Document Number:AAUG
Rev. 13.5.0_GA, 11/2012

Android Advanced User Guide

Contents

1 Overview

This document provides advanced information which explains how to customize or extend the feature lists in the 13.5.0-GA release package. It is primarily targeted for Android framework and kernel developer who wants to customize their product based on 13.5.0-GA.

2 Output Display Customization

The default image included in this package uses EInk as the primary output with 16bpp enabled. The diagram below shows the typical data path for each UI component (Windows and video) to the output display panel.

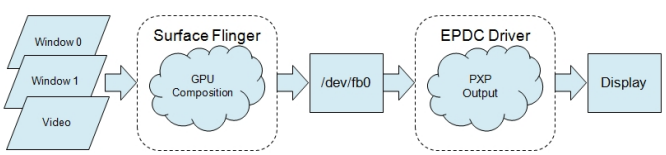

Figure 1. Typical Data Path for Each UI Component (Windows and video)

The description and data format for each node in the data path are described as shown below:

- **Window0-n**: each window represents an UI surface in the framework. The application draws their UI data into its window. The data format can be RGB565 or RGBA8888. There are two factors which impact the data format on the windows.
    - If Application requests transparent feature in its UI component, the data format is RGBA8888.
    - If no alpha blending feature is requested, the data format is decided by the Framebuffer 0' format. If the Framebuffer 0 is 16bpp (RGB565) format, the non-transparent windows format is RGB565. If the Framebuffer 0 is 32bpp(RGBA8888) format, the non-transparaent windows format is RGBA8888.

Transparent Request	Framebuffer 0' format	Windows' format
Y	16bpp	RGBA8888
Y	32bpp	RGBA8888
N	16bpp	RGB565
N	32bpp	RGBA8888

3 Memory Layout Customization

This section describes i.MX Android memory usage and layout for the entire system. When i.MX Android is running, the DDR memory is divided into 6 parts. The parts are:

- Linux Kernel reserved space
- Normal zone space managed by kernel's MM (high memory zone is also included)
- Reserved memory for GPU libs, drivers, surface view, normal surface double buffers and VPU's working buffer and bitstream
- Reserved space for framebuffer BG/FG triple buffers

The following diagram shows the default memory usage and layout on i.MX 6SoloLite board.

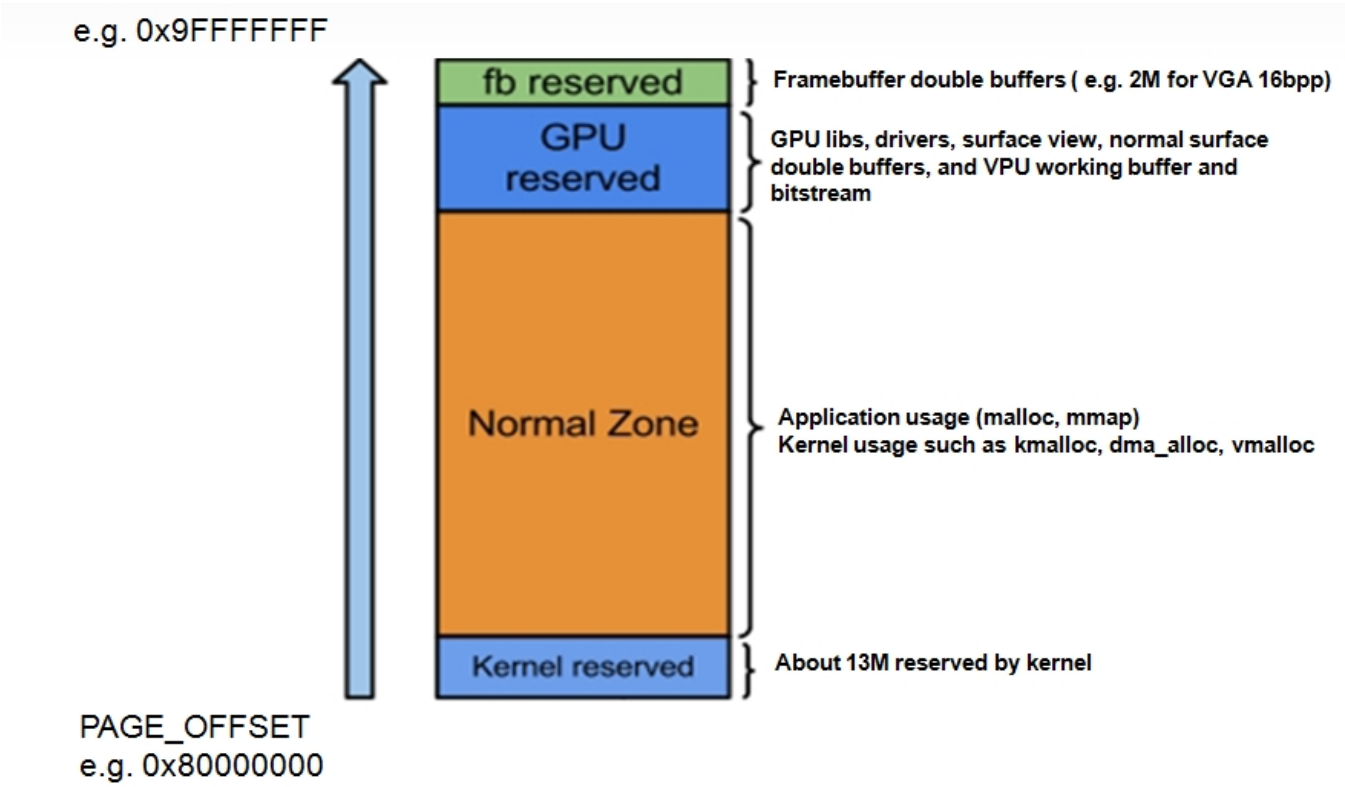

Figure 2. Default Memory Usage on i.MX 6SoloLite Board

The kernel reserved space cannot be adjusted. It is controlled by the kernel and the normal zone size and it depends on the total DDR size and the reserved spaces. It is not recommended to change reserved GPU memory size. Depending on the use case, a customer can adjust the reserved memory size for BG (background) framebuffer by adding the kernel parameters into the command line:

- for framebuffer

```
fbmem=<fb0 size>
```

For example:

If you have VGA EInk display, you have to specify the BG buffer size:

```
fbmem=10M
2M ~= 800x600x2 (16bpp) x2 (double buffers)
```

4 System Upgrade (OTA) Customization

This section introduces the Android OTA system. This section also outlines the making of the OTA package by using the Android build system, release image, signatures image, and the process of running the OTA application.

