GoDroid

Programming Guide

Ext-Recovery

v1.0

Table of Contents

List of Tables

List of Figures

Preface

Overview

This manual specifies the general overview and procedures to develop and implement the Recovery system. This manual is organized into the following chapters.

- **Chapter 1: Introduction**

This chapter provides details on implementation of the recovery system.

- **Chapter 2: Programming Guide**

This chapter gives compact description on how to program recovery system.

- **Chapter 3: API Description**

This chapter provides information on the related API.

Audience

This manual is applicable for the users who wish to learn how to carry out adding/porting new sensor in GoDroid. Readers of this manual are assumed to have certain knowledge and background on Android embedded development.

Applicable Products

This manual is applicable for the GoWarrior TIGER Board.

Reference Documents

GoWarrior_GoDroid_Application Notes_Flash Partition

Conventions

Typographical Conventions

Item	Format
codes, keyboard input commands, file names, equations, and math	`Courier New, Size 10.5`
Variables, code variables, and code comments	*`Courier New, Size, Italic`*
Menu item, buttons, tool names	Ebrima, Size 10.5, Bold e.g. Select USB Debugging
Screens, windows, dialog boxes, and tabs	**Ebrima, Size 10.5, Bold** **Enclosed in double quotation marks** **e.g. Open the "Debug Configuration" dialog box**

Table 1. Typographical Conventions

Symbol Conventions

Item	Description
⚠ *Caution*	Indicates a potential hazard or unsafe practice that, if not avoided, could result in data loss, device performance degradation, or other unpredictable results.
Note	Indicates additional and supplemental information for the main contents.
Tip	Indicates a suggestion that may help you solve a problem or save your time.

Table 2. Symbol Conventions

How to Contact Us

Submit all your comments and error reports to the author at:

info@gowarriorosh.com

Tel: +886-2-8752-2000

Fax: +886-2-8751-1001

For questions regarding GoWarrior, contact our support team at the email listed below:

support@gowarriorosh.com

1 Introduction

Recovery partition is an independent system based on Linux. It contains kernel, see, and rootfs. The recovery partition can be considered as an alternative boot partition that lets you boot the device into a recovery mode for performing upgrade firmware, factory reset, clear data partition, and so on.

1.1 Recovery System Structure

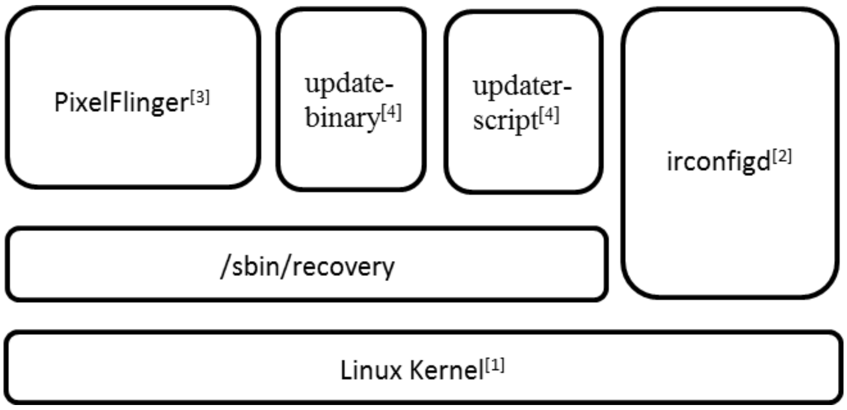


Figure 1. Recovery System Structure

Notes:

- Recovery runs on Linux Kernel, and starts as an Android App. Recovery program (/sbin/recovery) will be started by service in init.XXX.rc.
- irconfigd: Button pressing detecting program. You can find this file under recovery_system/bin/ (/recovery_system is added in recovery's rootfs, files needed for running the third-party programs can be found here). The structure of /recovery_system is as follows:

```
|----recovery_system
    |----bin
            |----ash            //A Linux command interpreter
```

```
       |----irconfigd      //Remote controller and front panel
                           buttons detecting program
           |----linker         //The dynamic linker, executable
                               programs under recovery_system/
                               and /sbin/recovery folder will
                               use this linker
           |----reboot→toolbox
           |----toolbox
   |----lib
           |----libadr_hld.so  //ALI driver library
           |----libc.so
           |----libcutils.so      //Android basic library, the source
                               code: system/core/libcutils/
           |----libblog.so
           |----libm.so
           |----libselinux.so
           |----libstdc++.so
   |----libusbhost.so  //Android USB related libraries,
 source code:
 system/core/libusbhost/
   |----xbin
           |----ash→busybox
           |----busybox
           |----ls→busybox
           |----rm→busybox
           |----sync→busybox
```

- PixelFlinger is a software renderer for OpenGL ES in Android system. It provides OpenGL ES engine a series of basic drawing functions. These features include defined color format pixel location, stipple line, and draw a rectangle and triangle filling texture. Please refer to 2.1.2.

