OUT)/% \
                      $(TARGET_OUT_DATA)/%, \
```

```
                                $(sort $(call get-tagged-modules,gnu)))
  $(info Removing from sdk:)$(foreach d,$(target_gnu_MODULES),$(info : $(d)))
   modules_to_install := \
           $(filter-out $(target_gnu_MODULES),$(modules_to_install))

  # Ensure every module listed in PRODUCT_PACKAGES gets something installed
   $(foreach m, $(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_PACKAGES), \
       $(if $(strip $(ALL_MODULES.$(m).INSTALLED)),,\
        $(error Module '$(m)' in PRODUCT_PACKAGES has nothing to install!)))
endif


# build/core/Makefile contains extra stuff that we don't want to pollute this
# top-level makefile with.  It expects that ALL_DEFAULT_INSTALLED_MODULES
# contains everything that's built during the current make, but it also further
# extends ALL_DEFAULT_INSTALLED_MODULES.
ALL_DEFAULT_INSTALLED_MODULES := $(modules_to_install)
include $(BUILD_SYSTEM)/Makefile
modules_to_install := $(sort $(ALL_DEFAULT_INSTALLED_MODULES))
ALL_DEFAULT_INSTALLED_MODULES :=

endif # dont_bother

# These are additional goals that we build, in order to make sure that there
# is as little code as possible in the tree that doesn't build.
modules_to_check := $(foreach m,$(ALL_MODULES),$(ALL_MODULES.$(m).CHECKED))

# If you would like to build all goals, and not skip any intermediate
# steps, you can pass the "all" modifier goal on the commandline.
ifneq ($(filter all,$(MAKECMDGOALS)),)
modules_to_check += $(foreach m,$(ALL_MODULES),$(ALL_MODULES.$(m).BUILT))
endif

# for easier debugging
modules_to_check := $(sort $(modules_to_check))
#$(error modules_to_check $(modules_to_check))

# ------------------------------------------------------------------
# This is used to to get the ordering right, you can also use these,
# but they're considered undocumented, so don't complain if their
# behavior changes.
.PHONY: prebuilt
prebuilt: $(ALL_PREBUILT)

# An internal target that depends on all copied headers
# (see copy_headers.make).  Other targets that need the
```

```
# headers to be copied first can depend on this target.
.PHONY: all_copied_headers
all_copied_headers: ;

$(ALL_C_CPP_ETC_OBJECTS): | all_copied_headers

# All the droid stuff, in directories
.PHONY: files
files: prebuilt \
        $(modules_to_install) \
        $(INSTALLED_ANDROID_INFO_TXT_TARGET)

# ------------------------------------------------------------------

.PHONY: checkbuild
checkbuild: $(modules_to_check)

.PHONY: ramdisk
ramdisk: $(INSTALLED_RAMDISK_TARGET)

.PHONY: factory_ramdisk
factory_ramdisk: $(INSTALLED_FACTORY_RAMDISK_TARGET)

.PHONY: systemtarball
systemtarball: $(INSTALLED_SYSTEMTARBALL_TARGET)

.PHONY: boottarball
boottarball: $(INSTALLED_BOOTTARBALL_TARGET)

.PHONY: userdataimage
userdataimage: $(INSTALLED_USERDATAIMAGE_TARGET)

ifneq (,$(filter userdataimage, $(MAKECMDGOALS)))
$(call dist-for-goals, userdataimage, $(BUILT_USERDATAIMAGE_TARGET))
endif

.PHONY: userdatatarball
userdatatarball: $(INSTALLED_USERDATATARBALL_TARGET)

.PHONY: cacheimage
cacheimage: $(INSTALLED_CACHEIMAGE_TARGET)

.PHONY: bootimage
bootimage: $(INSTALLED_BOOTIMAGE_TARGET)

ifeq ($(BUILD_TINY_ANDROID), true)
```

```
INSTALLED_RECOVERYIMAGE_TARGET :=
endif

#android的目标包括boot.img,recover.img,userdata.img,cache.img,system.img
等镜像。
# Build files and then package it into the rom formats
.PHONY: droidcore
droidcore: files \
systemimage \
$(INSTALLED_BOOTIMAGE_TARGET) \
$(INSTALLED_RECOVERYIMAGE_TARGET) \
$(INSTALLED_USERDATAIMAGE_TARGET) \
$(INSTALLED_CACHEIMAGE_TARGET) \
$(INSTALLED_FILES_FILE)

# dist_files only for putting your library into the dist directory with a full b
.PHONY: dist_files

ifeq ($(EMMA_INSTRUMENT),true)
  $(call dist-for-goals, dist_files, $(EMMA_META_ZIP))
endif

# Dist for droid if droid is among the cmd goals, or no cmd goal is given.
ifneq ($(filter droid,$(MAKECMDGOALS))$(filter ||,|$(filter-out $(INTERNAL_MO

ifneq ($(TARGET_BUILD_APPS),)
 # If this build is just for apps, only build apps and not the full system by def

  unbundled_build_modules :=
  ifneq ($(filter all,$(TARGET_BUILD_APPS)),)
   # If they used the magic goal "all" then build all apps in the source tree.
   unbundled_build_modules := $(foreach m,$(sort $(ALL_MODULES)),$(if $(filte
  else
    unbundled_build_modules := $(TARGET_BUILD_APPS)
  endif

  # dist the unbundled app.
  $(call dist-for-goals,apps_only, \
   $(foreach m,$(unbundled_build_modules),$(ALL_MODULES.$(m).INSTALLED)) \
  )

.PHONY: apps_only
apps_only: $(unbundled_build_modules)

droid: apps_only
```

```
else # TARGET_BUILD_APPS
  $(call dist-for-goals, droidcore, \
    $(INTERNAL_UPDATE_PACKAGE_TARGET) \
    $(INTERNAL_OTA_PACKAGE_TARGET) \
    $(SYMBOLS_ZIP) \
    $(INSTALLED_FILES_FILE) \
    $(INSTALLED_BUILD_PROP_TARGET) \
    $(BUILT_TARGET_FILES_PACKAGE) \
    $(INSTALLED_ANDROID_INFO_TXT_TARGET) \
    $(INSTALLED_RAMDISK_TARGET) \
    $(INSTALLED_FACTORY_RAMDISK_TARGET) \
   )

  ifneq ($(TARGET_BUILD_PDK),true)
    $(call dist-for-goals, droidcore, \
      $(APPS_ZIP) \
      $(INTERNAL_EMULATOR_PACKAGE_TARGET) \
      $(PACKAGE_STATS_FILE) \
    )
  endif

# Building a full system-- the default is to build droidcore
droid: droidcore dist_files

endif # TARGET_BUILD_APPS
endif # droid in $(MAKECMDGOALS)


.PHONY: droid

# phony target that include any targets in $(ALL_MODULES)
.PHONY: all_modules
all_modules: $(ALL_MODULES)

.PHONY: docs
docs: $(ALL_DOCS)

.PHONY: sdk
ALL_SDK_TARGETS := $(INTERNAL_SDK_TARGET)
sdk: $(ALL_SDK_TARGETS)
ifneq ($(filter sdk win_sdk,$(MAKECMDGOALS)),)
$(call dist-for-goals,sdk win_sdk, \
    $(ALL_SDK_TARGETS) \
    $(SYMBOLS_ZIP) \
    $(INSTALLED_BUILD_PROP_TARGET) \
)
```

```
endif

.PHONY: samplecode
sample_MODULES := $(sort $(call get-tagged-modules,samples))
sample_APKS_DEST_PATH := $(TARGET_COMMON_OUT_ROOT)/samples
sample_APKS_COLLECTION := \
     $(foreach module,$(sample_MODULES),$(sample_APKS_DEST_PATH)/$(notdir $(
$(foreach module,$(sample_MODULES),$(eval $(call \
     copy-one-file,$(module),$(sample_APKS_DEST_PATH)/$(notdir $(module))))
sample_ADDITIONAL_INSTALLED := \
     $(filter-out $(modules_to_install) $(modules_to_check) $(ALL_PREBUILT),
samplecode: $(sample_APKS_COLLECTION)
@echo -e ${CL_GRN}"Collect sample code apks:"${CL_RST}" $^"
# remove apks that are not intended to be installed.
rm -f $(sample_ADDITIONAL_INSTALLED)

.PHONY: findbugs
findbugs: $(INTERNAL_FINDBUGS_HTML_TARGET) $(INTERNAL_FINDBUGS_XML_TARGET)

.PHONY: clean
clean:
@rm -rf $(OUT_DIR)/*
@echo -e ${CL_GRN}"Entire build directory removed."${CL_RST}

.PHONY: clobber
clobber: clean

# The rules for dataclean and installclean are defined in cleanbuild.mk.

#xxx scrape this from ALL_MODULE_NAME_TAGS
.PHONY: modules
modules:
@echo -e ${CL_GRN}"Available sub-modules:"${CL_RST}
@echo "$(call module-names-for-tag-list,$(ALL_MODULE_TAGS))" | \
      tr -s ' ' '\n' | sort -u | $(COLUMN)

.PHONY: showcommands
showcommands:
@echo >/dev/null
```

6 Makefile

```
build/core/Makefile
# Put some miscellaneous rules here
```

```
# Build system colors

ifneq ($(BUILD_WITH_COLORS),0)
  CL_RED="\033[31m"
  CL_GRN="\033[32m"
  CL_YLW="\033[33m"
  CL_BLU="\033[34m"
  CL_MAG="\033[35m"
  CL_CYN="\033[36m"
  CL_RST="\033[0m"
endif

# Pick a reasonable string to use to identify files.
ifneq "" "$(filter eng.%,$(BUILD_NUMBER))"
  # BUILD_NUMBER has a timestamp in it, which means that
  # it will change every time.  Pick a stable value.
  FILE_NAME_TAG := eng.$(USER)
else
  FILE_NAME_TAG := $(BUILD_NUMBER)
endif

is_tests_build := $(filter tests,$(MAKECMDGOALS))

# ------------------------------------------------------------------
# Define rules to copy PRODUCT_COPY_FILES defined by the product.
# PRODUCT_COPY_FILES contains words like <source file>:<dest file>.
# <dest file> is relative to $(PRODUCT_OUT), so it should look like,
# e.g., "system/etc/file.xml".
# The filter part means "only eval the copy-one-file rule if this
# src:dest pair is the first one to match the same dest"
#$(1): the src:dest pair
#过滤APK
define check-product-copy-files
$(if $(filter %.apk, $(1)),$(error \
   Prebuilt apk found in PRODUCT_COPY_FILES: $(1), use BUILD_PREBUILT instead!
endef
# filter out the duplicate <source file>:<dest file> pairs.
unique_product_copy_files_pairs :=
$(foreach cf,$(PRODUCT_COPY_FILES), \
    $(if $(filter $(unique_product_copy_files_pairs),$(cf)),,\
        $(eval unique_product_copy_files_pairs += $(cf))))
unique_product_copy_files_destinations :=
$(foreach cf,$(unique_product_copy_files_pairs), \
    $(eval _src := $(call word-colon,1,$(cf))) \
    $(eval _dest := $(call word-colon,2,$(cf))) \
   $(if $(filter $(unique_product_copy_files_destinations),$(_dest)), \
```

```
        $(info PRODUCT_COPY_FILES $(cf) ignored.), \
     $(eval _fulldest := $(call append-path,$(PRODUCT_OUT),$(_dest))) \
        $(if $(filter %.xml,$(_dest)),\
        $(eval $(call copy-xml-file-checked,$(_src),$(_fulldest))),\
            $(eval $(call copy-one-file,$(_src),$(_fulldest)))) \
        $(eval ALL_DEFAULT_INSTALLED_MODULES += $(_fulldest)) \
       $(eval unique_product_copy_files_destinations += $(_dest))))
unique_product_copy_files_pairs :=
unique_product_copy_files_destinations :=

# ------------------------------------------------------------------
# docs/index.html
ifeq (,$(TARGET_BUILD_APPS))
gen := $(OUT_DOCS)/index.html
ALL_DOCS += $(gen)
$(gen): frameworks/base/docs/docs-redirect-index.html
@mkdir -p $(dir $@)
@cp -f $< $@
endif