Android OTA system, when combined with build system, application, and the server, achieves seamless and safe system upgrade.

The seamless upgrade is provided by means of a build, system generated, upgrade which is packaged automatically. Also, the diff package is provided to more easily deal with the size of the upgrade package. The security is provided by signing the upgrade package and comparing with device's release key sign. It is also provided by binary patch and it will not apply if you find that the binary is not correct. The server needs to implement push message, or pulling, and release information query related function. Google has a standard service to do this, provided the support from Google is available.

The benefit of this service is obvious: you can fix the software issue or deploy new software release with very little effort.

4.1 Building an OTA Package

Android build system supports auto generation of update.zip function. It will generate the updater_script and all system.img files.

You can use `$ make dist` to generate ota package.

Use this command to build evk_6sl_eink product `$ make PRODUCT=evk_6sl_eink-eng dist -j4`

After this step, the build is finished.

You can find the few packages in following path for the OTA update.zip to do the OTA and make diff package.

```
out/dist/evk_6sl_eink-ota-eng.xxx.zip
out/target/product/evk_6sl_eink/evk_6sl_eink-ota-eng.xxx.zip
```

The package we used before R13.1 was update.zip. In this package, however, everything is auto generated.

4.2 How to Sign the OTA Package

Android requires that each application be signed with the developer's digital keys to enforce signature permissions and application request to use shared user ID or target process. For more information on the general Android security principles and signing requirements, see the Android Security and Permissions section in the Android Developer Guide. The core Android platform uses four keys to maintain security of core platform components:

- **platform**: a key for packages that are part of the core platform.
- **shared**: a key for things that are shared in the home/contacts process.
- **media**: a key for packages that are part of the media/download system.
- **releasekey**: the default key to sign with if nothing specified.

These keys are used to sign applications separately for release images and are not used by the Android build system. The build system signs packages with the testkeys provided in build/target/product/security/. Because the testkeys are part of the standard Android open source distribution, they should never be used for production devices. Instead, device manufacturers should generate their own private keys for shipping release builds.

Generating keys

A device manufacturer's keys for each product should be stored under vendor/<vendor_name>/security/<product_name>, where <vendor_name> and <product_name> represents the manufacturer and product names. To simplify key creation, copy the script below to this directory in a file called mkkey.sh. To customize your keys, change the line that starts with AUTH to reflect the correct information for your company:

```
#!/bin/sh

AUTH='/C=US/ST=California/L=Mountain
View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'

if [ "$1" == "" ]; then

        echo "Create a test certificate key."

        echo "Usage: $0 NAME"

        echo "Will generate NAME.pk8 and NAME.x509.pem"

        echo "  $AUTH"

        exit
```

```
fi
openssl genrsa -3 -out $1.pem 2048


openssl req -new -x509 -key $1.pem -out $1.x509.pem -days 10000 \

    -subj "$AUTH"


echo "Please enter the password for this key:"

openssl pkcs8 -in $1.pem -topk8 -outform DER -out $1.pk8 -passout stdin
```

mkkey.sh is a helper script to generate the platform's keys. NOTE: the password you type will be visible in your terminal window. Make a note of the passwords you use as you will need them to sign release builds.

To generate the required 4 platform keys, run mkkey.sh four times specifying the key name and password for each:

```
$sh mkkey.sh platform # enter password

$sh mkkey.sh media # enter password

$sh mkkey.sh shared # enter password

$sh mkkey.sh release # enter password
```

You should now have new keys for your product.

4.2.1 Signing a Build for Release

Signing a build for a release is a two-step process.

1. Sign all the individual parts of the build.
2. Put the parts back together into image files.

4.2.2 Signing Applications

Use build/tools/releasetools/sign_target_files_apks to sign target_files package. The target_files package isn't built by default. You need to make sure to specify the "dist" target when you call make. For example:

```
make -j4 PRODUCT-<product_name>-user dist
```

The command above creates a file under out/dist called: <product_name>-target_files.zip. This is the file you need to pass to the sign_target_files_apks script.

You would typically run the script like this:

```
./build/tools/releasetools/sign_target_files_apks -d
vendor/<vendor_name>/security/<product_name> <product_name>-target_files.zip
signed-target-files.zip
```

If you have pre-built and pre-signed APK's in your build that you don't want re-signed, you must explicitly ignore them by adding -e Foo.apk= to the command line for each APK you wish to ignore.

sign_target_files_apks also has many other options that could be useful for signing release builds. Run it with -h as the only option to see the full help.

4.2.3 Creating Image Files

Once you have signed-target-files.zip, create the images to put on a device by using the command below:

```
build/tools/releasetools/img_from_target_files signed-target-files.zip signed-img.zip
```

signed-img.zip contains all the .img files. You can use fastboot update signed-img.zip to mount them on the device.

4.3 How to Generate a Diff OTA Package

In the previous section, you will find:

```
evk_6sl_eink_target_files-eng.xxx.zip
```

This file is a full package of your current build image. Save this file in a directory. For example:

```
mkdir ~/release-1
```

```
cp out/dist/evk_6sl_eink_target_files-eng.xxx.zip ~/release-1
```

After some time, if you found a bug and if you want to generate a new release package, do the same operation to make a "dist" build.

Save it to a new location, for example, ~/release-2

```
mkdir ~/release-2
```

```
cp out/dist/evk_6sl_eink_target_files-eng.xxx.zip ~/release2
```

To generate a diff package:

```
cd mydroid; # you must in your android root dir to do this command
```

```
./build/tools/releasetools/ota_from_target_files -i
~/release-1/evk_6sl_eink_target_files-eng.xxx.zip \
```

```
~/release-2/evk_6sl_eink_target_files-eng.xxx.zip ~/diff-from-release-1-to-2.zip
```

~/diff-from-release-1-to-2.zip is a diff package between release 1 and release 2.

In my case, I changed a kernel commit, and a framework/base.