PS: Pixelflinger's source code is located in *`system/core/libpixelflinger`. Header files are located in `system/core/include/libpixelflinger` and* `system/core/include/private/pixelflinger`.

- The updater-script should be written in Edify. The update-binary parses the updater-script and execute the commands. Refer to Chapter 4 for more details about making upgrade packages.

PS: update-binary's source code is located in `bootable/recovery/updater/`. `updater-script` will be automatically generated during building an upgrade package (refer to `build/tools/releasetools/ ota_from_target_files`).

1.2 Recovery Partitions

Partition	Description
Bootloader	Bootarea, loads and starts the kernel or recovery system.
bootloaderbak	Bootarea backup partition. If OTP signature function is enabled and bootloader partition is corrupted, the bootloaderbak partition will be loaded in bootrom.
bootargs	Boot parameter partition, containing partition table, cmdline and memmapping. This partition offset must be fixed in 2048th pages (NAND).
deviceinfo	Device information partition, containing sequence number, MAC address, hardware version and hdcpkey.
baseparams	System parameter partition, containing AVInfo and software version.
misc★	Upgrade flag partition, which directs the bootloader to load recovery partition or kernel partition.
recovery★	The recovery partition, which implements upgrade and factory reset.
recoverybak★	Recovery backup partition. If the recovery partition is damaged, the recoverybak partition will be loaded in bootloader.

Partition	Description
backup★	System backup partition, containing kernel and system partition image to restore system.
cache★	This is the partition where Android stores frequently accessed data, containing upgrade command, upgrade package and upgrade log.
bootmedia	Bootloader bootlogo/bootmedia.
kernel	Main+ramdisk+see+ae.
system	Android system (read only).
data	User data partition (read and write).

Table 3. Recovery Partition

The partition table above takes TIGER Board NAND Flash as an example. The partitions marked with the ★ symbol are added specifically for the recovery system.

1.3 Upgrade Types

1.3.1 Divided by Download Mode

According to method of downloading upgrade package, we can classify upgrade into the following types:

- IP upgrade

Download the upgrade package by internet to cache partition, and reboot into recovery.

- USB/SD upgrade

Select USB/SD Upgrade in Android System and reboot into recovery. Recovery will search update.zip in local USB/SD card and load it. The upgrade package must be located in the root directory of USB/SD, and the package name must be update.zip, otherwise the recovery will fail.

Possible extension: You can also load the upgrade package from USB/SD card to the cache partition first before reboot into recovery.

- TS OTA upgrade

The TS (Transport Stream) can trigger OTA (Over-The-Air) upgrade, and then reboot into recovery and download the upgrade package by TS.

Possible extension: You can also download the upgrade package to the cache partition by TS before reboot into recovery.

1.3.2 Divided by Upgrade Package Type

According to upgrade package type, upgrade can be classed into two kinds: full upgrade and incremental upgrade.

- Full Upgrade

When the upgrade package is a full upgrade package, the upgrade mode is referred as full upgrade. The upgrade package contains all the upgradable partition data at this moment. As it has relatively bigger size, data downloading in full upgrade mode will cost more.

Only with full upgrade can update the partition table. Namely, you can modify the partition table only by full upgrade.

- Incremental upgrade

When the upgrade package is an incremental upgrade package, the upgrade mode is referred to as incremental upgrade. As the incremental upgrade package has relatively smaller size, it is more suitable for IP upgrade. But the limitation is that the incremental package is only available for a specific version.

Suppose there is an incremental package called AB (upgrading from Version A to B), it can be upgraded to Version B properly only when the platform is Version A. If the platform is Version O, a version older than Version A, AB incremental package cannot be used. It can be upgraded to Version B only via the full package of Version B.

1.4 Trigger of Recovery System

There are several ways to trigger and enter recovery, some of which are descripted as below.

1.4.1 Triggered by Android APP

Triggering recovery via an Android APP is the most commonly used method by end users.