# ------------------------------------------------------------------
# default.prop
INSTALLED_DEFAULT_PROP_TARGET := $(TARGET_ROOT_OUT)/default.prop
ALL_DEFAULT_INSTALLED_MODULES += $(INSTALLED_DEFAULT_PROP_TARGET)
ADDITIONAL_DEFAULT_PROPERTIES := \
    $(call collapse-pairs, $(ADDITIONAL_DEFAULT_PROPERTIES))
ADDITIONAL_DEFAULT_PROPERTIES += \
    $(call collapse-pairs, $(PRODUCT_DEFAULT_PROPERTY_OVERRIDES))
ADDITIONAL_DEFAULT_PROPERTIES := $(call uniq-pairs-by-first-component, \
    $(ADDITIONAL_DEFAULT_PROPERTIES),=)

$(INSTALLED_DEFAULT_PROP_TARGET):
@echo Target buildinfo: $@
@mkdir -p $(dir $@)
$(hide) echo "#" > $@; \
        echo "# ADDITIONAL_DEFAULT_PROPERTIES" >> $@; \
        echo "#" >> $@;
$(hide) $(foreach line,$(ADDITIONAL_DEFAULT_PROPERTIES), \
echo "$(line)" >> $@;)
build/tools/post_process_props.py $@

# ------------------------------------------------------------------
# build.prop
INSTALLED_BUILD_PROP_TARGET := $(TARGET_OUT)/build.prop
ALL_DEFAULT_INSTALLED_MODULES += $(INSTALLED_BUILD_PROP_TARGET)
ADDITIONAL_BUILD_PROPERTIES := \
```

```
    $(call collapse-pairs, $(ADDITIONAL_BUILD_PROPERTIES))
ADDITIONAL_BUILD_PROPERTIES := $(call uniq-pairs-by-first-component, \
    $(ADDITIONAL_BUILD_PROPERTIES),=)

# A list of arbitrary tags describing the build configuration.
# Force ":=" so we can use +=
BUILD_VERSION_TAGS := $(BUILD_VERSION_TAGS)
ifeq ($(TARGET_BUILD_TYPE),debug)
  BUILD_VERSION_TAGS += debug
endif
# The "test-keys" tag marks builds signed with the old test keys,
# which are available in the SDK.  "dev-keys" marks builds signed with
# non-default dev keys (usually private keys from a vendor directory).
# Both of these tags will be removed and replaced with "release-keys"
# when the target-files is signed in a post-build step.
ifeq ($(DEFAULT_SYSTEM_DEV_CERTIFICATE),build/target/product/security/testke
BUILD_VERSION_TAGS += test-keys
else
BUILD_VERSION_TAGS += dev-keys
endif
BUILD_VERSION_TAGS := $(subst $(space),$(comma),$(sort $(BUILD_VERSION_TAGS))

# A human-readable string that descibes this build in detail.
build_desc := $(TARGET_PRODUCT)-$(TARGET_BUILD_VARIANT) $(PLATFORM_VERSION) $
$(INSTALLED_BUILD_PROP_TARGET): PRIVATE_BUILD_DESC := $(build_desc)

# The string used to uniquely identify this build;  used by the OTA server.
ifeq (,$(strip $(BUILD_FINGERPRINT)))
 BUILD_FINGERPRINT := $(PRODUCT_BRAND)/$(TARGET_PRODUCT)/$(TARGET_DEVICE):$(
endif
ifneq ($(words $(BUILD_FINGERPRINT)),1)
 $(error BUILD_FINGERPRINT cannot contain spaces: "$(BUILD_FINGERPRINT)")
endif

# Display parameters shown under Settings -> About Phone
ifeq ($(TARGET_BUILD_VARIANT),user)
  # User builds should show:
  # release build number or branch.buld_number non-release builds

  # Dev. branches should have DISPLAY_BUILD_NUMBER set
  ifeq "true" "$(DISPLAY_BUILD_NUMBER)"
    BUILD_DISPLAY_ID := $(BUILD_ID).$(BUILD_NUMBER)
  else
    BUILD_DISPLAY_ID := $(BUILD_ID)
  endif
else
```

```
  # Non-user builds should show detailed build information
  BUILD_DISPLAY_ID := $(build_desc)
endif

# Whether there is default locale set in PRODUCT_PROPERTY_OVERRIDES
product_property_override_locale_language := $(strip \
    $(patsubst ro.product.locale.language=%,%,\
   $(filter ro.product.locale.language=%,$(PRODUCT_PROPERTY_OVERRIDES))))
product_property_overrides_locale_region := $(strip \
    $(patsubst ro.product.locale.region=%,%,\
   $(filter ro.product.locale.region=%,$(PRODUCT_PROPERTY_OVERRIDES))))

# Selects the first locale in the list given as the argument,
# and splits it into language and region, which each may be
# empty.
define default-locale
$(subst _, , $(firstword $(1)))
endef

# Selects the first locale in the list given as the argument
# and returns the language (or the region), if it's not set in PRODUCT_PROPERTY_
# Return empty string if it's already set in PRODUCT_PROPERTY_OVERRIDES.
define default-locale-language
$(if $(product_property_override_locale_language),,$(word 1, $(call default-l
endef
define default-locale-region
$(if $(product_property_overrides_locale_region),,$(word 2, $(call default-lo
endef

BUILDINFO_SH := build/tools/buildinfo.sh
$(INSTALLED_BUILD_PROP_TARGET): $(BUILDINFO_SH) $(INTERNAL_BUILD_ID_MAKEFILE
@echo Target buildinfo: $@
@mkdir -p $(dir $@)
$(hide) TARGET_BUILD_TYPE="$(TARGET_BUILD_VARIANT)" \
TARGET_DEVICE="$(TARGET_DEVICE)" \
CM_DEVICE="$(TARGET_DEVICE)" \
PRODUCT_NAME="$(TARGET_PRODUCT)" \
PRODUCT_BRAND="$(PRODUCT_BRAND)" \
PRODUCT_DEFAULT_LANGUAGE="$(call default-locale-language,$(PRODUCT_LOCALES))
PRODUCT_DEFAULT_REGION="$(call default-locale-region,$(PRODUCT_LOCALES))" \
PRODUCT_DEFAULT_WIFI_CHANNELS="$(PRODUCT_DEFAULT_WIFI_CHANNELS)" \
PRODUCT_MODEL="$(PRODUCT_MODEL)" \
PRODUCT_MANUFACTURER="$(PRODUCT_MANUFACTURER)" \
PRIVATE_BUILD_DESC="$(PRIVATE_BUILD_DESC)" \
BUILD_ID="$(BUILD_ID)" \
BUILD_DISPLAY_ID="$(BUILD_DISPLAY_ID)" \
```

```
BUILD_NUMBER="$(BUILD_NUMBER)" \
PLATFORM_VERSION="$(PLATFORM_VERSION)" \
PLATFORM_SDK_VERSION="$(PLATFORM_SDK_VERSION)" \
PLATFORM_VERSION_CODENAME="$(PLATFORM_VERSION_CODENAME)" \
BUILD_VERSION_TAGS="$(BUILD_VERSION_TAGS)" \
TARGET_BOOTLOADER_BOARD_NAME="$(TARGET_BOOTLOADER_BOARD_NAME)" \
BUILD_FINGERPRINT="$(BUILD_FINGERPRINT)" \
TARGET_BOARD_PLATFORM="$(TARGET_BOARD_PLATFORM)" \
TARGET_CPU_ABI="$(TARGET_CPU_ABI)" \
TARGET_CPU_ABI2="$(TARGET_CPU_ABI2)" \
TARGET_AAPT_CHARACTERISTICS="$(TARGET_AAPT_CHARACTERISTICS)" \
$(PRODUCT_BUILD_PROP_OVERRIDES) \
        bash $(BUILDINFO_SH) > $@
$(hide) if [ -f $(TARGET_DEVICE_DIR)/system.prop ]; then \
          cat $(TARGET_DEVICE_DIR)/system.prop >> $@; \
        fi
$(if $(ADDITIONAL_BUILD_PROPERTIES), \
$(hide) echo >> $@; \
        echo "#" >> $@; \
        echo "# ADDITIONAL_BUILD_PROPERTIES" >> $@; \
        echo "#" >> $@; )
$(hide) $(foreach line,$(ADDITIONAL_BUILD_PROPERTIES), \
echo "$(line)" >> $@;)
$(hide) build/tools/post_process_props.py $@

build_desc :=

# ---------------------------------------------------------------
# sdk-build.prop
#
# There are certain things in build.prop that we don't want to
# ship with the sdk; remove them.

# This must be a list of entire property keys followed by
# "=" characters, without any internal spaces.
sdk_build_prop_remove := \
ro.build.user= \
ro.build.host= \
ro.product.brand= \
ro.product.manufacturer= \
ro.product.device=
# TODO: Remove this soon-to-be obsolete property
sdk_build_prop_remove += ro.build.product=
INSTALLED_SDK_BUILD_PROP_TARGET := $(PRODUCT_OUT)/sdk/sdk-build.prop
$(INSTALLED_SDK_BUILD_PROP_TARGET): $(INSTALLED_BUILD_PROP_TARGET)
@echo SDK buildinfo: $@
```

```
@mkdir -p $(dir $@)
$(hide) grep -v "$(subst $(space),\|,$(strip \
$(sdk_build_prop_remove)))" $< > $@.tmp
$(hide) for x in $(sdk_build_prop_remove); do \
echo "$$x"generic >> $@.tmp; done
$(hide) mv $@.tmp $@

# ------------------------------------------------------------------
# package stats
PACKAGE_STATS_FILE := $(PRODUCT_OUT)/package-stats.txt
PACKAGES_TO_STAT := \
    $(sort $(filter $(TARGET_OUT)/% $(TARGET_OUT_DATA)/%, \
$(filter %.jar %.apk, $(ALL_DEFAULT_INSTALLED_MODULES))))
$(PACKAGE_STATS_FILE): $(PACKAGES_TO_STAT)
@echo Package stats: $@
@mkdir -p $(dir $@)
$(hide) rm -f $@
$(hide) build/tools/dump-package-stats $^ > $@

.PHONY: package-stats
package-stats: $(PACKAGE_STATS_FILE)

# ------------------------------------------------------------------
# Cert-to-package mapping.  Used by the post-build signing tools.
# Use a macro to add newline to each echo command
define _apkcerts_echo_with_newline
$(hide) echo $(1)

endef

name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-apkcerts-$(FILE_NAME_TAG)
intermediates := \
$(call intermediates-dir-for,PACKAGING,apkcerts)
APKCERTS_FILE := $(intermediates)/$(name).txt
# We don't need to really build all the modules.
# TODO: rebuild APKCERTS_FILE if any app change its cert.
$(APKCERTS_FILE):
@echo APK certs list: $@
@mkdir -p $(dir $@)
@rm -f $@
$(foreach p,$(PACKAGES),\
  $(if $(PACKAGES.$(p).EXTERNAL_KEY),\
```

```
    $(call _apkcerts_echo_with_newline,\
      'name="$(p).apk" certificate="EXTERNAL" \
      private_key=""' >> $@),\
    $(call _apkcerts_echo_with_newline,\
      'name="$(p).apk" certificate="$(PACKAGES.$(p).CERTIFICATE)" \
      private_key="$(PACKAGES.$(p).PRIVATE_KEY)"' >> $@)))
# In case value of PACKAGES is empty.
$(hide) touch $@

.PHONY: apkcerts-list
apkcerts-list: $(APKCERTS_FILE)

ifneq (,$(TARGET_BUILD_APPS))
  $(call dist-for-goals, apps_only, $(APKCERTS_FILE):apkcerts.txt)
endif

# -----------------------------------------------------------------
# module info file
ifdef CREATE_MODULE_INFO_FILE
  MODULE_INFO_FILE := $(PRODUCT_OUT)/module-info.txt
  $(info Generating $(MODULE_INFO_FILE)...)
  $(shell rm -f $(MODULE_INFO_FILE))
  $(foreach m,$(ALL_MODULES), \
    $(shell echo "NAME=\"$(m)\"" \
"PATH=\"$(strip $(ALL_MODULES.$(m).PATH))\"" \
"TAGS=\"$(strip $(filter-out _%,$(ALL_MODULES.$(m).TAGS)))\"" \
"BUILT=\"$(strip $(ALL_MODULES.$(m).BUILT))\"" \
"INSTALLED=\"$(strip $(ALL_MODULES.$(m).INSTALLED))\"" >> $(MODULE_INFO_FILE
endif

# -----------------------------------------------------------------

# The dev key is used to sign this package, and as the key required
# for future OTA packages installed by this system.  Actual product
# deliverables will be re-signed by hand.  We expect this file to
# exist with the suffixes ".x509.pem" and ".pk8".
DEFAULT_KEY_CERT_PAIR := $(DEFAULT_SYSTEM_DEV_CERTIFICATE)