The zip package only 4.5M.

You can refer to the next section for how to upgrade your system by using this package.

4.4 How to Use the OTA Package

Update by Android fastboot

NOTE

fastboot is not avaliabe for ARM2 platform.

Perform the command below on your PC:

```
$ fastboot update update.zip
```

Reboot board to recovery mode.

```
u-boot> fastboot
```

Then, connect the USB cable.

Update by Android recovery (manually)

You can copy the ota.zip or the diff-ota.zip in the previous example to your SD card, and update your system to recovery. Choose menu to apply this update.zip to perform the upgrade.

Update by Android recovery (automatically)

You can copy the update.zip to /cache dir and then do the the following:

```
busybox cp /sdcard/update.zip /cache/
mkdir /cache/recovery/
echo "--update_package=/cache/update.zip" > /cache/recovery/command
reboot recovery
```

The recovery will auto apply the command and install this update package.

```
Update by Android framework API (OTA App)
```

In the previous example, you can see that the recovery system applies the update package that is generated by 5.1, 5.2, 5.3, for an OTA App. You only need to do the following:

- Check if an update package is available and notify the user to do the upgrade.
- Download the update zip package to /cache/ dir.
- Call recovery API to do the upgrade.

The following are the two APIs that can verify and install the update package:

```
import android.os.RecoverySystem;
          RecoverySystem.verifyPackage();
\tabRecoverySystem.installPackage();
```

4.5 OTA Application

In this release, we have added a reference application that can do OTA job, that will check the update from server, and download and install the package to upgrade the software in the board without any external tools.

The application is under packages/apps/fsl_imx_demo/FSLOta/

It is a dialog activity, and can be enabled in **Settings->About->Additional System Update menu**.

The server side of this OTA Application is a common Http server, eg, lighttpd, Apache.

Shown below is a `tree` command output of my sample ota site:

```
$ tree ~/ota-site                                  #the root dir of http server
/home/user/ota-site/
`-- evk_6sl_eink                                        # the product 's name as directory name
         |-- build.prop                                  # for checking version
         `-- evk_6sl_eink.ota.zip                      # name is product_name.ota.zip, the OTA application will download this package, this should be a update.zip
```

Figure 3. Tree Command Output

You can check the comment in the right side.

After you setup the Http server, you can change the /system/etc/ota.conf of server ip and server port.

The evk_6sl_eink.ota.zip is the output from section 5.3.

The build.prop is copy from this build's system/build.prop. The OTA application will download this file, parse it to get the build time, and compare with the build time of itself.

If the server build is newer, it will show the version information and package size and prompt user to upgrade The user can "upgrade" the server by clicking the "Update" button.

Then you can see the download starting.

In this example, this application is now downloading evk_6sl_eink.ota.zip package.

After downloading is finished, the title will change to "Verify package". During this time, it is actually doing RecoverySystem.verifyPackage() API to verify whether the package was complete, such as a MD5 checksum checking. It will also check whether the key chain in package aligns to the key chain in the device.

After the verifying is finished, it will call RecoverySystem.installPackage() API to install the package. This was generally similar to Section 5.4. It will write a recovery command and store it into /cache/recovery/command file, and reboot.

After reboot, the system will boot to recovery mode. After it installs the update package, you should see the "Android Robot" spinning on the screen. If you meet an error, you should see the "Error Robot", also it will stop spinning. Press "MENU" button to see the log output on the screen.

4.6 General Reason for Failure to Upgrade

If you failed with install update package in the recovery system, you can click "MENU" key to switch menu mode, which can show the error log on the screen. This way, you can see why the update package has failed. The most common reasons are:

- Keychain in update.zip and recovery system do not align.
- Product name is not aligned.

4.7 Customize the Update Script to Update U-Boot

About U-boot upgrade:

Because Android will only upgrade the boot.img, system.img, and the recovery partitions, the auto generated update package does not support upgrade bootloader. If you need to upgrade the bootloader, you need to modify the update package and do the sign job manually.

First, you need unzip the update.zip.

Then, modify the updater_script by implementing the following steps:

For U-Boot upgrade to NOR flash, you can refer to this script:

```
ui_print("writting u-boot...");
write_raw_image("u-boot.bin", "/dev/mtd0");
show_progress(0.1, 5);
```

For U-Boot upgrade for eMMC storage, since U-Boot is perhaps stored in the "boot partition" of eMMC, you need to perform some sys file operation before dd:

```
# Write u-boot to 1K position.
# u-boot binary should be a no padding uboot!
# For eMMC(iNand) device, needs to unlock boot partition.
ui_print("writting u-boot...");
package_extract_file("files/u-boot-no-padding.bin", "/tmp/u-boot-no-padding.bin");
sysfs_file_write("class/mmc_host/mmc0/mmc0:0001/boot_config", "1");
simple_dd("/tmp/u-boot-no-padding.bin", "/dev/block/mmcblk0", 1024);
sysfs_file_write("class/mmc_host/mmc0/mmc0:0001/boot_config", "8");
show_progress(0.1, 5);
```

Then, you need to resign the update package by using the following command:

```
$ make_update_zip.sh ~/mydroid ~/update-dir
```

5 Storage Customization

Multi-storages are supported in this release:

- SD card in MMC/SD connector (i.MX 6SoloLite EVK has three SD slots: one for boot up, the others for Wi-Fi card or SD card).
- UDISK connected to the board through the USB port.

They are mounted to /sdcard, /extsd and /udisk in Android rootfs.The storage can be accessed through MTP by PC host.

To support more disks, please change the following files according to the example below:

- device/fsl/imx6/evk/vold.fstab, or the path according to the board used, for example:

```
dev_mount extsd  /mnt/extsd auto /devices/platform/sdhci-esdhc-imx.2/mmc_host/mmc1
```

- device/fsl/imx6/evk/overlay/framework/base/core/res/res/xml/storage_list.xml, for example:

```
<storage android:mountPoint="/mnt/extsd"
android:storageDescription="@string/storage_usb"
android:removable="true"
android:primary="false" />
```

6 Customization on the Boot Logo

If there is a file called initlogo.rle under the root folder, Android displays it while booting. Complete the following steps to create a rle compressed picture from a png file.