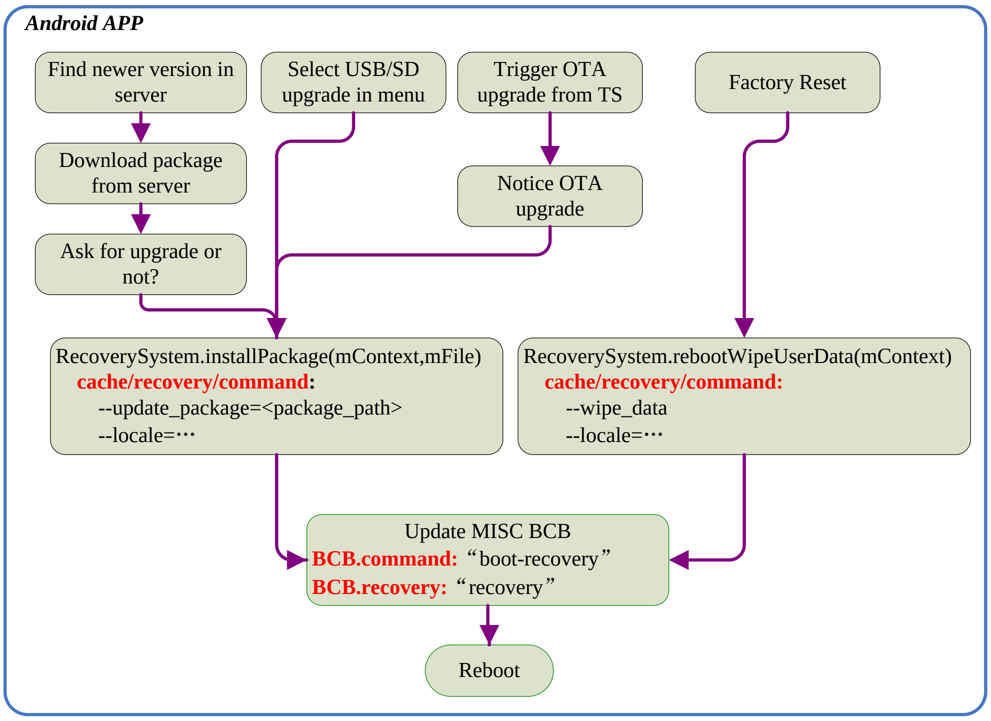

Android APP
Find newer version in server
Download package from server
Ask for upgrade or not?
Select USB/SD upgrade in menu
Trigger OTA upgrade from TS
Notice OTA upgrade
Factory Reset
RecoverySystem.installPackage(mContext,mFile)
cache/recovery/command:
--update_package=<package_path>
--locale=…
RecoverySystem.rebootWipeUserData(mContext)
cache/recovery/command:
--wipe_data
--locale=…
Update MISC BCB
BCB.command: “boot-recovery”
BCB.recovery: “recovery”
Reboot

Figure 2. Trigger Recovery via an Android APP

- Using RecoverySystem.installPackage to install an upgrade package.
- Using RecoverySystem.rebootWipeUserData to wipe data partition.

1.4.2 Force Trigger

Recovery will be forced to load when U-Boot has detected that any IR key has been pressed for a long time.

In this mode, recovery will first check whether a legal upgrade package exists in the USB/SD card. If not, it will then try to find upgrade package in cache partition. Once the upgrade package is found, upgrade package will be installed. Otherwise, the system will jump to recovery menu where users can select the next step.

Please refer to chapter 4.2.6: Forced to enter recovery configuration in "*GoWarrior_GoDroid Application Notes_Flash Partition* "that describes how to configure force trigger.

1.4.3 System Restore Trigger

Recovery will be forced to load when U-Boot has detected a specified GPIO, and restore system (Restoring can also be performed by force trigger and selection of the system restore function). System restore is different from factory reset

- System restore

Restore partition data backed up in backup partition to corresponding partition:

Clear data partition;

Clear cache partition;

Recover the default AVInfo.

- Factory reset

Clear data partition;

Clear cache partition;

Please refer to chapter 4.2.7: Trigger system restore configuration in "*GoWarrior_GoDroid Application Notes_Flash Partition*" that describes how to configure system restore trigger.

1.4.4 TS OTA trigger

A TS OTA trigger is available for DVB project.

OTA trigger flow will only be detected when the system starts. It is implemented by the OTA monitor service configured in init.XXXX.rc. If a legal trigger TS is detected in the main frequency, the system will reboot into recovery.

- Main frequency

The initial value of main frequency is defined in:

```
device/gowarrior/tigerboard/ota_monitor/main_freq.c
```

When a platform boots up, you can find it in:

```
/data/system/mainfreq.conf
```

Such as:

main_freq=379

main_sym=6875

main_qam=6

- OTA trigger TS

Please contact your manufacturer to make an OTA trigger flow.