# Rules that need to be present for the all targets, even
# if they don't do anything.
.PHONY: systemimage
systemimage:

# -----------------------------------------------------------------
```

```
.PHONY: event-log-tags

# Produce an event logs tag file for everything we know about, in order
# to properly allocate numbers.  Then produce a file that's filtered
# for what's going to be installed.

all_event_log_tags_file := $(TARGET_OUT_COMMON_INTERMEDIATES)/all-event-log-

event_log_tags_file := $(TARGET_OUT)/etc/event-log-tags

# Include tags from all packages that we know about
all_event_log_tags_src := \
   $(sort $(foreach m, $(ALL_MODULES), $(ALL_MODULES.$(m).EVENT_LOG_TAGS)))

# PDK builds will already have a full list of tags that needs to get merged
# in with the ones from source
pdk_fusion_log_tags_file := $(patsubst $(PRODUCT_OUT)/%,$(_pdk_fusion_interm

$(all_event_log_tags_file): PRIVATE_SRC_FILES := $(all_event_log_tags_src) $(
$(all_event_log_tags_file): $(all_event_log_tags_src) $(pdk_fusion_log_tags_
$(hide) mkdir -p $(dir $@)
$(hide) build/tools/merge-event-log-tags.py -o $@ $(PRIVATE_SRC_FILES)

# Include tags from all packages included in this product, plus all
# tags that are part of the system (ie, not in a vendor/ or device/
# directory).
event_log_tags_src := \
    $(sort $(foreach m,\
      $(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_PACKAGES) \
      $(call module-names-for-tag-list,user), \
      $(ALL_MODULES.$(m).EVENT_LOG_TAGS)) \
    $(filter-out vendor/% device/% out/%,$(all_event_log_tags_src)))

$(event_log_tags_file): PRIVATE_SRC_FILES := $(event_log_tags_src) $(pdk_fusi
$(event_log_tags_file): PRIVATE_MERGED_FILE := $(all_event_log_tags_file)
$(event_log_tags_file): $(event_log_tags_src) $(all_event_log_tags_file) $(p
$(hide) mkdir -p $(dir $@)
$(hide) build/tools/merge-event-log-tags.py -o $@ -m $(PRIVATE_MERGED_FILE) $

event-log-tags: $(event_log_tags_file)

ALL_DEFAULT_INSTALLED_MODULES += $(event_log_tags_file)


# ####################################################################
# Targets for boot/OS images
```

```
# ###############################################################

# ---------------------------------------------------------------
# the ramdisk
INTERNAL_RAMDISK_FILES := $(filter $(TARGET_ROOT_OUT)/%, \
$(ALL_PREBUILT) \
$(ALL_COPIED_HEADERS) \
$(ALL_GENERATED_SOURCES) \
$(ALL_DEFAULT_INSTALLED_MODULES))

BUILT_RAMDISK_TARGET := $(PRODUCT_OUT)/ramdisk.img

ifeq ($(HAVE_SELINUX),true)
SELINUX_DEPENDS := sepolicy file_contexts seapp_contexts
endif

# We just build this directly to the install location.
INSTALLED_RAMDISK_TARGET := $(BUILT_RAMDISK_TARGET)
$(INSTALLED_RAMDISK_TARGET): $(MKBOOTFS) $(INTERNAL_RAMDISK_FILES) $(SELINUX
$(call pretty,"Target ram disk: $@")
$(hide) $(MKBOOTFS) $(TARGET_ROOT_OUT) | $(MINIGZIP) > $@

ifneq ($(strip $(TARGET_NO_KERNEL)),true)

# ---------------------------------------------------------------
# the boot image, which is a collection of other images.
INTERNAL_BOOTIMAGE_ARGS := \
$(addprefix --second ,$(INSTALLED_2NDBOOTLOADER_TARGET)) \
--kernel $(INSTALLED_KERNEL_TARGET) \
--ramdisk $(INSTALLED_RAMDISK_TARGET)

INTERNAL_BOOTIMAGE_FILES := $(filter-out --%,$(INTERNAL_BOOTIMAGE_ARGS))

BOARD_KERNEL_CMDLINE := $(strip $(BOARD_KERNEL_CMDLINE))
ifdef BOARD_KERNEL_CMDLINE
  INTERNAL_BOOTIMAGE_ARGS += --cmdline "$(BOARD_KERNEL_CMDLINE)"
endif

BOARD_KERNEL_BASE := $(strip $(BOARD_KERNEL_BASE))
ifdef BOARD_KERNEL_BASE
  INTERNAL_BOOTIMAGE_ARGS += --base $(BOARD_KERNEL_BASE)
endif

BOARD_KERNEL_PAGESIZE := $(strip $(BOARD_KERNEL_PAGESIZE))
ifdef BOARD_KERNEL_PAGESIZE
  INTERNAL_BOOTIMAGE_ARGS += --pagesize $(BOARD_KERNEL_PAGESIZE)
```

```
endif

BOARD_FORCE_RAMDISK_ADDRESS := $(strip $(BOARD_FORCE_RAMDISK_ADDRESS))
ifneq ($(BOARD_FORCE_RAMDISK_ADDRESS),)
  INTERNAL_BOOTIMAGE_ARGS += --ramdiskaddr $(BOARD_FORCE_RAMDISK_ADDRESS)
endif

INSTALLED_BOOTIMAGE_TARGET := $(PRODUCT_OUT)/boot.img

ifeq ($(TARGET_BOOTIMAGE_USE_EXT2),true)
tmp_dir_for_image := $(call intermediates-dir-for,EXECUTABLES,boot_img)/boot
INTERNAL_BOOTIMAGE_ARGS += --tmpdir $(tmp_dir_for_image)
INTERNAL_BOOTIMAGE_ARGS += --genext2fs $(MKEXT2IMG)
$(INSTALLED_BOOTIMAGE_TARGET): $(MKEXT2IMG) $(INTERNAL_BOOTIMAGE_FILES)
$(call pretty,"Target boot image: $@")
$(hide) $(MKEXT2BOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $@
@echo -e ${CL_CYN}"Made boot image: $@"${CL_RST}

else ifndef BOARD_CUSTOM_BOOTIMG_MK # TARGET_BOOTIMAGE_USE_EXT2 != true

$(INSTALLED_BOOTIMAGE_TARGET): $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_FILES)
$(call pretty,"Target boot image: $@")
$(hide) $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $@
$(hide) $(call assert-max-image-size,$@,$(BOARD_BOOTIMAGE_PARTITION_SIZE),ra
@echo -e ${CL_CYN}"Made boot image: $@"${CL_RST}

endif # ifndef BOARD_CUSTOM_BOOTIMG_MK

else # TARGET_NO_KERNEL
# HACK: The top-level targets depend on the bootimage.  Not all targets
# can produce a bootimage, though, and emulator targets need the ramdisk
# instead.  Fake it out by calling the ramdisk the bootimage.
# TODO: make the emulator use bootimages, and make mkbootimg accept
#       kernel-less inputs.
INSTALLED_BOOTIMAGE_TARGET := $(INSTALLED_RAMDISK_TARGET)
endif

# ------------------------------------------------------------------
# NOTICE files
#
# We are required to publish the licenses for all code under BSD, GPL and
# Apache licenses (and possibly other more exotic ones as well). We err on the
# side of caution, so the licenses for other third-party code are included here
# too.
#
# This needs to be before the systemimage rules, because it adds to
```

```
# ALL_DEFAULT_INSTALLED_MODULES, which those use to pick which files
# go into the systemimage.

.PHONY: notice_files

# Create the rule to combine the files into text and html forms
# $(1) - Plain text output file
# $(2) - HTML output file
# $(3) - File title
# $(4) - Directory to use.  Notice files are all $(4)/src.  Other
#  directories in there will be used for scratch
# $(5) - Dependencies for the output files
#
# The algorithm here is that we go collect a hash for each of the notice
# files and write the names of the files that match that hash.  Then
# to generate the real files, we go print out all of the files and their
# hashes.
#
# These rules are fairly complex, so they depend on this makefile so if
# it changes, they'll run again.
#
# TODO: We could clean this up so that we just record the locations of the
# original notice files instead of making rules to copy them somwehere.
# Then we could traverse that without quite as much bash drama.
define combine-notice-files
$(1) $(2): PRIVATE_MESSAGE := $(3)
$(1) $(2): PRIVATE_DIR := $(4)
$(1) : $(2)
$(2) : $(5) $(BUILD_SYSTEM)/Makefile build/tools/generate-notice-files.py
build/tools/generate-notice-files.py $(1) $(2) $$(PRIVATE_MESSAGE) $$(PRIVATE
notice_files: $(1) $(2)
endef

# TODO These intermediate NOTICE.txt/NOTICE.html files should go into
# TARGET_OUT_NOTICE_FILES now that the notice files are gathered from
# the src subdirectory.

target_notice_file_txt := $(TARGET_OUT_INTERMEDIATES)/NOTICE.txt
target_notice_file_html := $(TARGET_OUT_INTERMEDIATES)/NOTICE.html
target_notice_file_html_gz := $(TARGET_OUT_INTERMEDIATES)/NOTICE.html.gz
tools_notice_file_txt := $(HOST_OUT_INTERMEDIATES)/NOTICE.txt
tools_notice_file_html := $(HOST_OUT_INTERMEDIATES)/NOTICE.html

kernel_notice_file := $(TARGET_OUT_NOTICE_FILES)/src/kernel.txt
pdk_fusion_notice_files := $(filter $(TARGET_OUT_NOTICE_FILES)/%, $(ALL_PDK_F
```

```
$(eval $(call combine-notice-files, \
$(target_notice_file_txt), \
$(target_notice_file_html), \
"Notices for files contained in the filesystem images in this directory:", \
$(TARGET_OUT_NOTICE_FILES), \
$(ALL_DEFAULT_INSTALLED_MODULES) $(kernel_notice_file) $(pdk_fusion_notice_f

$(eval $(call combine-notice-files, \
$(tools_notice_file_txt), \
$(tools_notice_file_html), \
"Notices for files contained in the tools directory:", \
$(HOST_OUT_NOTICE_FILES), \
$(ALL_DEFAULT_INSTALLED_MODULES)))

# Install the html file at /system/etc/NOTICE.html.gz.
# This is not ideal, but this is very late in the game, after a lot of
# the module processing has already been done -- in fact, we used the
# fact that all that has been done to get the list of modules that we
# need notice files for.
$(target_notice_file_html_gz): $(target_notice_file_html) | $(MINIGZIP)
$(hide) $(MINIGZIP) -9 < $< > $@
installed_notice_html_gz := $(TARGET_OUT)/etc/NOTICE.html.gz
$(installed_notice_html_gz): $(target_notice_file_html_gz) | $(ACP)
$(copy-file-to-target)

# if we've been run my mm, mmm, etc, don't reinstall this every time
ifeq ($(ONE_SHOT_MAKEFILE),)
ALL_DEFAULT_INSTALLED_MODULES += $(installed_notice_html_gz)
endif

# The kernel isn't really a module, so to get its module file in there, we
# make the target NOTICE files depend on this particular file too, which will
# then be in the right directory for the find in combine-notice-files to work.
$(kernel_notice_file): \
    prebuilts/qemu-kernel/arm/LINUX_KERNEL_COPYING \
    | $(ACP)
@echo -e ${CL_CYN}"Copying:"${CL_RST}" $@"
$(hide) mkdir -p $(dir $@)
$(hide) $(ACP) $< $@


# ------------------------------------------------------------------
# Build a keystore with the authorized keys in it, used to verify the
# authenticity of downloaded OTA packages.
#
# This rule adds to ALL_DEFAULT_INSTALLED_MODULES, so it needs to come
```

```
# before the rules that use that variable to build the image.
ALL_DEFAULT_INSTALLED_MODULES += $(TARGET_OUT_ETC)/security/otacerts.zip
$(TARGET_OUT_ETC)/security/otacerts.zip: KEY_CERT_PAIR := $(DEFAULT_KEY_CERT
$(TARGET_OUT_ETC)/security/otacerts.zip: $(addsuffix .x509.pem,$(DEFAULT_KEY
$(hide) rm -f $@
$(hide) mkdir -p $(dir $@)
$(hide) zip -qj $@ $<

.PHONY: otacerts
otacerts: $(TARGET_OUT_ETC)/security/otacerts.zip


# ###################################################################
# Targets for user images
# ###################################################################