1. Find an image.
2. Edit it with an editing suite and scale it to 480x640.
3. After scaling it, convert the colorspace to 256 colors (8-bit).
4. Save it as a PNG without alpha channel/transparency.
5. Use the convert tool from the ImageMagick toolkit: convert -depth 8 splash rgb:splash.raw
6. Compile the Android tool in mydroid/build/tools/rgb2565(gcc -O2 -Wall -Wno-unused-parameter -o rgb2565 to565.c)
7. Run the conversion command: rgb2565 < splash.raw > splash.raw565.
8. Copy splash.raw565 to mydroid/out/target/product/evk_6sl_eink/root/initlogo.rle.

7 Connectivity Customization

This section describes how to port connectivity parts, including WI-FI, on the EVK board.

7.1 About Wi-Fi

Atheros AR6103 is the default Wi-Fi module supported in this package.

Wi-Fi code topology is shown below:

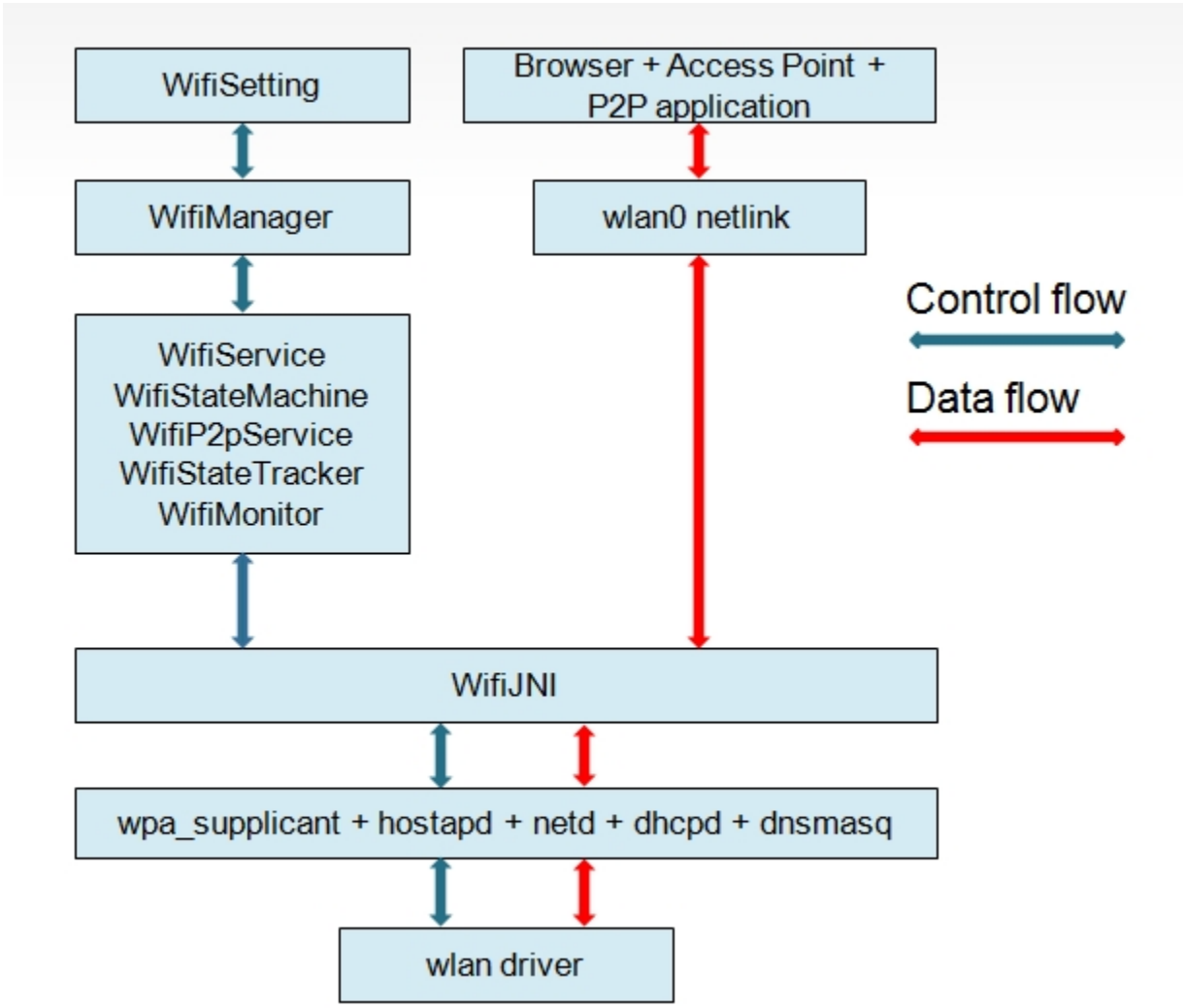

Figure 4. Wi-Fi Code Topology

1. Wi-Fi control flow

   The Wi-Fi setting app supports the control of STA, AP and P2P profiles. You can customize it through modifying the following values in required_hardware.xml which is located in myandroid/device/fsl/imx6/evk/:

   ```
   <feature name="android.hardware.wifi" />
   <feature name="android.hardware.wifi.direct" />
   ```

   The Wi-Fi App will realize enable/disable, scan, discover, connect/disconnect calls through Wi-Fi services in framework. For this part, no changes are needed. You can use Google code only. Through JNI, Wi-Fi services will communicate with wpa_supplicant or hostpad.

   In STA and P2P case, wpa_supplicant plays an important part in making a bridge between Wi-Fi framework and driver. Due to the default wpa_supplicant from ASOP is based on Broadcom Wi-Fi driver, you have to make some changes for other vendor's Wi-Fi module. Take Atheros Wi-Fi module as an example, you can directly use the wpa_supplicant from Atheros to replace the default wpa_supplicant_8 in ASOP. But for Broadcom or CSR Wi-Fi module, you can directly use the default wpa_supplicant_8 without any change.

2. How to port Wi-Fi to ICS?

   If you should take Atheros AR6103 Wi-Fi as an example, you would follow the steps to

port Ath6kl-3.2 SDK.

3. Fetch Ath6kl-3.2 SDK package from Atheros, like

   AndroidReleaseAth6kl-3.2-CS-Patch4.tgz

4. decompress the .tgz package. There are mainly three parts:Generic_Packages;

   Firmware_Package;Proprietary_tools.