- How to enable OTA trigger detection function

Add NIM, DMX, and TSI in the Kernel configuration;

Include the `ota_monitor` source code, and the source code path is:

`device/gowarrior/tigerboard/ota_monitor`

Start OTA monitor service:

`device/gowarrior/tigerboard/init.tigerboard.rc`

```
service OTA monitor /system/bin/OTA monitor
    class main
    user root
    group root
    oneshot
```

1.5 Recovery Boot Flow

1.5.1 U-Boot Loads Recovery

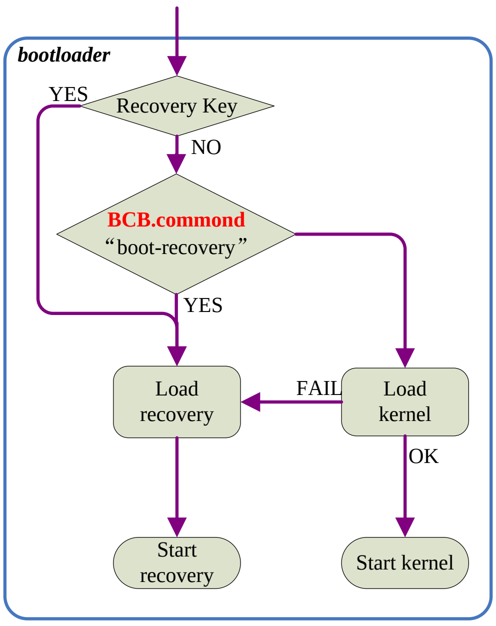


Figure 3. Recovery Loaded by U-Boot

U-Boot will load the recovery under the following conditions:

1. IR or GPIO key press detected
2. Recovery marked "boot-recovery" detected
3. Load Kernel failed

U-Boot will load recovery partition first. If recovery partition is corrupted and recoverybak partition is available, U-Boot will try to load recoverybak partition. U-Boot notes which partition (`recovery/recoverybak`) is loaded in cmdline.

1.5.2 Init.rc Starts the Recovery System

```
bootable/recovery/etc/init.rc
```

```
service recovery /sbin/recovery

    class main

    oneshot
```