INTERNAL_USERIMAGES_EXT_VARIANT :=
ifeq ($(TARGET_USERIMAGES_USE_EXT2),true)
INTERNAL_USERIMAGES_USE_EXT := true
INTERNAL_USERIMAGES_EXT_VARIANT := ext2
else
ifeq ($(TARGET_USERIMAGES_USE_EXT3),true)
INTERNAL_USERIMAGES_USE_EXT := true
INTERNAL_USERIMAGES_EXT_VARIANT := ext3
else
ifeq ($(TARGET_USERIMAGES_USE_EXT4),true)
INTERNAL_USERIMAGES_USE_EXT := true
INTERNAL_USERIMAGES_EXT_VARIANT := ext4
endif
endif
endif

ifneq (true,$(TARGET_USERIMAGES_SPARSE_EXT_DISABLED))
  INTERNAL_USERIMAGES_SPARSE_EXT_FLAG := -s
endif

ifeq ($(INTERNAL_USERIMAGES_USE_EXT),true)
INTERNAL_USERIMAGES_DEPS := $(MKEXTUSERIMG) $(MAKE_EXT4FS)
else
INTERNAL_USERIMAGES_DEPS := $(MKYAFFS2)
endif
INTERNAL_USERIMAGES_BINARY_PATHS := $(sort $(dir $(INTERNAL_USERIMAGES_DEPS)

# $(1): the path of the output dictionary file
define generate-userimage-prop-dictionary
$(if $(INTERNAL_USERIMAGES_EXT_VARIANT),$(hide) echo "fs_type=$(INTERNAL_USE
```

```
$(if $(BOARD_SYSTEMIMAGE_PARTITION_SIZE),$(hide) echo "system_size=$(BOARD_S
$(if $(BOARD_USERDATAIMAGE_PARTITION_SIZE),$(hide) echo "userdata_size=$(BOA
$(if $(BOARD_CACHEIMAGE_FILE_SYSTEM_TYPE),$(hide) echo "cache_fs_type=$(BOAR
$(if $(BOARD_CACHEIMAGE_PARTITION_SIZE),$(hide) echo "cache_size=$(BOARD_CAC
$(if $(INTERNAL_USERIMAGES_SPARSE_EXT_FLAG),$(hide) echo "extfs_sparse_flag=
$(if $(mkyaffs2_extra_flags),$(hide) echo "mkyaffs2_extra_flags=$(mkyaffs2_e
$(if $(filter true, $(strip $(HAVE_SELINUX))), echo "selinux_fc=$(TARGET_ROOT
endef

# ---------------------------------------------------------------
# Utility executables

INTERNAL_UTILITY_FILES := $(filter $(PRODUCT_OUT)/utilities/%, \
$(foreach module, $(ALL_MODULES), $(ALL_MODULES.$(module).INSTALLED)) \
$(ALL_DEFAULT_INSTALLED_MODULES))

.PHONY: utilities
utilities: $(INTERNAL_UTILITY_FILES)

# ---------------------------------------------------------------
# Recovery image

# If neither TARGET_NO_KERNEL nor TARGET_NO_RECOVERY are true
ifeq (,$(filter true, $(TARGET_NO_KERNEL) $(TARGET_NO_RECOVERY) $(BUILD_TINY_

INSTALLED_RECOVERYIMAGE_TARGET := $(PRODUCT_OUT)/recovery.img

ifneq ($(TARGET_RECOVERY_INITRC),)
  recovery_initrc := $(TARGET_RECOVERY_INITRC) # Use target specific init.rc
else
ifeq ($(BOARD_USES_RECOVERY_CHARGEMODE),true)
  $(error BOARD_USES_RECOVERY_CHARGEMODE is deprecated. Please see http://bit.
else
   recovery_initrc := $(call include-path-for, recovery)/etc/init.rc
endif
endif
ifneq ($(TARGET_PREBUILT_RECOVERY_KERNEL),)
  recovery_kernel := $(TARGET_PREBUILT_RECOVERY_KERNEL) # Use prebuilt recover
else
  recovery_kernel := $(INSTALLED_KERNEL_TARGET) # same as a non-recovery syste
endif
recovery_uncompressed_ramdisk := $(PRODUCT_OUT)/ramdisk-recovery.cpio
recovery_ramdisk := $(PRODUCT_OUT)/ramdisk-recovery.img
recovery_build_prop := $(INSTALLED_BUILD_PROP_TARGET)
recovery_binary := $(call intermediates-dir-for,EXECUTABLES,recovery)/recove
recovery_resources_common := $(call include-path-for, recovery)/res
```

```
recovery_resources_private := $(strip $(wildcard $(TARGET_DEVICE_DIR)/recover
recovery_root_private := $(strip $(wildcard $(TARGET_DEVICE_DIR)/recovery/roo
recovery_resource_deps := $(shell find $(recovery_resources_common) \
  $(recovery_resources_private) $(recovery_root_private) -type f)

ifneq ($(TARGET_RECOVERY_FSTAB),)
  recovery_fstab := $(strip $(wildcard $(TARGET_RECOVERY_FSTAB)))
else
 recovery_fstab := $(strip $(wildcard $(TARGET_DEVICE_DIR)/recovery.fstab))
endif

ifeq ($(recovery_resources_private),)
 $(info No private recovery resources for TARGET_DEVICE $(TARGET_DEVICE))
endif

ifeq ($(recovery_fstab),)
  $(info No recovery.fstab for TARGET_DEVICE $(TARGET_DEVICE))
endif

INTERNAL_RECOVERYIMAGE_ARGS := \
$(addprefix --second ,$(INSTALLED_2NDBOOTLOADER_TARGET)) \
--kernel $(recovery_kernel) \
--ramdisk $(recovery_ramdisk)

# Assumes this has already been stripped
ifdef BOARD_KERNEL_CMDLINE
  INTERNAL_RECOVERYIMAGE_ARGS += --cmdline "$(BOARD_KERNEL_CMDLINE)"
endif
ifdef BOARD_KERNEL_BASE
  INTERNAL_RECOVERYIMAGE_ARGS += --base $(BOARD_KERNEL_BASE)
endif
BOARD_KERNEL_PAGESIZE := $(strip $(BOARD_KERNEL_PAGESIZE))
ifdef BOARD_KERNEL_PAGESIZE
  INTERNAL_RECOVERYIMAGE_ARGS += --pagesize $(BOARD_KERNEL_PAGESIZE)
endif
ifneq ($(BOARD_FORCE_RAMDISK_ADDRESS),)
 INTERNAL_RECOVERYIMAGE_ARGS += --ramdiskaddr $(BOARD_FORCE_RAMDISK_ADDRESS)
endif
INTERNAL_RECOVERY_FILES := $(filter $(TARGET_RECOVERY_OUT)/%, \
$(foreach module, $(ALL_MODULES), $(ALL_MODULES.$(module).INSTALLED)) \
$(ALL_DEFAULT_INSTALLED_MODULES))

# Keys authorized to sign OTA packages this build will accept.  The
# build always uses dev-keys for this; release packaging tools will
# substitute other keys for this one.
OTA_PUBLIC_KEYS := $(DEFAULT_SYSTEM_DEV_CERTIFICATE).x509.pem
```

```
# Generate a file containing the keys that will be read by the
# recovery binary.
RECOVERY_INSTALL_OTA_KEYS := \
$(call intermediates-dir-for,PACKAGING,ota_keys)/keys
DUMPKEY_JAR := $(HOST_OUT_JAVA_LIBRARIES)/dumpkey.jar
$(RECOVERY_INSTALL_OTA_KEYS): PRIVATE_OTA_PUBLIC_KEYS := $(OTA_PUBLIC_KEYS)
$(RECOVERY_INSTALL_OTA_KEYS): extra_keys := $(patsubst %,%.x509.pem,$(PRODUCT
$(RECOVERY_INSTALL_OTA_KEYS): $(OTA_PUBLIC_KEYS) $(DUMPKEY_JAR) $(extra_keys
@echo "DumpPublicKey: $@ <= $(PRIVATE_OTA_PUBLIC_KEYS) $(extra_keys)"
@rm -rf $@
@mkdir -p $(dir $@)
java -jar $(DUMPKEY_JAR) $(PRIVATE_OTA_PUBLIC_KEYS) $(extra_keys) > $@

TARGET_RECOVERY_ROOT_TIMESTAMP := $(TARGET_RECOVERY_OUT)/root.ts

$(TARGET_RECOVERY_ROOT_TIMESTAMP): $(INTERNAL_RECOVERY_FILES) \
$(INSTALLED_RAMDISK_TARGET) \
$(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_FILES) \
$(recovery_binary) \
$(recovery_initrc) \
$(INSTALLED_2NDBOOTLOADER_TARGET) \
$(recovery_build_prop) $(recovery_resource_deps) \
$(recovery_fstab) \
$(RECOVERY_INSTALL_OTA_KEYS)
@echo -e ${CL_CYN}"----- Making recovery filesystem ------"${CL_RST}
mkdir -p $(TARGET_RECOVERY_OUT)
mkdir -p $(TARGET_RECOVERY_ROOT_OUT)
mkdir -p $(TARGET_RECOVERY_ROOT_OUT)/etc
mkdir -p $(TARGET_RECOVERY_ROOT_OUT)/tmp
@echo -e ${CL_CYN}"Copying baseline ramdisk..."${CL_RST}
cp -R $(TARGET_ROOT_OUT) $(TARGET_RECOVERY_OUT)
rm $(TARGET_RECOVERY_ROOT_OUT)/init*.rc
@echo -e ${CL_CYN}"Modifying ramdisk contents..."${CL_RST}
cp -f $(recovery_initrc) $(TARGET_RECOVERY_ROOT_OUT)/init.rc
cp -f $(recovery_binary) $(TARGET_RECOVERY_ROOT_OUT)/sbin/
rm -f $(TARGET_RECOVERY_ROOT_OUT)/init.*.rc
mkdir -p $(TARGET_RECOVERY_ROOT_OUT)/system/bin
cp -rf $(recovery_resources_common) $(TARGET_RECOVERY_ROOT_OUT)/
$(foreach item,$(recovery_resources_private), \
  cp -rf $(item) $(TARGET_RECOVERY_ROOT_OUT)/)
$(foreach item,$(recovery_root_private), \
  cp -rf $(item) $(TARGET_RECOVERY_OUT)/)
$(foreach item,$(recovery_fstab), \
  cp -f $(item) $(TARGET_RECOVERY_ROOT_OUT)/etc/recovery.fstab)
cp $(RECOVERY_INSTALL_OTA_KEYS) $(TARGET_RECOVERY_ROOT_OUT)/res/keys
```

```
cat $(INSTALLED_DEFAULT_PROP_TARGET) $(recovery_build_prop) \
        > $(TARGET_RECOVERY_ROOT_OUT)/default.prop
@echo -e ${CL_YLW}"Modifying default.prop"${CL_RST}
sed -i 's/ro.build.date.utc=.*/ro.build.date.utc=0/g' $(TARGET_RECOVERY_ROOT
@echo -e ${CL_CYN}"----- Made recovery filesystem --------"$(TARGET_RECOVERY_
@touch $(TARGET_RECOVERY_ROOT_TIMESTAMP)

$(recovery_uncompressed_ramdisk): $(MINIGZIP) \
    $(TARGET_RECOVERY_ROOT_TIMESTAMP)
@echo -e ${CL_CYN}"----- Making uncompressed recovery ramdisk ------"${CL_RST}
$(MKBOOTFS) $(TARGET_RECOVERY_ROOT_OUT) > $@

$(recovery_ramdisk): $(MKBOOTFS) \
    $(recovery_uncompressed_ramdisk)
@echo -e ${CL_CYN}"----- Making recovery ramdisk ------"${CL_RST}
$(MINIGZIP) < $(recovery_uncompressed_ramdisk) > $@

ifndef BOARD_CUSTOM_BOOTIMG_MK
$(INSTALLED_RECOVERYIMAGE_TARGET): $(MKBOOTIMG) \
$(recovery_ramdisk) \
$(recovery_kernel)
@echo -e ${CL_CYN}"----- Making recovery image ------"${CL_RST}
$(MKBOOTIMG) $(INTERNAL_RECOVERYIMAGE_ARGS) --output $@
@echo -e ${CL_CYN}"Made recovery image: $@"${CL_RST}
$(hide) $(call assert-max-image-size,$@,$(BOARD_RECOVERYIMAGE_PARTITION_SIZE
endif

else
INSTALLED_RECOVERYIMAGE_TARGET :=
endif

.PHONY: recoveryimage
recoveryimage: $(INSTALLED_RECOVERYIMAGE_TARGET)