5. In Generic_Packages, Wi-Fi driver named compact-wireless and wpa_supplicant are included. Copy compact-wireless to /ssytem/wlan/atheros directory. Then, add compile Android.mk file. To edit this file, take ours as an example: Copy wpa_supplicant to replace the wpa_supplicant_8 in external/ directory. You need define several micro variables to control how to compile these two directories.

   In 13.5.0-GA, we define them as shown below:

```
BOARD_WLAN_DEVICE             :=
ar6003
BOARD_HAS_ATH_WLAN          :=
true
BOARD_WLAN_ATHEROS_SDK    := system/wlan/atheros/compat-
wireless
BOARD_WPA_SUPPLICANT_DRIVER :=
NL80211
BOARD_HOSTAPD_DRIVER            :=
NL80211
WPA_SUPPLICANT_VERSION         :=
VER_0_9_ATHEROS
HOSTAPD_VERSION                      :=
VER_0_9_ATHEROS
WIFI_DRIVER_MODULE_PATH        := "/system/lib/modules/
ath6kl_sdio.ko"
WIFI_DRIVER_MODULE_NAME        :=
"ath6kl_sdio"
WIFI_DRIVER_MODULE_ARG          := "suspend_mode=3
ath6kl_p2p=1"
WIFI_DRIVER_P2P_MODULE_ARG    := "suspend_mode=3 ath6kl_p2p=1
                                                debug_mask=0x2413"
WIFI_SDIO_IF_DRIVER_MODULE_PATH :="/system/lib/modules/
cfg80211.ko"
WIFI_SDIO_IF_DRIVER_MODULE_NAME :=
"cfg80211"
WIFI_SDIO_IF_DRIVER_MODULE_ARG  :=
""
WIFI_COMPAT_MODULE_PATH        := "/system/lib/modules/
compat.ko"
WIFI_COMPAT_MODULE_NAME        :=
"compat"
WIFI_COMPAT_MODULE_ARG          :=
""
WIFI_TEST_INTERFACE                  := "sta"
```

6. Firmware_Package includes the firmware needed by Wi-Fi module. You can add them in device/fsl-proprietary/ath6k/firmware.
7. Proprietary_tools include the tools source codes for Wi-Fi. Because of a license issue we published them as binary in device/fsl-proprietary/ath6k/tools.
8. Modify HAL Wi-Fi code to adapt Atheros Wi-Fi SDK. To see how to change Wi-Fi HAL, you can refer to the file wifi_ath.c in hardware/libhardware_legacy/wifi. Since the Ath6kl driver is using nl80211 mode, cfg80211 will be changed. Therefore, you need to config the cfg80211 into module in kernel. Then insmod cfg80211.ko before loading Wi-Fi driver.
9. Change certain Wi-Fi settings in init.rc.

```
service wpa_supplicant /system/bin/wpa_supplicant \
    -Dnl80211 -puse_p2p_group_interface=0  -e/data/misc/wifi/entropy.bin
    class late_start
    disabled
    oneshot
```

```
service dhcpcd_wlan0 /system/bin/dhcpcd -ABKL
    class late_start
    disabled
    oneshot
mkdir /data/system 0775 system system
mkdir /data/system/wpa_supplicant 0771 wifi wifi
chmod 0771 /data/system/wpa_supplicant
```

10. How to enable INTEL PCIe Wi-Fi?

    Intel PCIe Wi-Fi is supported in this release, but is disabled by default. If you want to enable it, you can change the BOARD_WLAN_VENDOR to INTEL in EVKBoardConfigComm.mk.

    There are three versions of wpa_supplicant in external directory. Among them, wpa_supplicant_7 is for INTEL PCIe Wi-Fi. If you define BOARD_WLAN_VENDOR to INTEL, this version of wpa_supplicant will be compiled.

    Also in HAL, wifi_intel.c will be compiled if BOARD_WLAN_VENDOR is defined as INTEL.

    PCIe subsystem is not enabled in default kernel. In order to enable INTEL PCIe Wi-Fi, you need to run "make menuconfig".

    ```
    Kernel configuration:
    * -> System Type
           -> Freescale MXC Implementations
    Select the PCI Express support.
    ```

    Then, config intel Wi-Fi driver:

    ```
    Generic IEEE 802.11 Networking Stack (mac80211) used by Wi-Fi devices
    Symbol: MAC80211 [=y]
    Type  : tristate
    Prompt: Generic IEEE 802.11 Networking Stack (mac80211)
      Defined at net/mac80211/Kconfig:1
      Depends on: NET [=y] && WIRELESS [=y] && CFG80211 [=y]
      Location:
        -> Networking support (NET [=y])
          -> Wireless (WIRELESS [=y])
    Intel iwl4965 or iwl6300 card driver
    Symbol: IWL4965 [=y]
    Type  : tristate
    Prompt: Intel Wireless Wi-Fi 4965AGN (iwl4965)
      Defined at drivers/net/wireless/iwlegacy/Kconfig:65
      Depends on: NETDEVICES [=y] && WLAN [=y] && PCI [=y] && MAC80211 [=y]
      Location:
        -> Device Drivers
          -> Network device support (NETDEVICES [=y])
    ```

    Finally, you need to add a compile rule in /system/wlan to compile the INTEL Wi-Fi driver.

8 Customization of the MFGTool

MFGTool can be used to download firmware into i.MX board. To use this tool, please refer to "Android Quick Start Guide" and "User Guide".

NOTE

MFGTool v2 uses ucl2.xml and cfg.ini to configure. It does no longer uses GUI to configure which profile it is using.