1.6 Recovery Upgrade Process

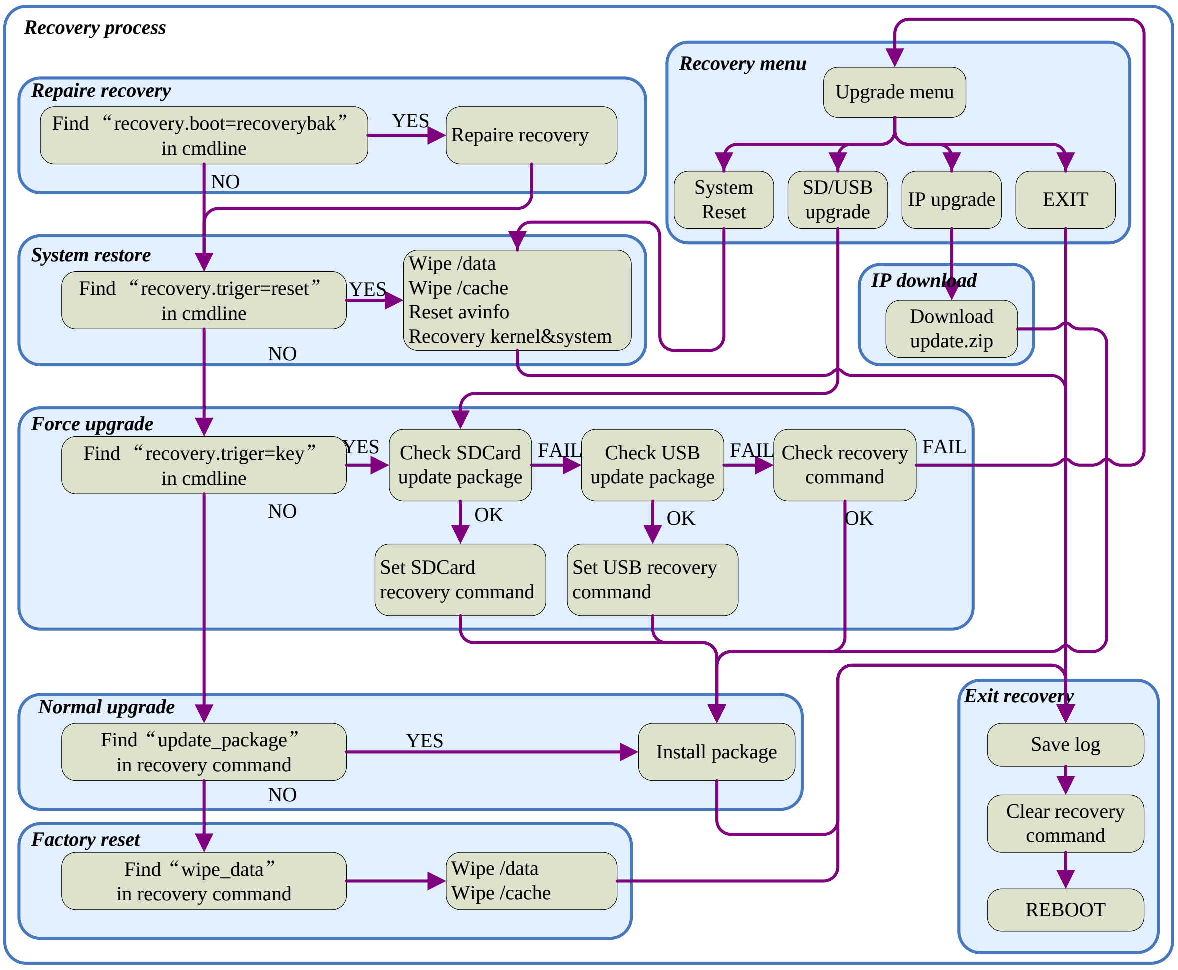


Figure 4. Recovery Upgrade Process

1.6.1 Repair Recovery

If recovery partition needs to be upgraded, recovery partition and recoverybak partition must be contained in partition table.

Once recovery partition is damaged, recoverybak partition will be the replacement. And a prompt "recovery.boot=recoverybak" will be added to cmdline. It will repair recovery partition by restoring the recoverybak partition to recovery partition.

If recoverybak partition is damaged, there are no any operations.

1.6.2 System Restore

If system restore mode is triggered, *cmdline* will prompt "`recovery.trigger=reset`".

In this mode, the partition backed up in the backup partition will be restored to corresponding partition. And default AVInfo will be reset. This function can be useful if TV can't output video with unsuitable TV mode.

1.6.3 Force Upgrade

If force upgrade mode is triggered, *cmdline* will prompt "`recovery.trigger=key`".

In this mode, recovery will first check whether a legal upgrade package exists in the USB/SD card. If not, it will then try to find an upgrade package in cache partition. Once the upgrade package is found, upgrade package will be installed. Otherwise, the system will jump to recovery menu where users can select the next step.

Note

Upgrade package must be located in USB/SD root directory, and must be named as /update.zip.

1.6.4 Recovery Menu

Recovery menu contains the current recovery version, the reason to trigger recovery system (CODE: XXX) and supported function options which can be configured by *cmdline*:

```
device/gowarrior/tigerboard/image/fs_ubi/Ali_nand_desc_
XXX.xml
```

Configure *recovery.param* in <recovery_cmdline>.

```
<!-- more cmdline define here -->
<cmdline>init=/init androidboot.console=ttyS0</cmdline>
<!--recovery.param(1:enable, 0:disable) -->
<!--bit0: reset system -->
<!--bit1: USB/SD upgrade -->
<!--bit2: OTA upgrade -->
<!--bit3: NET upgrade -->
<recovery_cmdline>init=/init androidboot.console=ttyS0 recovery.param=7</recovery_cmdline>
```

The detailed configurations are defined as follows:

Recovery Parameter	Menu Item	1	0
Bit0	System restore	display	hide
Bit1	USB/SD upgrade	display	hide
Bit2	OTA upgrade	display	hide
Bit3	IP upgrade	display	hide

Table 4. Detailed Configurations

For example, if recovery.param value is 7, These menu items: “System Recovery”, “SD Card/USB Upgrade”, “OTA Upgrade” and “Exit Upgrade” are visible from the recovery menu. Some functions like OTA upgrade and network upgrade can also be shielded by menu configurations.

The meaning of CODE:

```
typedef enum {
    CODE_TRIGGER_START = 0,
    CODE_TRIGGER_APP =1,    //trigger by Android App
    CODE_TRIGGER_RESET=2,   //trigger by Reset GPIO
    CODE_TRIGGER_FORCE_UPDRADE=3,   //trigger by force upgrade
    CODE_TRIGGER_POWER_OFF=4,   //trigger by power off when upgrade
    CODE_TRIGGER_CHECK_FAILED=5, //trigger by others
    CODE_TRIGGER_MAX,
} RecoveryCode;
```

1.6.5 Normal Upgrade

Normal upgrade is the native way in Android. It upgrades the system by the specified upgrade package.

1.6.6 Factory Reset

This function will wipe data partition and cache partition.

1.6.7 Exit Recovery

Both success and failure, the last step is to exit recovery:

1. Store log file
2. Clear recovery command

1.7 Installing an Upgrade Package

There are two important files in the upgrade package:

- Update-binary

During the installation of an upgrade package, recovery copies `update-binary` to RAM and then interpret and execute it.

The source file is located in `bootable/recovery/updater/`.

- Update-script

The format of this script is edify. It defines a specific upgrade process.

You only need to modify the `update-binary` and `update-script` file in the upgrade package to control the upgrade process, instead of modifying the recovery code. This mechanism, executing alongside interpreting, makes upgrade process easily to customize.