INSTALLED_RECOVERYZIP_TARGET := $(PRODUCT_OUT)/utilities/update.zip
$(INSTALLED_RECOVERYZIP_TARGET): $(INSTALLED_RECOVERYIMAGE_TARGET) $(TARGET_
@echo -e ${CL_CYN}"----- Making recovery zip -----"${CL_RST}
./build/tools/device/mkrecoveryzip.sh $(PRODUCT_OUT) $(HOST_OUT_JAVA_LIBRARI

.PHONY: recoveryzip
recoveryzip: $(INSTALLED_RECOVERYZIP_TARGET)

ifneq ($(BOARD_NAND_PAGE_SIZE),)
mkyaffs2_extra_flags := -c $(BOARD_NAND_PAGE_SIZE)
else
mkyaffs2_extra_flags :=
```

```
BOARD_NAND_PAGE_SIZE := 2048
endif

ifneq ($(BOARD_NAND_SPARE_SIZE),)
mkyaffs2_extra_flags += -s $(BOARD_NAND_SPARE_SIZE)
else
BOARD_NAND_SPARE_SIZE := 64
endif

ifdef BOARD_CUSTOM_BOOTIMG_MK
include $(BOARD_CUSTOM_BOOTIMG_MK)
endif

# ---------------------------------------------------------------
# system image
#

INTERNAL_SYSTEMIMAGE_FILES := $(filter $(TARGET_OUT)/%, \
    $(ALL_PREBUILT) \
    $(ALL_COPIED_HEADERS) \
    $(ALL_GENERATED_SOURCES) \
    $(ALL_DEFAULT_INSTALLED_MODULES)\
    $(ALL_PDK_FUSION_FILES))

ifdef is_tests_build
# We don't want to install tests modules to the system partition
# when building "tests", because now "tests" may be built in a user, userdebug
# or eng build variant and we don't want to pollute the system partition.
# INTERNAL_SYSTEMIMAGE_FILES += $(filter $(TARGET_OUT)/%, \
#    $(tests_MODULES))
endif

FULL_SYSTEMIMAGE_DEPS := $(INTERNAL_SYSTEMIMAGE_FILES) $(INTERNAL_USERIMAGES
# ---------------------------------------------------------------
# installed file list
# Depending on anything that $(BUILT_SYSTEMIMAGE) depends on.
# We put installed-files.txt ahead of image itself in the dependency graph
# so that we can get the size stat even if the build fails due to too large
# system image.
INSTALLED_FILES_FILE := $(PRODUCT_OUT)/installed-files.txt
$(INSTALLED_FILES_FILE): $(FULL_SYSTEMIMAGE_DEPS)
@echo Installed file list: $@
@mkdir -p $(dir $@)
@rm -f $@
$(hide) build/tools/fileslist.py $(TARGET_OUT) > $@
```

```
.PHONY: installed-file-list
installed-file-list: $(INSTALLED_FILES_FILE)
ifneq ($(filter sdk win_sdk,$(MAKECMDGOALS)),)
$(call dist-for-goals, sdk win_sdk, $(INSTALLED_FILES_FILE))
endif
ifneq ($(filter sdk_addon,$(MAKECMDGOALS)),)
$(call dist-for-goals, sdk_addon, $(INSTALLED_FILES_FILE))
endif

systemimage_intermediates := \
    $(call intermediates-dir-for,PACKAGING,systemimage)
BUILT_SYSTEMIMAGE := $(systemimage_intermediates)/system.img

# $(1): output file
define build-systemimage-target
  @echo "Target system fs image: $(1)"
 @mkdir -p $(dir $(1)) $(systemimage_intermediates) && rm -rf $(systemimage_in
 $(call generate-userimage-prop-dictionary, $(systemimage_intermediates)/sys
 $(hide) PATH=$(foreach p,$(INTERNAL_USERIMAGES_BINARY_PATHS),$(p):)$$PATH \
      ./build/tools/releasetools/build_image.py \
    $(TARGET_OUT) $(systemimage_intermediates)/system_image_info.txt $(1)
endef

$(BUILT_SYSTEMIMAGE): $(FULL_SYSTEMIMAGE_DEPS) $(INSTALLED_FILES_FILE)
$(call build-systemimage-target,$@)

INSTALLED_SYSTEMIMAGE := $(PRODUCT_OUT)/system.img
SYSTEMIMAGE_SOURCE_DIR := $(TARGET_OUT)

# The system partition needs room for the recovery image as well.  We
# now store the recovery image as a binary patch using the boot image
# as the source (since they are very similar).  Generate the patch so
# we can see how big it's going to be, and include that in the system
# image size check calculation.
ifneq ($(INSTALLED_RECOVERYIMAGE_TARGET),)
intermediates := $(call intermediates-dir-for,PACKAGING,recovery_patch)
ifndef BOARD_CUSTOM_BOOTIMG_MK
ifeq ($(CM_BUILD),)
RECOVERY_FROM_BOOT_PATCH := $(intermediates)/recovery_from_boot.p
else
RECOVERY_FROM_BOOT_PATCH :=
endif
else
RECOVERY_FROM_BOOT_PATCH :=
endif
$(RECOVERY_FROM_BOOT_PATCH): $(INSTALLED_RECOVERYIMAGE_TARGET) \
```

```
                              $(INSTALLED_BOOTIMAGE_TARGET) \
     $(HOST_OUT_EXECUTABLES)/imgdiff \
                       $(HOST_OUT_EXECUTABLES)/bsdiff
@echo -e ${CL_CYN}"Construct recovery from boot"${CL_RST}
mkdir -p $(dir $@)
PATH=$(HOST_OUT_EXECUTABLES):$$PATH $(HOST_OUT_EXECUTABLES)/imgdiff $(INSTAL
endif


$(INSTALLED_SYSTEMIMAGE): $(BUILT_SYSTEMIMAGE) $(RECOVERY_FROM_BOOT_PATCH) |
@echo -e ${CL_CYN}"Install system fs image: $@"${CL_RST}
$(copy-file-to-target)
$(hide) $(call assert-max-image-size,$@ $(RECOVERY_FROM_BOOT_PATCH),$(BOARD_

systemimage: $(INSTALLED_SYSTEMIMAGE)

.PHONY: systemimage-nodeps snod
systemimage-nodeps snod: $(filter-out systemimage-nodeps snod,$(MAKECMDGOALS)
            | $(INTERNAL_USERIMAGES_DEPS)
@echo "make $@: ignoring dependencies"
$(call build-systemimage-target,$(INSTALLED_SYSTEMIMAGE))
$(hide) $(call assert-max-image-size,$(INSTALLED_SYSTEMIMAGE),$(BOARD_SYSTEM

ifneq (,$(filter systemimage-nodeps snod, $(MAKECMDGOALS)))
ifeq (true,$(WITH_DEXPREOPT))
$(warning Warning: with dexpreopt enabled, you may need a full rebuild.)
endif
endif

#######
## system tarball
define build-systemtarball-target
   $(call pretty,"Target system fs tarball: $(INSTALLED_SYSTEMTARBALL_TARGET)
    $(MKTARBALL) $(FS_GET_STATS) \
$(PRODUCT_OUT) system $(PRIVATE_SYSTEM_TAR) \
$(INSTALLED_SYSTEMTARBALL_TARGET)
endef

ifndef SYSTEM_TARBALL_FORMAT
    SYSTEM_TARBALL_FORMAT := bz2
endif

system_tar := $(PRODUCT_OUT)/system.tar
INSTALLED_SYSTEMTARBALL_TARGET := $(system_tar).$(SYSTEM_TARBALL_FORMAT)
$(INSTALLED_SYSTEMTARBALL_TARGET): PRIVATE_SYSTEM_TAR := $(system_tar)
$(INSTALLED_SYSTEMTARBALL_TARGET): $(FS_GET_STATS) $(INTERNAL_SYSTEMIMAGE_FI
```

```
$(build-systemtarball-target)

.PHONY: systemtarball-nodeps
systemtarball-nodeps: $(FS_GET_STATS) \
               $(filter-out systemtarball-nodeps stnod,$(MAKECMDGOALS))
$(build-systemtarball-target)

.PHONY: stnod
stnod: systemtarball-nodeps

# For platform-java goal, add platform as well
ifneq (,$(filter platform-java, $(MAKECMDGOALS)))
PLATFORM_ZIP_ADD_JAVA := true
endif

#######
## platform.zip: system, plus other files to be used in PDK fusion build,
## in a zip file
INSTALLED_PLATFORM_ZIP := $(PRODUCT_OUT)/platform.zip
$(INSTALLED_PLATFORM_ZIP) : $(INTERNAL_SYSTEMIMAGE_FILES)
$(call pretty,"Platform zip package: $(INSTALLED_PLATFORM_ZIP)")
$(hide) rm -f $@
$(hide) cd $(dir $@) && zip -qry $(notdir $@) \
$(TARGET_COPY_OUT_SYSTEM) \
$(patsubst $(PRODUCT_OUT)/%, %, $(TARGET_OUT_NOTICE_FILES))
ifeq (true,$(PLATFORM_ZIP_ADD_JAVA))
$(hide) cd $(OUT_DIR) && zip -qry $(patsubst $(OUT_DIR)/%,%,$@) $(PDK_PLATFORM
endif


.PHONY: platform
platform: $(INSTALLED_PLATFORM_ZIP)

.PHONY: platform-java
platform-java: platform

# Dist the platform.zip
ifneq (,$(filter platform platform-java, $(MAKECMDGOALS)))
$(call dist-for-goals, platform platform-java, $(INSTALLED_PLATFORM_ZIP))
endif

#######
## boot tarball
define build-boottarball-target
   $(hide) echo "Target boot fs tarball: $(INSTALLED_BOOTTARBALL_TARGET)"
    $(hide) mkdir -p $(PRODUCT_OUT)/boot
```

```
    $(hide) cp -f $(INTERNAL_BOOTIMAGE_FILES) $(PRODUCT_OUT)/boot/.
   $(hide) echo $(BOARD_KERNEL_CMDLINE) > $(PRODUCT_OUT)/boot/cmdline
    $(hide) $(MKTARBALL) $(FS_GET_STATS) \
                $(PRODUCT_OUT) boot $(PRIVATE_BOOT_TAR) \
                $(INSTALLED_BOOTTARBALL_TARGET)
endef

ifndef BOOT_TARBALL_FORMAT
    BOOT_TARBALL_FORMAT := bz2
endif

boot_tar := $(PRODUCT_OUT)/boot.tar
INSTALLED_BOOTTARBALL_TARGET := $(boot_tar).$(BOOT_TARBALL_FORMAT)
$(INSTALLED_BOOTTARBALL_TARGET): PRIVATE_BOOT_TAR := $(boot_tar)
$(INSTALLED_BOOTTARBALL_TARGET): $(FS_GET_STATS) $(INTERNAL_BOOTIMAGE_FILES)
$(build-boottarball-target)

.PHONY: boottarball-nodeps btnod
boottarball-nodeps btnod: $(FS_GET_STATS) \
               $(filter-out boottarball-nodeps btnod,$(MAKECMDGOALS))
$(build-boottarball-target)


# ------------------------------------------------------------------
# data partition image
INTERNAL_USERDATAIMAGE_FILES := \
    $(filter $(TARGET_OUT_DATA)/%,$(ALL_DEFAULT_INSTALLED_MODULES))

# If we build "tests" at the same time, make sure $(tests_MODULES) get covered.
ifdef is_tests_build
INTERNAL_USERDATAIMAGE_FILES += \
    $(filter $(TARGET_OUT_DATA)/%,$(tests_MODULES))
endif

userdataimage_intermediates := \
    $(call intermediates-dir-for,PACKAGING,userdata)
BUILT_USERDATAIMAGE_TARGET := $(PRODUCT_OUT)/userdata.img

define build-userdataimage-target
 $(call pretty,"Target userdata fs image: $(INSTALLED_USERDATAIMAGE_TARGET)")
  @mkdir -p $(TARGET_OUT_DATA)
 @mkdir -p $(userdataimage_intermediates) && rm -rf $(userdataimage_intermedi
 $(call generate-userimage-prop-dictionary, $(userdataimage_intermediates)/u
 $(hide) PATH=$(foreach p,$(INTERNAL_USERIMAGES_BINARY_PATHS),$(p):)$$PATH \
      ./build/tools/releasetools/build_image.py \
    $(TARGET_OUT_DATA) $(userdataimage_intermediates)/userdata_image_info.tx
```

```
 $(hide) $(call assert-max-image-size,$(INSTALLED_USERDATAIMAGE_TARGET),$(BO
endef