Below is the description of the MFGTool directory tree (output of 'tree' command with comments):

```
      |-- Document [ Document of MFGTool ]
|-- Drivers [ Driver of MFGTool USB Protocol ]
|  `-- iMX_BulkIO_Driver
```

```
| |-- amd64
| `-- i386
`-- Profiles            [ Profile includes download
  |                       profile with chip name]
  `-- MX6Q Linux Update [ This directory contains Linux
    |                     Update profile]
    |-- OS Firmware     [ uboot and kernel under this dir
    | |                   is used by MFGTooll Firmware ]
    | |-- files
    | | |-- android     [ all android image you want to
    | | | |               download in this dir ]
    | | | |-- boot.img
    | | | |-- recovery.img
    | | | |-- system.img
    | | | `-- u-boot.bin
    | | |-- rootfs.tar.bz2
    | | |-- u-boot-mx6sl-arm2.bin
    | | |-- u-boot-mx6sl-evk.bin
    | | `-- uImage
    | |-- initramfs.cpio.gz.uboot [ the rootfs running by MFGTool,
    | |                             communicating with Host]
    | |-- mksdcard-android.sh.tar [ Android partition script ]
    | |-- mksdcard.sh.tar         [ not used by android ]
    | |-- u-boot-mx6sl-evk.bin
    | |-- u-boot-mx6sl-arm2.bin   [ * board's uboot with mfg
    | |                               profile ]
    | |-- u-boot-mx6sl.bin
    | |--
    | |-- uImage                  [ * board's uboot with updater
    | |                               profile ]
    | `-- ucl2.xml1                     [ * script of download ]
    `-- player.ini
```

After you have taken a look at MFGTool directory, consider an example of how to customize a MFGTool firmware. The board is i.MX 6SoloLite EVK in this example. You can change the name of U-Boot as needed.

First, you will notice these images:

- Profiles/MX6SL Linux Update/OS Firmware/u-boot-mx6sl-evk.bin
- Profiles/MX6SL Linux Update/OS Firmware/uImage

You need to replace these two images if you want to download images into your board.

The U-Boot and kernel images are **NOT** the images you want to download into the board. They are special images, helpers, used to communicate with MFGTool and download images into your board. The images you want to download are under **Profiles/MX6SL Linux Update/OS Firmware/files/android/** directory.

For special U-Boot:

You need to build bootload with mfg profile:

```
make distclean
make mx6sl-evk_mfg_config
```

For special uImage, you need to build kernel with config shown below:

```
make imx6s_updater_defconfig
make uImage -j4
```

Then, replace these two output images with original images.

You should make sure that you have enabled your BOARD in kernel and U-Boot configure file.

9 Customization of the USB Gadget Product ID and Vendor ID

In ICS release, we use Google's VendorID (0x18d1) and ProductID (0x0d02) for USB gadget functions. This makes it easier for the customer to use the ADB function by default for Google windows driver in Android SDK. If you want to change the ID to your own, please perform the following steps:

1. Change the vid/pid in init.usb.rc

   Android ICS exports the USB gadget functions pid/vid configuration in the android gadget driver by sysfs. User can configure them in the init.usb.rc. Therefore, you need to change all the pid/vid in the init.usb.rc file to let system set the correct pid/vid when doing gadget functions' switch. For example: change all the /sys/class/android_usb/android0/ idVendor and idProduct to your vid/pid in the **device/fsl/imx6/etc/init.usb.rc**:

```
# USB massstorage configuration with
ADB

on
property:sys.usb.config=mass_storage,adb


          write /sys/class/android_usb/android0/enable
0

          write /sys/class/android_usb/android0/idVendor <your
vid>

          write /sys/class/android_usb/android0/idProduct <your
pid>

          write /sys/class/android_usb/android0/functions
$sys.usb.config

          write /sys/class/android_usb/android0/enable
1

          start
adbd


          setprop sys.usb.state
$sys.usb.config



# USB massstorage
configuration


on
property:sys.usb.config=mass_storage


          write /sys/class/android_usb/android0/enable
0

          write /sys/class/android_usb/android0/idVendor <your
vid>

          write /sys/class/android_usb/android0/idProduct <your
pid>
```

```
          write /sys/class/android_usb/android0/functions
$sys.usb.config

          write /sys/class/android_usb/android0/enable
1

          setprop sys.usb.state $sys.usb.config
…..
```

2. Update the HOST configuration

   This configuration change is mainly for ADB function. The MTP/PTP function does not need any changes on the HOST. The RNDIS needs to update the vid/pid in the tool/tetherxp.inf file from the release package.

   For Windows PC:

   - Download the Android SDK.
   - Update the ADB configuration to scan for your pid.
   - Run the SDK's tools to generate a configure file.
   - android-sdk-windows\tools\android.bat update adb
   - Modify the files:
   - X:\Profile\<your account>\.android\adb_usb.ini, to add your vendor id, e.g: 0x18d1:

```
# ANDROID 3RD PARTY USB VENDOR ID LIST -- DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x18d1
```

   - Restart the ADB server.
   - adb kill-server
   - adb start-server

For Linux PC

- Download the Android SDK.
- Update the ADB configuration to scan for your pid.
- Run the SDK tools to generate a configure file.
- android-sdk-linux_86/tools/android update adb
- Modify the files:~/.android/adb_usb.ini, to add your vendor id, e.g. 0x18d1:

```
# ANDROID 3RD PARTY USB VENDOR ID LIST -- DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x18d1
```

- Create a new udev rule file under the PC's /etc/udev/rules.d/ named: imx-android.rules. Fill in the following line into the file:

```
SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", MODE="0666"
```

- Change the new udev rule file's permission.
- chmod a+r /etc/udev/rules.d/imx-android.rules
- Connect the Android Device.
- Restart the ADB server.
- adb kill-server
- adb start-server

10 EPDC Splash Screen Support

EPDC splash screen is turned off by default. If you need to show a splash screen on EPD panel, you should add it yourself by following the instructions below:

1. In config file, enable splash screen. Find the following string, remove the comments indicator. After you have done so, the splash screen feature should be enabled.

```
/*
#define CONFIG_SPLASH_SCREEN
*/
```

2. Program the waveform file to a SD card at offset 0x100000.
   - Identify the EPDC waveform file from the Linux kernel firmware directory that is the best match for the panel you are using. For the DC2/DC3 boards, that would be the waveform file /firmware/imx/epdc_E060SCM.fw.ihex. You should request ihex2bin.py script from Freescale.
   - Use ihex2bin.py to get the waveform binary data.
   - Program wv_data.bin to SD card at offset 0x100000. You can use the ATK tool or dd command in Linux for this:

```
root@freescale /$ dd if=./wv_data.bin of=/dev/mmcblk0 bs=1024 seek=1024
```

NOTE

You can also program wv_data.bin to a different offset on the SD card. However, you need to modify the CONFIG_WAVEFORM_FILE_OFFSET in config file and rebuild U-Boot. Currently, we only support waveform data file on sd/mmc card. If you need to access another memory device for waveform data, such as SPI NOR, CONFIG_WAVEFORM_FILE_IN_MMC should be removed and CONFIG_WAVEFORM_FILE_IN_SF should be added. Codes to access SPI NOR should be added in function setup_waveform_file().