1.8 Power-Off Protection of the Recovery

The recovery process is finished only after the recovery command is cleared. If power-off happens before this, it indicates that the recovery process is not completed yet, and the system will continue the recovery process when powered on.

1.9 Log of Recovery

In recovery mode, current log information will be written into a temporary file in `/tmp/recovery.log`. When exiting recovery, the log will be saved in cache partition. The last five log files will be saved in the cache partition:

```
/cache/recovery/last_log.x
```

Note

In the last_log.x, the smaller the x is, the newer the log is.

2 Programming Guide

2.1 Using the Native Recovery System to Display Chinese Characters

Skip this section and refer to section 2.2 if you wish to design an UI interface that is different from an Android one.

2.1.1 Implementation Method

The native Android Recovery doesn't support Chinese character display. To minimize the changes, we use the native UI interface and make Chinese fonts (whose reference codes are located in `bootable/recovery/minui/font/`) based on the implementation of this interface. The steps are as follows:

1. Get the original bmp file of the words needed, and make sure their height is the same.

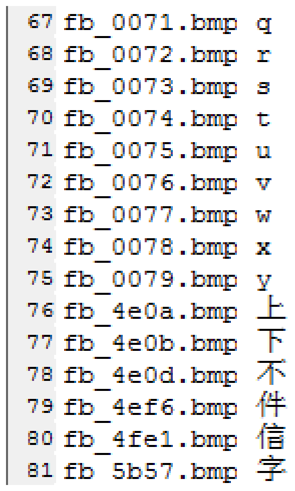
Note

We assume that you have had the software to make a bmp font file or you can use font2bmp.exe in bootable/recovery/minui/font/rsc/tools running in Windows.

2. Unify the width and height of the files to 32 pixels respectively by running the *bmpresize* command like this:

```
/bmpresize --width=32 --height=32 font.txt
```

3. Bmp files and its characters are stored in font.txt as follows:



```
fb_0071.bmp q
fb_0072.bmp r
fb_0073.bmp s
fb_0074.bmp t
fb_0075.bmp u
fb_0076.bmp v
fb_0077.bmp w
fb_0078.bmp x
fb_0079.bmp y
fb_4e0a.bmp 上
fb_4e0b.bmp 下
fb_4e0d.bmp 不
fb_4ef6.bmp 件
fb_4fe1.bmp 信
fb_5b57.bmp 字
```

Figure 5. Bmp Files and Its Characters in font.txt

Note

The source code of bmpresize is `bmpresize_recovery.c`, and you can generate this tool by running the command:

gcc –o bmpresize bmpresize_recovery.c

4. Compile all the fonts to generate `font_lib.c`, a temporary file used to generate the final `font_XXX.h` file, by using the bmp2font command like this:

```
./bmp2font font.txt >font_lib.c
```

Note

The source code of bmp2font is bmp2font_recovery.c, and you can generate this tool by running the command:

gcc –o bmp2font bmp2font_recovery.c

5. Add "`#include "font_lib.c"`" in mkfont.c, compile mkfont in host computer by running the command: gcc `-o mkfont mkfont.c` and the following command: `./mkfont >font_`XXX`.h`. The generated `font_`XXX`.h` file will be used by recovery as fonts

Note

font_XXX.h is required when compiling recovery code.

2.1.2 Reference Functions

Function	Description	Elements
`void gr_flip(void)`	Update the memory data to display device	Params: null Return: void

`void gr_color(unsigned char r, unsigned char g, unsigned char b, unsigned char a)`	Set the font color	Params: [in]
		r: red intensity;
		g: green intensity;
		b: blue intensity;
		a: Alpha transparency value.
		Return: void

`int gr_measure(const char *s)`	Display the total width of the text (unit:pixel)	Params: [in] s: the text string Return: int the width value (unit:pixel)
`void gr_font_size(int *x, int *y)`	Get font width and height (unit:pixel)	Params: [out] x: the font width y: the font height Return: void

Function	Description	Elements
`int gr_text(int x, int y, const char *s, int bold)`	Fill the memory with the displayed text	Params: [in] x: the address of the width direction (the left-most address is 0) y: the address of height direction (the highest address is 0) s: the displayed text Bold: bold font flag Return: int the address of width direction
`void gr_fill(int x1, int y1, int x2, int y2)`	Rectangle filling	Params: [in] x1,y1: Coordinate of Rectangular vertex 1 x2,y2: Coordinate of Rectangular vertex 2 Return: void
`void gr_blit(gr_surface source, int sx, int sy, int w, int h, int dx, int dy)`	Rectangle picture combination	Params: [in] source: source picture sx,sy,w,h: rectangle of source picture dx,dy: direct location Return: void
`unsigned int gr_get_width(gr_surface surface)`	Get the width of the picture(unit:pixel)	Params: [in] Surface: picture information Return: unsigned int The width of picture(unit:pixel)