# We just build this directly to the install location.
INSTALLED_USERDATAIMAGE_TARGET := $(BUILT_USERDATAIMAGE_TARGET)
$(INSTALLED_USERDATAIMAGE_TARGET): $(INTERNAL_USERIMAGES_DEPS) \
                                   $(INTERNAL_USERDATAIMAGE_FILES)
$(build-userdataimage-target)

.PHONY: userdataimage-nodeps
userdataimage-nodeps: | $(INTERNAL_USERIMAGES_DEPS)
$(build-userdataimage-target)

#######
## data partition tarball
define build-userdatatarball-target
    $(call pretty,"Target userdata fs tarball: " \
                  "$(INSTALLED_USERDATATARBALL_TARGET)")
    $(MKTARBALL) $(FS_GET_STATS) \
$(PRODUCT_OUT) data $(PRIVATE_USERDATA_TAR) \
$(INSTALLED_USERDATATARBALL_TARGET)
endef

userdata_tar := $(PRODUCT_OUT)/userdata.tar
INSTALLED_USERDATATARBALL_TARGET := $(userdata_tar).bz2
$(INSTALLED_USERDATATARBALL_TARGET): PRIVATE_USERDATA_TAR := $(userdata_tar)
$(INSTALLED_USERDATATARBALL_TARGET): $(FS_GET_STATS) $(INTERNAL_USERDATAIMAG
$(build-userdatatarball-target)

.PHONY: userdatatarball-nodeps
userdatatarball-nodeps: $(FS_GET_STATS)
$(build-userdatatarball-target)


# ----------------------------------------------------------------
# cache partition image
ifdef BOARD_CACHEIMAGE_FILE_SYSTEM_TYPE
INTERNAL_CACHEIMAGE_FILES := \
   $(filter $(TARGET_OUT_CACHE)/%,$(ALL_DEFAULT_INSTALLED_MODULES))

cacheimage_intermediates := \
    $(call intermediates-dir-for,PACKAGING,cache)
BUILT_CACHEIMAGE_TARGET := $(PRODUCT_OUT)/cache.img

define build-cacheimage-target
 $(call pretty,"Target cache fs image: $(INSTALLED_CACHEIMAGE_TARGET)")
```

```
  @mkdir -p $(TARGET_OUT_CACHE)
 @mkdir -p $(cacheimage_intermediates) && rm -rf $(cacheimage_intermediates)/
 $(call generate-userimage-prop-dictionary, $(cacheimage_intermediates)/cach
 $(hide) PATH=$(foreach p,$(INTERNAL_USERIMAGES_BINARY_PATHS),$(p):)$$PATH \
      ./build/tools/releasetools/build_image.py \
    $(TARGET_OUT_CACHE) $(cacheimage_intermediates)/cache_image_info.txt $(I
 $(hide) $(call assert-max-image-size,$(INSTALLED_CACHEIMAGE_TARGET),$(BOARD
endef

# We just build this directly to the install location.
INSTALLED_CACHEIMAGE_TARGET := $(BUILT_CACHEIMAGE_TARGET)
$(INSTALLED_CACHEIMAGE_TARGET): $(INTERNAL_USERIMAGES_DEPS) $(INTERNAL_CACHE
$(build-cacheimage-target)

.PHONY: cacheimage-nodeps
cacheimage-nodeps: | $(INTERNAL_USERIMAGES_DEPS)
$(build-cacheimage-target)

endif # BOARD_CACHEIMAGE_FILE_SYSTEM_TYPE

# ----------------------------------------------------------------
# bring in the installer image generation defines if necessary
ifeq ($(TARGET_USE_DISKINSTALLER),true)
include bootable/diskinstaller/config.mk
endif

# ----------------------------------------------------------------
# host tools needed to build dist and OTA packages

DISTTOOLS :=  $(HOST_OUT_EXECUTABLES)/minigzip \
  $(HOST_OUT_EXECUTABLES)/mkbootfs \
  $(HOST_OUT_EXECUTABLES)/mkbootimg \
  $(HOST_OUT_EXECUTABLES)/unpackbootimg \
  $(HOST_OUT_EXECUTABLES)/fs_config \
  $(HOST_OUT_EXECUTABLES)/mkyaffs2image \
  $(HOST_OUT_EXECUTABLES)/zipalign \
  $(HOST_OUT_EXECUTABLES)/bsdiff \
  $(HOST_OUT_EXECUTABLES)/imgdiff \
  $(HOST_OUT_JAVA_LIBRARIES)/dumpkey.jar \
  $(HOST_OUT_JAVA_LIBRARIES)/signapk.jar \
  $(HOST_OUT_EXECUTABLES)/mkuserimg.sh \
  $(HOST_OUT_EXECUTABLES)/make_ext4fs

OTATOOLS := $(DISTTOOLS) \
  $(HOST_OUT_EXECUTABLES)/aapt
```

```
.PHONY: otatools
otatools: $(OTATOOLS)


UNPACKBOOTIMG :=  $(HOST_OUT_EXECUTABLES)/unpackbootimg

.PHONY: unpackbootimg
otatools: $(UNPACKBOOTIMG)

# ----------------------------------------------------------------
# A zip of the directories that map to the target filesystem.
# This zip can be used to create an OTA package or filesystem image
# as a post-build step.
#
name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-target_files-$(FILE_NAME_TAG)

intermediates := $(call intermediates-dir-for,PACKAGING,target_files)
BUILT_TARGET_FILES_PACKAGE := $(intermediates)/$(name).zip
$(BUILT_TARGET_FILES_PACKAGE): intermediates := $(intermediates)
$(BUILT_TARGET_FILES_PACKAGE): \
zip_root := $(intermediates)/$(name)

# $(1): Directory to copy
# $(2): Location to copy it to
# The "ls -A" is to prevent "acp s/* d" from failing if s is empty.
define package_files-copy-root
  if [ -d "$(strip $(1))" -a "$$(ls -A $(1))" ]; then \
    mkdir -p $(2) && \
    $(ACP) -rd $(strip $(1))/* $(2); \
  fi
endef

built_ota_tools := \
$(call intermediates-dir-for,EXECUTABLES,applypatch)/applypatch \
$(call intermediates-dir-for,EXECUTABLES,applypatch_static)/applypatch_stati
$(call intermediates-dir-for,EXECUTABLES,check_prereq)/check_prereq \
$(call intermediates-dir-for,EXECUTABLES,sqlite3)/sqlite3 \
$(call intermediates-dir-for,EXECUTABLES,updater)/updater
$(BUILT_TARGET_FILES_PACKAGE): PRIVATE_OTA_TOOLS := $(built_ota_tools)

$(BUILT_TARGET_FILES_PACKAGE): PRIVATE_RECOVERY_API_VERSION := $(RECOVERY_AP
```

```
ifeq ($(TARGET_RELEASETOOLS_EXTENSIONS),)
# default to common dir for device vendor
$(BUILT_TARGET_FILES_PACKAGE): tool_extensions := $(TARGET_DEVICE_DIR)/../co
else
$(BUILT_TARGET_FILES_PACKAGE): tool_extensions := $(TARGET_RELEASETOOLS_EXTE
endif

# Depending on the various images guarantees that the underlying
# directories are up-to-date.
$(BUILT_TARGET_FILES_PACKAGE): \
$(INSTALLED_BOOTIMAGE_TARGET) \
$(INSTALLED_RADIOIMAGE_TARGET) \
$(INSTALLED_RECOVERYIMAGE_TARGET) \
$(INSTALLED_SYSTEMIMAGE) \
$(INSTALLED_USERDATAIMAGE_TARGET) \
$(INSTALLED_CACHEIMAGE_TARGET) \
$(INSTALLED_ANDROID_INFO_TXT_TARGET) \
$(built_ota_tools) \
$(APKCERTS_FILE) \
$(HOST_OUT_EXECUTABLES)/fs_config \
| $(ACP)
@echo -e ${CL_YLW}"Package target files:"${CL_RST}" $@"
$(hide) rm -rf $@ $(zip_root)
$(hide) mkdir -p $(dir $@) $(zip_root)
@# Components of the recovery image
$(hide) mkdir -p $(zip_root)/RECOVERY
$(hide) $(call package_files-copy-root, \
$(TARGET_RECOVERY_ROOT_OUT),$(zip_root)/RECOVERY/RAMDISK)
ifdef INSTALLED_KERNEL_TARGET
$(hide) $(ACP) $(INSTALLED_KERNEL_TARGET) $(zip_root)/RECOVERY/kernel
endif
ifdef INSTALLED_2NDBOOTLOADER_TARGET
$(hide) $(ACP) \
$(INSTALLED_2NDBOOTLOADER_TARGET) $(zip_root)/RECOVERY/second
endif
ifdef BOARD_KERNEL_CMDLINE
$(hide) echo "$(BOARD_KERNEL_CMDLINE)" > $(zip_root)/RECOVERY/cmdline
endif
ifdef BOARD_KERNEL_BASE
$(hide) echo "$(BOARD_KERNEL_BASE)" > $(zip_root)/RECOVERY/base
endif
ifdef BOARD_KERNEL_PAGESIZE
$(hide) echo "$(BOARD_KERNEL_PAGESIZE)" > $(zip_root)/RECOVERY/pagesize
endif

@# Components of the boot image
```

```
$(hide) mkdir -p $(zip_root)/BOOT
$(hide) $(call package_files-copy-root, \
$(TARGET_ROOT_OUT),$(zip_root)/BOOT/RAMDISK)
ifdef INSTALLED_KERNEL_TARGET
$(hide) $(ACP) $(INSTALLED_KERNEL_TARGET) $(zip_root)/BOOT/kernel
endif
ifdef INSTALLED_RAMDISK_TARGET
$(hide) $(ACP) $(INSTALLED_RAMDISK_TARGET) $(zip_root)/BOOT/ramdisk.img
endif
ifdef INSTALLED_BOOTLOADER_MODULE
$(hide) $(ACP) \
$(INSTALLED_BOOTLOADER_MODULE) $(zip_root)/BOOT/bootloader
endif
ifdef INSTALLED_2NDBOOTLOADER_TARGET
$(hide) $(ACP) \
$(INSTALLED_2NDBOOTLOADER_TARGET) $(zip_root)/BOOT/second
endif
ifdef BOARD_KERNEL_CMDLINE
$(hide) echo "$(BOARD_KERNEL_CMDLINE)" > $(zip_root)/BOOT/cmdline
endif
ifdef BOARD_KERNEL_BASE
$(hide) echo "$(BOARD_KERNEL_BASE)" > $(zip_root)/BOOT/base
endif
ifdef BOARD_KERNEL_PAGESIZE
$(hide) echo "$(BOARD_KERNEL_PAGESIZE)" > $(zip_root)/BOOT/pagesize
endif
ifdef BOARD_FORCE_RAMDISK_ADDRESS
$(hide) echo "$(BOARD_FORCE_RAMDISK_ADDRESS)" > $(zip_root)/BOOT/ramdiskaddr
endif
ifdef ZIP_SAVE_UBOOTIMG_ARGS
$(hide) echo "$(ZIP_SAVE_UBOOTIMG_ARGS)" > $(zip_root)/BOOT/ubootargs
endif
$(hide) $(foreach t,$(INSTALLED_RADIOIMAGE_TARGET),\
            mkdir -p $(zip_root)/RADIO; \
            $(ACP) $(t) $(zip_root)/RADIO/$(notdir $(t));)
@# Contents of the system image
$(hide) $(call package_files-copy-root, \
$(SYSTEMIMAGE_SOURCE_DIR),$(zip_root)/SYSTEM)
@# Contents of the data image
$(hide) $(call package_files-copy-root, \
$(TARGET_OUT_DATA),$(zip_root)/DATA)
@# Extra contents of the OTA package
$(hide) mkdir -p $(zip_root)/OTA/bin
$(hide) $(ACP) $(INSTALLED_ANDROID_INFO_TXT_TARGET) $(zip_root)/OTA/
$(hide) $(ACP) $(PRIVATE_OTA_TOOLS) $(zip_root)/OTA/bin/
@# Files that do not end up in any images, but are necessary to
```