3. Boot your board and you will see the splash screen on EPD panel.

11 Using a Custom Waveform File

To ensure the optimal E Ink display quality, use a waveform file specific to E Ink panel being used. The raw waveform file type (.wbf) requires conversion to a format that can be understood and read by the EPDC. This conversion script is not included as part of the BSP, so please contact Freescale to acquire this conversion script.

Once the waveform conversion script has been run on the raw waveform file, the converted waveform file should be renamed so that the EPDC driver can find it and load it. The driver is going to search for a waveform file with the name "imx/epdc_[panel_name].fw", where panel_name refers to the string specified in the fb_videomode `name` field. For example, if the panel is named "E60_ABCD", then the converted waveform file should be named epdc_E60_ABCD.fw.

The firmware script firmware.sh (lib/udev/firmware in the Linux root file system) contains the search path used to locate the firmware file. The default search path for firmware files is /lib/firmware;/usr/local/lib/firmware. A custom search path can be specified by modifying firmware.sh. You'll need to create an imx directory in one of these paths and add your new epdc_[panel_name].fw file there.

NOTE

If the EPDC driver is searching for a firmware waveform file that matches the names of one of the default waveform files (see preceding chapter), it will choose the default firmware files that are built into the BSP over any firmware file that has been added in the firmware search path. Thus, if you leave the BSP so that it builds those default firmware files into the OS image, be sure to use a panel name other than those associated with the default firmware files, since those default waveform files will be preferred and selected over a new waveform file placed in the firmware search path.

12 How to Trace the Low Level Malloc

Libc has a malloc debug framework for different debuggers. Each debugger takes as a library, and overrides the default malloc/free/calloc/realloc/, hooked before calling the real functions.

NOTE

This tip assumes that you are working with an eng or userdebug build of the platform, not on a production device.

Trace the low level malloc/free in bionic.

Bionic has a malloc debugger called leak debugger, which can record all the malloc/free in low level. Developers can use ddms on host to check each block of memory on heap by malloc. ddms supports convert caller function address to name conver. That makes it easy for us to check which component and which function is allocated to how much memory.

You can turn on memory tracking with debug level 1:

```
$ adb shell setprop libc.debug.malloc 1

$ adb shell stop

$ adb shell start
```

You need to restart the runtime so that zygote and all processes launched from it are restarted with the property set. All Dalvik processes have memory tracking turned on. You can look at these with DDMS, but first you need to turn on its native memory UI:

- Open ~/.android/ddms.cfg
- Add a line "native=true"

Upon relaunching DDMS and selecting a process, you can switch to the new native allocation tab and populate it with a list of allocations. This is especially useful for debugging memory leaks.

NOTE

To solve the module symbols, please export two env on HOST:

```
$ export PATH=$PATH:<android src>/prebuilt/linux-x86/toolchain/arm-linux-androideabi-4.4.x/
bin

$ export ANDROID_PRODUCT_OUT=<android src>/out/target/product/<platform>
```

Expose the memory leak

In this release he Electric Fence 2.2.0 has been ported to Android. It also has a memory debugger tool as the leak debugger above.

It helps you detect two common programming bugs: software that overruns the boundaries of a malloc() memory allocation, and software that touches a memory allocation that has been released by free(). It will dump the call stack and mmap of the process, if the malloc/free is not called with correct parameters. It will also make a segment fault, when applications want to access the address which is freed.

The usage of efence is almost the same as above. We define its debug level to 15:

```
$ adb shell setprop libc.debug.malloc 15
```

After you set this property, you can run your applications. If there is any memory leakage, logcat will show that information.

Example:

Access the memory, which has been freed already:

```
root@android:/data # setprop libc.debug.malloc 15
I/libc    ( 4136): setprop using MALLOC_DEBUG = 1 (leak checker)
root@android:/data # ./memtest
I/libc    ( 4138): ./memtest using MALLOC_DEBUG = 15 (efence)
F/libc    ( 4138): Fatal signal 11 (SIGSEGV) at 0x4015bff4 (code=2)
I/DEBUG   ( 3856): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
I/DEBUG   ( 3856): Build fingerprint:
freescale/evk_6sl_eink/evk_6sl:4.0.4/13.5.0-rc1/eng.b03824.20120711.11133
8:eng/test-keys'
I/DEBUG   ( 3856): pid: 4138, tid: 4138  >>> ./memtest <<<
I/DEBUG   ( 3856): signal 11 (SIGSEGV), code 2 (SEGV_ACCERR), fault addr 4015bff4
I/DEBUG   ( 3856):  r0 00000000  r1 00000002  r2 40151c6c  r3 00000000
I/DEBUG   ( 3856):  r4 4015bff4  r5 bec75a54  r6 00000001  r7 bec75a5c
I/DEBUG   ( 3856):  r8 00000000  r9 00000000  10 00000000  fp 00000000
I/DEBUG   ( 3856):  ip 00000000  sp bec75a20  lr 40133f8f  pc 0000a554  cpsr 60000010
I/DEBUG   ( 3856):  d0  203f810033915fe5  d1  0000000000000000
I/DEBUG   ( 3856):  d2  0000000000000000  d3  0000000000000000
I/DEBUG   ( 3856):  d4  0000000000000000  d5  0000000000000000
I/DEBUG   ( 3856):  d6  0000000000000000  d7  204cb48d00000000
I/DEBUG   ( 3856):  d8  0000000000000000  d9  0000000000000000
I/DEBUG   ( 3856):  d10 0000000000000000  d11 0000000000000000
I/DEBUG   ( 3856):  d12 0000000000000000  d13 0000000000000000
I/DEBUG   ( 3856):  d14 0000000000000000  d15 0000000000000000
I/DEBUG   ( 3856):  d16 41c0265a46a47ae1  d17 3f50624dd2f1a9fc
I/DEBUG   ( 3856):  d18 41c9c8aff2800000  d19 0000000000000000
I/DEBUG   ( 3856):  d20 0000000000000000  d21 0000000000000000
I/DEBUG   ( 3856):  d22 0000000000000000  d23 0000000000000000
I/DEBUG   ( 3856):  d24 0000000000000000  d25 0000000000000000
I/DEBUG   ( 3856):  d26 0000000000000000  d27 0000000000000000
I/DEBUG   ( 3856):  d28 0000000000000000  d29 0000000000000000
I/DEBUG   ( 3856):  d30 0000000000000000  d31 0000000000000000
I/DEBUG   ( 3856):  scr 00000010
I/DEBUG   ( 3856):
I/DEBUG   ( 3856):          #00  pc 0000a554  /data/memtest
I/DEBUG   ( 3856):          #01  pc 00016834  /system/lib/libc.so (__libc_init)
I/DEBUG   ( 3856):
I/DEBUG   ( 3856): code around pc:
I/DEBUG   ( 3856): 0000a534 e3a0000a ebfff8d1 e3a01001 e1a00004  ................
I/DEBUG   ( 3856): 0000a544 e5c4100a ebfff8d9 e3560001 e3a03000  ..........V..0..
I/DEBUG   ( 3856): 0000a554 e5c43000 0a000064 e59f21f0 e2857004  .0..d....!...p..
I/DEBUG   ( 3856): 0000a564 e5954004 e08f1002 e1a00004 ebfff8c0  .@..............
I/DEBUG   ( 3856): 0000a574 e3500000 0a00003f e59fc1d4 e1a00004  ..P.?...........
I/DEBUG   ( 3856):
I/DEBUG   ( 3856): code around lr:
I/DEBUG   ( 3856): 40133f6c 68200701 0001f020 454e1046 2e00d018  .. h ...F.NE....
I/DEBUG   ( 3856): 40133f7c 2102da01 1c81e000 43394338 f7eb4622  ...!....8C9C"F..
I/DEBUG   ( 3856): 40133f8c 4605ed56 d1ec2800 da062e00 0101f008  V..F.(..........
I/DEBUG   ( 3856): 40133f9c f06f4620 f7ea4200 4628e8d4 87f0e8bd   Fo..B....(F....
I/DEBUG   ( 3856): 40133fac eff0f7e9 6003234b 30fff04f bf00e7f6  ....K#.`O..0....
I/DEBUG   ( 3856):
I/DEBUG   ( 3856): memory map around addr 4015bff4:
I/DEBUG   ( 3856): 40150000-4015a000
I/DEBUG   ( 3856): 4015a000-4015d000
I/DEBUG   ( 3856): 4015d000-4015e000
I/DEBUG   ( 3856):
I/DEBUG   ( 3856): stack:
I/DEBUG   ( 3856):     bec759e0  00000000
I/DEBUG   ( 3856):     bec759e4  00000000
I/DEBUG   ( 3856):     bec759e8  00000000
I/DEBUG   ( 3856):     bec759ec  4012090f  /system/lib/libefence.so
I/DEBUG   ( 3856):     bec759f0  4015a014
I/DEBUG   ( 3856):     bec759f4  00002000
I/DEBUG   ( 3856):     bec759f8  00000004
I/DEBUG   ( 3856):     bec759fc  40120df1  /system/lib/libefence.so
I/DEBUG   ( 3856):     bec75a00  4015bff4
I/DEBUG   ( 3856):     bec75a04  bec75a54  [stack]
I/DEBUG   ( 3856):     bec75a08  00000001
I/DEBUG   ( 3856):     bec75a0c  bec75a5c  [stack]
I/DEBUG   ( 3856):     bec75a10  00000000
I/DEBUG   ( 3856):     bec75a14  400d9167  /system/lib/libc.so
```

```
I/DEBUG   ( 3856):     bec75a18  df0027ad
I/DEBUG   ( 3856):     bec75a1c  00000000
I/DEBUG   ( 3856): #00 bec75a20  00008924  /data/memtest
I/DEBUG   ( 3856):     bec75a24  bec75a54  [stack]
I/DEBUG   ( 3856):     bec75a28  00000001
I/DEBUG   ( 3856):     bec75a2c  bec75a5c  [stack]
I/DEBUG   ( 3856):     bec75a30  00000000
I/DEBUG   ( 3856):     bec75a34  400d9837  /system/lib/libc.so
I/DEBUG   ( 3856): #01 bec75a38  00000000
I/DEBUG   ( 3856):     bec75a3c  00000000
I/DEBUG   ( 3856):     bec75a40  00000000
I/DEBUG   ( 3856):     bec75a44  00000000
I/DEBUG   ( 3856):     bec75a48  00000000
I/DEBUG   ( 3856):     bec75a4c  b00046ef  /system/bin/linker
I/DEBUG   ( 3856):     bec75a50  00000001
I/DEBUG   ( 3856):     bec75a54  bec75b79  [stack]
I/DEBUG   ( 3856):     bec75a58  00000000
I/DEBUG   ( 3856):     bec75a5c  bec75b83  [stack]
I/DEBUG   ( 3856):     bec75a60  bec75b8f  [stack]
I/DEBUG   ( 3856):     bec75a64  bec75ba2  [stack]
I/DEBUG   ( 3856):     bec75a68  bec75bc5  [stack]
I/DEBUG   ( 3856):     bec75a6c  bec75bde  [stack]
I/DEBUG   ( 3856):     bec75a70  bec75c08  [stack]
I/DEBUG   ( 3856):     bec75a74  bec75c20  [stack]
I/DEBUG   ( 3856):     bec75a78  bec75c57  [stack]
I/DEBUG   ( 3856):     bec75a7c  bec75c61  [stack]
I/BootReceiver( 3551): Copying /data/tombstones/tombstone_00 to DropBox (SYSTEM_TOMBSTONE)
D/dalvikvm( 3551): GC_CONCURRENT freed 398K, 10% free 8805K/9735K, paused 3ms+5ms
[2] + Segmentation fault   ./memtest
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AAUG
Rev. 13.5.0_GA
11/2012