Function	Description	Elements
`unsigned int gr_get_height (gr_surface surface)`	Get the height of picture(unit:pixel)	Params: [in] Surface: picture information Return: unsigned int The height of picture(unit:pixel)
`int gr_init(void)`	The initialization of graphics UI function	Params: null Return: int 0: succeed -1: failed
`void gr_exit(void)`	The exit to initialization of graphics UI function	Params: null Return: void
`int gr_fb_width(void)`	Get the displaying width of OSD	Params: null Return: int OSD width (unit: Pixel)
`int gr_fb_height(void)`	Get the displaying height of OSD	Params: null Return: int OSD height (unit: Pixel)

Table 5. Relevant Functions of Ext-Recovery

2.1.3 Reference Codes

1. Initialize before calling the Graphics UI function
2. Set up the color of the display area
3. Fill area or show the text
4. Update date to the device

Reference code as follows:

```
gr_init();
```

```
…

gr_color(0, 0, 0, 255);

gr_fill(dx, dy, dx+14*char_width, dy+char_height);

gr_color(255, 255, 255, 255);

gr_text(dx, dy, s,0);

…

gr_flip();
```

2.2 Multi-Language Support

According to current language environment, recovery can switch to corresponding language for display. English and Chinese are now available.

If you want to modify or add a string, please edit the flowing files:

String index list: `bootable/recovery/string/string.id`

Chinese string list: `bootable/recovery/string/res/str_chn.c`

English string list: `bootable/recovery/string/res/str_eng.c`

It must be pointed out that the string definition in these three files must have one-to-one correspondence. Please keep in mind that these three files must be modified simultaneously when adding or deleting a string.

```
char *get_string(int id)
```

According to current language environment, get the string specified index.

Params:

[in]id: string index, refer to bootable/recovery/string/string.id

Return: string

2.3 UI Customization

The ALIScreenRecoveryUI class, needed to be defined by user, inherits the RecoveryUI. The main functions are as follows:

Function	Description	Elements

Function	Description	Elements
`void Init()`	Initialization function	Params: null Return: void
`void SetLocale(const char* locale)`	Set the display language	Params: [in] locale: language type Return: void
`void SetBackground(Icon icon)`	Set the background picture	Params: [in] icon: background picture Return: void
`void SetProgressType(ProgressType type)`	Set the progress bar type	Params: [in] type: progress bar type Return: void
`void ShowProgress(float portion, float seconds)`	Show the progress bar	Params: [in] portion: progress bar portion seconds: progress bar duration Return: void
`void SetProgress(float fraction)`	Set progress bar portion	Params: [in] fraction: progress bar portion Return: void
`void ShowText(bool visible)`	Show text	Params: [in] visible: 1--visible; 0 -- invisible Return: void
`bool IsTextVisible()`	Judge if the text is visible currently	Params: null Return: bool 1: visible0: invisible

Function	Description	Elements
`bool WasTextEverVisible ()`	Judge if the text is visible currently	Params: null Return: bool 1: visible 0: invisible
`void Print(const char* fmt, ...)`	Log information	Params: [in] fmt,...: information to be displayed Return: void
`int WaitKey()`	Wait for the response key value	Params: null Return: int the response key value
`bool IsKeyPressed(int key)`	Judge if a key is pressed	Params: [in] key: key value Return: bool 1: pressed 0: released
`void FlushKeys()`	Clear all the key values	Params: null Return: void
`KeyAction CheckKey(int key)`	Special key value processing	Params: [in] key: key value Return: KeyAction Special key type
`void NextCheckKeyIsLong(bool is_long_press)`	Set next detection of special key value to long press key	Params: [in] is_long_press: 1—set to long_press key; 0—cancel long_press key; Return: void

Function	Description	Elements
`void KeyLongPress(int key)`	Long press the key	Params: [in] key: the value of the long pressed key Return: void
`void StartMenu(const char* const * headers, const char* const * items, int initial_selection)`	Start the menu	Params: [in] headers: the menu header items: the menu items initial_selection: the initial options Return: void
`int SelectMenu(int sel)`	Select some option in the menu	Params: [in] sel: the selection index Return: int the selection index
`void EndMenu()`	End the menu	Params: null Return: void
`int ProcessKey(int keycode, int param)`	Handle some pressed key	Params: [in]keycode: the value of the pressed key param: extension parameter Return: int 0: succeed 1: failed
`void ChangeMenu(int frame, int param)`	Change the menu	Params: [in] frame: the menu to be changed param: extension parameter Return: void

Function	Description	Elements
`void GetFrameIdx(int *frame, int *items)`	Get the current menu information	Params: [out] frame: the current menu index; item: the current menu selection initial index Return: void
`int GetOTAParam(void *param)`	Get OTA related params	Params: [out] param: output param. Return: int 0: succeed 1: failed
`bool DownloadPackage()`	Start download from network	Params: null Return: bool true: download succeed false: download failed