```
@# build them.
$(hide) mkdir -p $(zip_root)/META
$(hide) $(ACP) $(APKCERTS_FILE) $(zip_root)/META/apkcerts.txt
$(hide) echo "$(PRODUCT_OTA_PUBLIC_KEYS)" > $(zip_root)/META/otakeys.txt
$(hide) echo "recovery_api_version=$(PRIVATE_RECOVERY_API_VERSION)" > $(zip_r
ifdef BOARD_FLASH_BLOCK_SIZE
$(hide) echo "blocksize=$(BOARD_FLASH_BLOCK_SIZE)" >> $(zip_root)/META/misc_i
endif
ifdef BOARD_BOOTIMAGE_PARTITION_SIZE
$(hide) echo "boot_size=$(BOARD_BOOTIMAGE_PARTITION_SIZE)" >> $(zip_root)/MET
endif
ifdef BOARD_RECOVERYIMAGE_PARTITION_SIZE
$(hide) echo "recovery_size=$(BOARD_RECOVERYIMAGE_PARTITION_SIZE)" >> $(zip_r
endif
$(hide) echo "tool_extensions=$(tool_extensions)" >> $(zip_root)/META/misc_in
$(hide) echo "default_system_dev_certificate=$(DEFAULT_SYSTEM_DEV_CERTIFICAT
ifdef PRODUCT_EXTRA_RECOVERY_KEYS
$(hide) echo "extra_recovery_keys=$(PRODUCT_EXTRA_RECOVERY_KEYS)" >> $(zip_ro
endif
$(call generate-userimage-prop-dictionary, $(zip_root)/META/misc_info.txt)
@# Zip everything up, preserving symlinks
$(hide) (cd $(zip_root) && zip -qry ../$(notdir $@) .)
@# Run fs_config on all the system, boot ramdisk, and recovery ramdisk files in
$(hide) zipinfo -1 $@ | awk 'BEGIN { FS="SYSTEM/" } /^SYSTEM\// {print "system/
$(hide) zipinfo -1 $@ | awk 'BEGIN { FS="BOOT/RAMDISK/" } /^BOOT\/RAMDISK\// {p
$(hide) zipinfo -1 $@ | awk 'BEGIN { FS="RECOVERY/RAMDISK/" } /^RECOVERY\/RAMDI
$(hide) (cd $(zip_root) && zip -q ../$(notdir $@) META/*filesystem_config.txt)


target-files-package: $(BUILT_TARGET_FILES_PACKAGE)


ifneq ($(TARGET_PRODUCT),sdk)
ifeq ($(filter generic%,$(TARGET_DEVICE)),)
ifneq ($(TARGET_NO_KERNEL),true)
ifneq ($(recovery_fstab),)

# ------------------------------------------------------------------
# OTA update package

name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-ota-$(FILE_NAME_TAG)
```

```
INTERNAL_OTA_PACKAGE_TARGET := $(PRODUCT_OUT)/$(name).zip

$(INTERNAL_OTA_PACKAGE_TARGET): KEY_CERT_PAIR := $(DEFAULT_KEY_CERT_PAIR)

ifeq ($(TARGET_RELEASETOOL_OTA_FROM_TARGET_SCRIPT),)
   OTA_FROM_TARGET_SCRIPT := ./build/tools/releasetools/ota_from_target_file
else
   OTA_FROM_TARGET_SCRIPT := $(TARGET_RELEASETOOL_OTA_FROM_TARGET_SCRIPT)
endif

ifneq ($(CM_BUILD),)
    $(INTERNAL_OTA_PACKAGE_TARGET): backuptool := true
else
    $(INTERNAL_OTA_PACKAGE_TARGET): backuptool := false
endif

ifneq ($(NAM_VARIANT),)
    $(INTERNAL_OTA_PACKAGE_TARGET): modelidcfg := true
else
    $(INTERNAL_OTA_PACKAGE_TARGET): modelidcfg := false
endif

ifeq ($(TARGET_OTA_ASSERT_DEVICE),)
    $(INTERNAL_OTA_PACKAGE_TARGET): override_device := auto
else
   $(INTERNAL_OTA_PACKAGE_TARGET): override_device := $(TARGET_OTA_ASSERT_DEV
endif

$(INTERNAL_OTA_PACKAGE_TARGET): $(BUILT_TARGET_FILES_PACKAGE) $(DISTTOOLS) $
@echo -e ${CL_YLW}"Package OTA:"${CL_RST}" $@"
$(OTA_FROM_TARGET_SCRIPT) -v \
   -p $(HOST_OUT) \
   -k $(KEY_CERT_PAIR) \
   --backup=$(backuptool) \
   --override_device=$(override_device) \
   $(BUILT_TARGET_FILES_PACKAGE) $@

.PHONY: otapackage bacon
otapackage: $(INTERNAL_OTA_PACKAGE_TARGET)
bacon: otapackage
ifneq ($(TARGET_CUSTOM_RELEASETOOL),)
@echo -e ${CL_YLW}"Running custom releasetool..."${CL_RST}
$(hide) OTAPACKAGE=$(INTERNAL_OTA_PACKAGE_TARGET) APKCERTS=$(APKCERTS_FILE)
else
@echo -e ${CL_YLW}"Running releasetool..."${CL_RST}
$(hide) OTAPACKAGE=$(INTERNAL_OTA_PACKAGE_TARGET) APKCERTS=$(APKCERTS_FILE)
```

```
endif

# ---------------------------------------------------------------
# The update package

name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-img-$(FILE_NAME_TAG)

INTERNAL_UPDATE_PACKAGE_TARGET := $(PRODUCT_OUT)/$(name).zip

ifeq ($(TARGET_RELEASETOOLS_EXTENSIONS),)
# default to common dir for device vendor
$(INTERNAL_UPDATE_PACKAGE_TARGET): extensions := $(TARGET_DEVICE_DIR)/../com
else
$(INTERNAL_UPDATE_PACKAGE_TARGET): extensions := $(TARGET_RELEASETOOLS_EXTEN
endif

ifeq ($(TARGET_RELEASETOOL_IMG_FROM_TARGET_SCRIPT),)
   IMG_FROM_TARGET_SCRIPT := ./build/tools/releasetools/img_from_target_file
else
   IMG_FROM_TARGET_SCRIPT := $(TARGET_RELEASETOOL_IMG_FROM_TARGET_SCRIPT)
endif

$(INTERNAL_UPDATE_PACKAGE_TARGET): $(BUILT_TARGET_FILES_PACKAGE) $(DISTTOOLS
@echo -e ${CL_YLW}"Package:"${CL_RST}" $@"
$(IMG_FROM_TARGET_SCRIPT) -v \
   -s $(extensions) \
   -p $(HOST_OUT) \
   $(BUILT_TARGET_FILES_PACKAGE) $@

.PHONY: updatepackage
updatepackage: $(INTERNAL_UPDATE_PACKAGE_TARGET)

endif    # recovery_fstab is defined
endif    # TARGET_NO_KERNEL != true
endif    # TARGET_DEVICE != generic*
endif    # TARGET_PRODUCT != sdk

# ---------------------------------------------------------------
# A zip of the tests that are built when running "make tests".
# This is very similar to BUILT_TARGET_FILES_PACKAGE, but we
# only grab DATA, and it's called "*-tests-*.zip".
#
```

```
name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-tests-$(FILE_NAME_TAG)

intermediates := $(call intermediates-dir-for,PACKAGING,tests_zip)
BUILT_TESTS_ZIP_PACKAGE := $(intermediates)/$(name).zip
$(BUILT_TESTS_ZIP_PACKAGE): intermediates := $(intermediates)
$(BUILT_TESTS_ZIP_PACKAGE): zip_root := $(intermediates)/$(name)

# Depending on the images guarantees that the underlying
# directories are up-to-date.
$(BUILT_TESTS_ZIP_PACKAGE): \
    $(INSTALLED_USERDATAIMAGE_TARGET) \
    | $(ACP)
@echo -e ${CL_YLW}"Package test files:"${CL_RST} $@"
$(hide) rm -rf $@ $(zip_root)
$(hide) mkdir -p $(dir $@) $(zip_root)
@# Contents of the data image
$(hide) $(call package_files-copy-root, \
$(TARGET_OUT_DATA),$(zip_root)/DATA)
$(hide) (cd $(zip_root) && zip -qry ../$(notdir $@) .)

.PHONY: tests-zip-package
tests-zip-package: $(BUILT_TESTS_ZIP_PACKAGE)

# Target needed by tests build
.PHONY: tests-build-target
tests-build-target: $(BUILT_TESTS_ZIP_PACKAGE)

ifneq (,$(filter $(MAKECMDGOALS),tests-build-target))
  $(call dist-for-goals, tests-build-target, \
          $(BUILT_TESTS_ZIP_PACKAGE))
endif

.PHONY: tests
tests: $(BUILT_TESTS_ZIP_PACKAGE)
ifneq (,$(filter tests, $(MAKECMDGOALS)))
$(call dist-for-goals, tests, $(BUILT_TESTS_ZIP_PACKAGE))
endif

# ------------------------------------------------------------------
# A zip of the symbols directory.  Keep the full paths to make it
# more obvious where these files came from.
#
```

```
name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-symbols-$(FILE_NAME_TAG)

SYMBOLS_ZIP := $(PRODUCT_OUT)/$(name).zip
$(SYMBOLS_ZIP): $(INSTALLED_SYSTEMIMAGE) $(INSTALLED_BOOTIMAGE_TARGET)
@echo "Package symbols: $@"
$(hide) rm -rf $@
$(hide) mkdir -p $(dir $@)
$(hide) zip -qr $@ $(TARGET_OUT_UNSTRIPPED)

# -----------------------------------------------------------------
# A zip of the Android Apps. Not keeping full path so that we don't
# include product names when distributing
#
name := $(TARGET_PRODUCT)
ifeq ($(TARGET_BUILD_TYPE),debug)
  name := $(name)_debug
endif
name := $(name)-apps-$(FILE_NAME_TAG)

APPS_ZIP := $(PRODUCT_OUT)/$(name).zip
$(APPS_ZIP): $(INSTALLED_SYSTEMIMAGE)
@echo -e ${CL_YLW}"Package apps:"${CL_RST}" $@"
$(hide) rm -rf $@
$(hide) mkdir -p $(dir $@)
$(hide) zip -qj $@ $(TARGET_OUT_APPS)/*


#-----------------------------------------------------------------
# A zip of emma code coverage meta files. Generated for fully emma
# instrumented build.
#
EMMA_META_ZIP := $(PRODUCT_OUT)/emma_meta.zip
$(EMMA_META_ZIP): $(INSTALLED_SYSTEMIMAGE)
@echo "Collecting Emma coverage meta files."
$(hide) find $(TARGET_COMMON_OUT_ROOT) -name "coverage.em" | \
zip -@ -q $@

# -----------------------------------------------------------------
# dalvik something
.PHONY: dalvikfiles
dalvikfiles: $(INTERNAL_DALVIK_MODULES)
```

```
# ------------------------------------------------------------------
# The emulator package

INTERNAL_EMULATOR_PACKAGE_FILES += \
        $(HOST_OUT_EXECUTABLES)/emulator$(HOST_EXECUTABLE_SUFFIX) \
        prebuilts/qemu-kernel/$(TARGET_ARCH)/kernel-qemu \
        $(INSTALLED_RAMDISK_TARGET) \
$(INSTALLED_SYSTEMIMAGE) \
$(INSTALLED_USERDATAIMAGE_TARGET)

name := $(TARGET_PRODUCT)-emulator-$(FILE_NAME_TAG)

INTERNAL_EMULATOR_PACKAGE_TARGET := $(PRODUCT_OUT)/$(name).zip

$(INTERNAL_EMULATOR_PACKAGE_TARGET): $(INTERNAL_EMULATOR_PACKAGE_FILES)
@echo -e ${CL_YLW}"Package:"${CL_RST}" $@"
$(hide) zip -qj $@ $(INTERNAL_EMULATOR_PACKAGE_FILES)

# ------------------------------------------------------------------
# Old PDK stuffs, retired
# The pdk package (Platform Development Kit)

#ifneq (,$(filter pdk,$(MAKECMDGOALS)))
#  include development/pdk/Pdk.mk
#endif


# ------------------------------------------------------------------
# The SDK

# The SDK includes host-specific components, so it belongs under HOST_OUT.
sdk_dir := $(HOST_OUT)/sdk

# Build a name that looks like:
#
#     linux-x86   --> android-sdk_12345_linux-x86
#     darwin-x86  --> android-sdk_12345_mac-x86
#     windows-x86 --> android-sdk_12345_windows
#
sdk_name := android-sdk_$(FILE_NAME_TAG)
ifeq ($(HOST_OS),darwin)
  INTERNAL_SDK_HOST_OS_NAME := mac
else
  INTERNAL_SDK_HOST_OS_NAME := $(HOST_OS)
endif
ifneq ($(HOST_OS),windows)
```

```
  INTERNAL_SDK_HOST_OS_NAME := $(INTERNAL_SDK_HOST_OS_NAME)-$(HOST_ARCH)
endif
sdk_name := $(sdk_name)_$(INTERNAL_SDK_HOST_OS_NAME)

sdk_dep_file := $(sdk_dir)/sdk_deps.mk

ATREE_FILES :=
-include $(sdk_dep_file)

# if we don't have a real list, then use "everything"
ifeq ($(strip $(ATREE_FILES)),)
ATREE_FILES := \
$(ALL_PREBUILT) \
$(ALL_COPIED_HEADERS) \
$(ALL_GENERATED_SOURCES) \
$(ALL_DEFAULT_INSTALLED_MODULES) \
$(INSTALLED_RAMDISK_TARGET) \
$(ALL_DOCS) \
$(ALL_SDK_FILES)
endif

atree_dir := development/build

# sdk/build/tools.atree contains the generic rules, while
#
# sdk/build/tools.$(TARGET_ARCH).atree contains target-specific rules
# the latter is optional.
#
sdk_tools_atree_files := sdk/build/tools.atree
ifneq (,$(strip $(wildcard sdk/build/tools.$(TARGET_ARCH).atree)))
  sdk_tools_atree_files += sdk/build/tools.$(TARGET_ARCH).atree
endif
ifneq (,$(strip $(wildcard sdk/build/tools.$(HOST_OS).atree)))
  sdk_tools_atree_files += sdk/build/tools.$(HOST_OS).atree
endif
ifneq (,$(strip $(wildcard sdk/build/tools.$(HOST_OS)-$(HOST_ARCH).atree)))
  sdk_tools_atree_files += sdk/build/tools.$(HOST_OS)-$(HOST_ARCH).atree
endif

sdk_atree_files := \
$(atree_dir)/sdk.exclude.atree \
$(atree_dir)/sdk.atree \
$(atree_dir)/sdk-$(HOST_OS)-$(HOST_ARCH).atree \
$(sdk_tools_atree_files)

# development/build/sdk-android-<abi>.atree is used to differentiate
```

```
# between architecture models (e.g. ARMv5TE versus ARMv7) when copying
# files like the kernel image. We use TARGET_CPU_ABI because we don't
# have a better way to distinguish between CPU models.
ifneq (,$(strip $(wildcard $(atree_dir)/sdk-android-$(TARGET_CPU_ABI).atree))
  sdk_atree_files += $(atree_dir)/sdk-android-$(TARGET_CPU_ABI).atree
endif

deps := \
$(target_notice_file_txt) \
$(tools_notice_file_txt) \
$(OUT_DOCS)/offline-sdk-timestamp \
$(SYMBOLS_ZIP) \
$(INSTALLED_SYSTEMIMAGE) \
$(INSTALLED_USERDATAIMAGE_TARGET) \
$(INSTALLED_RAMDISK_TARGET) \
$(INSTALLED_SDK_BUILD_PROP_TARGET) \
$(INSTALLED_BUILD_PROP_TARGET) \
$(ATREE_FILES) \
$(atree_dir)/sdk.atree \
$(sdk_tools_atree_files) \
$(HOST_OUT_EXECUTABLES)/atree \
    $(HOST_OUT_EXECUTABLES)/line_endings

INTERNAL_SDK_TARGET := $(sdk_dir)/$(sdk_name).zip
$(INTERNAL_SDK_TARGET): PRIVATE_NAME := $(sdk_name)
$(INTERNAL_SDK_TARGET): PRIVATE_DIR := $(sdk_dir)/$(sdk_name)
$(INTERNAL_SDK_TARGET): PRIVATE_DEP_FILE := $(sdk_dep_file)
$(INTERNAL_SDK_TARGET): PRIVATE_INPUT_FILES := $(sdk_atree_files)

# Set SDK_GNU_ERROR to non-empty to fail when a GNU target is built.
#
#SDK_GNU_ERROR := true

$(INTERNAL_SDK_TARGET): $(deps)
@echo "Package SDK: $@"
$(hide) rm -rf $(PRIVATE_DIR) $@
$(hide) for f in $(target_gnu_MODULES); do \
  if [ -f $$f ]; then \
    echo SDK: $(if $(SDK_GNU_ERROR),ERROR:,warning:) \
        including GNU target $$f >&2; \
    FAIL=$(SDK_GNU_ERROR); \
  fi; \
done; \
if [ $$FAIL ]; then exit 1; fi
$(hide) ( \
$(HOST_OUT_EXECUTABLES)/atree \
```

```
$(addprefix -f ,$(PRIVATE_INPUT_FILES)) \
-m $(PRIVATE_DEP_FILE) \
-I / \
-I . \
-I $(PRODUCT_OUT) \
-I $(HOST_OUT) \
-I $(TARGET_COMMON_OUT_ROOT) \
-v "PLATFORM_NAME=android-$(PLATFORM_VERSION)" \
-v "OUT_DIR=$(OUT_DIR)" \
-v "HOST_OUT=$(HOST_OUT)" \
-v "TARGET_ARCH=$(TARGET_ARCH)" \
-v "TARGET_CPU_ABI=$(TARGET_CPU_ABI)" \
-v "DLL_EXTENSION=$(HOST_SHLIB_SUFFIX)" \
-o $(PRIVATE_DIR) && \
cp -f $(target_notice_file_txt) \
$(PRIVATE_DIR)/system-images/android-$(PLATFORM_VERSION)/$(TARGET_CPU_ABI)/
cp -f $(tools_notice_file_txt) $(PRIVATE_DIR)/tools/NOTICE.txt && \
cp -f $(tools_notice_file_txt) $(PRIVATE_DIR)/platform-tools/NOTICE.txt && \
HOST_OUT_EXECUTABLES=$(HOST_OUT_EXECUTABLES) HOST_OS=$(HOST_OS) \
development/build/tools/sdk_clean.sh $(PRIVATE_DIR) && \
chmod -R ug+rwX $(PRIVATE_DIR) && \
cd $(dir $@) && zip -rq $(notdir $@) $(PRIVATE_NAME) \
) || ( rm -rf $(PRIVATE_DIR) $@ && exit 44 )


# Is a Windows SDK requested? If so, we need some definitions from here
# in order to find the Linux SDK used to create the Windows one.
MAIN_SDK_NAME := $(sdk_name)
MAIN_SDK_DIR  := $(sdk_dir)
MAIN_SDK_ZIP  := $(INTERNAL_SDK_TARGET)
ifneq ($(filter win_sdk,$(MAKECMDGOALS)),)
include $(TOPDIR)development/build/tools/windows_sdk.mk
endif

# ---------------------------------------------------------------
# Findbugs
INTERNAL_FINDBUGS_XML_TARGET := $(PRODUCT_OUT)/findbugs.xml
INTERNAL_FINDBUGS_HTML_TARGET := $(PRODUCT_OUT)/findbugs.html
$(INTERNAL_FINDBUGS_XML_TARGET): $(ALL_FINDBUGS_FILES)
@echo UnionBugs: $@
$(hide) prebuilt/common/findbugs/bin/unionBugs $(ALL_FINDBUGS_FILES) \
> $@
$(INTERNAL_FINDBUGS_HTML_TARGET): $(INTERNAL_FINDBUGS_XML_TARGET)
@echo ConvertXmlToText: $@
$(hide) prebuilt/common/findbugs/bin/convertXmlToText -html:fancy.xsl \
$(INTERNAL_FINDBUGS_XML_TARGET) > $@
```

```
# ---------------------------------------------------------------
# Findbugs

# ---------------------------------------------------------------
# These are some additional build tasks that need to be run.
include $(sort $(wildcard $(BUILD_SYSTEM)/tasks/*.mk))
-include $(sort $(wildcard vendor/*/build/tasks/*.mk))

# ---------------------------------------------------------------
# Create SDK repository packages. Must be done after tasks/* since
# we need the addon rules defined.
ifneq ($(sdk_repo_goal),)
include $(TOPDIR)development/build/tools/sdk_repo.mk
endif
```

在build/core/Makefile里的629行，可以看到这么一段文字

```
# The installed image, which may be optimized or unoptimized.
#
INSTALLED_SYSTEMIMAGE := $(PRODUCT_OUT)/system.img
```

从这里可以看出，系统应该会在$(PRODUCT_OUT)目录下生成system.img
再继续往下看，在662行有一个copy-file-to-target，这实现了将system.img从一个中间目录复制到/generic目录。
BUILD_SYSTEM的定义在636行。
这里的system.img不是/generic目录下面我们看到的那个system.img，而是另一个中间目录下的，但是是同一个文件。一开始看到的复制就是把out /target/product/g目录下面的system.img复制到/generic目录下。
现在，知道了system.img的来历，然后要分析它是一个什么东西，里面包含什么？？

Makefile line624

```
$(BUILT_SYSTEMIMAGE_UNOPT): $(INTERNAL_SYSTEMIMAGE_FILES) $(INTERNAL_MKUSERF
$(call build-systemimage-target,$@)
```

这里调用了build-systemimg-target Makefile line605

```
ifeq ($(TARGET_USERIMAGES_USE_EXT2),true)
## generate an ext2 image
# $(1): output file
define build-systemimage-target
@echo "Target system fs image: $(1)"
$(call build-userimage-ext2-target,$(TARGET_OUT),$(1),system,)
endef
else # TARGET_USERIMAGES_USE_EXT2 != true
## generate a yaffs2 image
# $(1): output file
define build-systemimage-target
@echo "Target system fs image: $(1)"
```

```
@mkdir -p $(dir $(1))
*$(hide) $(MKYAFFS2) -f $(TARGET_OUT) $(1)*
endef
endif # TARGET_USERIMAGES_USE_EXT2
```

找不到TARGET_USERIMAGES_USE_EXT2的定义！！！不过从上面的分析可以推断出应该是yaffs2文件系统

其中MKYAFFS2：（core/config.mk line161）

```
MKYAFFS2 := $(HOST_OUT_EXECUTABLES)/mkyaffs2image$(HOST_EXECUTABLE_SUFFIX)
```

定义MKYAFFS2是目录/home/chiplua/work/cm4.1/out/host/linux-x86/bin下的一个可执行文件mkyaffs2image，运行这个程序可得到如下信息：

```
/home/chiplua/work/cm4.1/out/host/linux-x86/bin$ ./mkyaffs2image
mkyaffs2image: image building tool for YAFFS2 built Nov 13 2009
usage: mkyaffs2image [-f] dir image_file [convert]
-f fix file stat (mods, user, group) for device
dir the directory tree to be converted
image_file the output file to hold the image
'convert' produce a big-endian image from a little-endian machine
```

得知这个程序可以生成yaffs2的文件系统映像。并且也清楚了上面*$(hide) $(MKYAFFS*的功能，把TARGET_OUT目录转变成yaffs2格式并输出成/home/chiplua/work/cm4.1/o就是我们最终在/generic目录下看到的那个system.img)。

到现在已经差不多知道system.img的产生过程，要弄清楚system.img里面的内容，就要分析TARGET_OUT目录的内容了。（想用mount把system.img挂载到linux下面看看里面什么东西，却不支持yaffs和yaffs2文件系统！！！）

下一步：分析TARGET_OUT 在build/core/envsetup.sh文件（line205）中找到了TARGET_OUT的定义：

```
TARGET_OUT := $(PRODUCT_OUT)/system
```

也就是/home/chiplua/work/cm4.1/out/target /product/generic目录下的system目录。

```
chiplua@chiplua:~/work/cm4.1/out/target/product/generic/system$ tree -L 1
|-- app
|-- bin
|-- build.prop
|-- etc
|-- fonts
|-- framework
|-- lib
|-- usr
`-- xbin
```

现在一切都明白了，我们最终看到的system.img文件是该目录下的system目录的一个映像，类似于linux的根文件系统的映像，放着android的应用程序，配置文件，字体等。

Userdata.img来来自于data目录，默认里面是没有文件的。

Annex A: Review Record

待评审的工作成果	【】文档【】代码		
评审方式	【】正规技术文档【】非正规技术文档		
评审时间			
评审所需设备			
参见技术评审的人员			
类别	名字	部门	职务
主持人			
评审小组成员			
记录员			
作者			
其他人员			

表 2: 评审表格

已识别的缺陷	建议缺陷解决方案

表 3: 缺陷识别

评审结论	【】工作成果合格，“无需修改”或者“需要轻微修改但不必再审核” 【】工作成果基本合格，需要做少量的修改，之后通过审核即可 【】工作成果不合格，需要做比较大的修改，之后必须重新对其评审
意见	
负责人签字	签字： 日期：

表 4: 评审结论与意见