Table 6. Relevant Functions of UI Customization

2.4 Recovery Trigger

```
import android.os.RecoverySystem;


// package has been downloaded to /cache/update.zip by IP
RecoverySystem.installPackage(getApplicationContext(),           new
File("/cache/update.zip"));


// trigger recovery by usb mode
RecoverySystem.installPackage(getApplicationContext(),           new
File("/usb/recovery/update.zip"));
```

```
// factory reset

RecoverySystem.rebootWipeUserData(mContext)
```

2.5 Self-Defined Upgrade Process

The upgrade process is update-binary in the upgrade package, and the path is:

`bootable/recovery/updater/`

File	Instruction
`install.c`	Functions required by the updater-script, like mount, format, and show_progress
`install_extension.c`	Updater-script extended functions, like version_check, bootargs_check, recovery_update
`updater.c`	Upgrade main function, executing the updater-script
`version.c`	Version control

Table 7. Self-Defined Upgrade Process

2.5.1 How to Add Self-Defined Upgrade Process

Please add the extended upgrade process in `install_extension.c`.

Function type: typedef char `*(*CTRL_FUNCTION)(int argc, char *argv[]);`

Add this function into `func_ctrl[]`. From example:

```
struct FUNC_CTRL {

    char name[64];

    CTRL_FUNCTION func;

} func_ctrl[] = {

    {"version_check", VersionCheck},

    {"md5_save", Md5Save},
    {"bootargs_check", BootargsCheck},

    {"bootloader_update", BootloaderUpdate},

    {"recovery_update", RecoveryUpdate},

};
```

And the calling mode in updater-script is like this:

```
func_ctrl("version_check",1,"ALi1.3.4");

func_ctrl("bootargs_check",1,package_extract_file("bootargs.img")
);

func_ctrl("recovery_update",2,package_extract_file("recovery.img"
),"ALiRecovery1.2.3");
```

2.5.2 Version Check

Whether a target platform can use an upgrade package to upgrade is controlled by the platform version and software version of the upgrade package.

- Upgrade script

```
func_ctrl("version_check",1,"ALi1.3.4");
```

The first line of updater-script always calls the "`version_check`" function to check the software version. And the param "ALi1.3.4" is the software version of the current upgrade package.

- Upgrader

```
static char *VersionCheck(int argc, char *argv[]) ;
```

"`version_check`" corresponds to the function `VersionCheck` and implemented in `install_extension.c.` When the package version meets upgrade requirements, execution of the upgrade-script can continue. Otherwise, the upgrade will be interrupted.

2.5.3 Partition Table Check

Partition table comparison begins after software comparison. An upgrade package contains partition table information (bootargs.img). When the partition table is consistent with that in the target platform, executing the upgrade-script will continue. Otherwise, the partition table needs to be updated before auto reboot to update other contents.

- Upgrade script

```
func_ctrl("bootargs_check",1,package_extract_file("bootargs.img"));
```

This function calls " `bootargs_check` " to check bootargs partition. It takes one parameter: bootargs.img, bootargs partition image in the upgrade package.

- Upgrader

```
static char *BootargsCheck(int argc, void *argv[])
```

`" bootargs_check " is the BootargsCheck function defined in install_extension.c.`

2.5.4 Recovery Partition Upgrade

Whether upgrade recovery partition is controlled by its own version.

- Upgrade script

```
func_ctrl("recovery_update",2,package_extract_file("recovery.img"),"A
LiRecovery1.2.4");
```

This function calls "`recovery_update`" to upgrade the recovery partition. It takes two parameters; one is "`recovery.img`", a recovery partition image in upgrade package; the other is "ALiRecovery1.2.4", the recovery version in upgrade package.

- Upgrader

```
static char *RecoveryUpdate(int argc, void *argv[])
```

"`recovery_update`" is the RecoveryUpdate function defined in install_extension.c. Recovery partition will be upgraded if the recovery and recoverybak partition both exist and the recovery version in the upgrade package meets the upgrade condition. The recoverybak partition will be upgraded before upgrading recovery. The version comparison rule is also customizable.

2.5.5 Bootloader Upgrade

Whether upgrade bootloader is controlled by bootloader version.

- Upgrade script

```
func_ctrl("bootloader_update",2,package_extract_file("bootloader.img"
),"ALiBoot1.1.1");
```

This function calls `"bootloader_update"` to upgrade the Bootloade partition. It takes two parameters. One is `"bootloader.img"`, bootloader partition image in the upgrade package, and the other is " ALiBoot1.1.1", the bootloader version in the upgrade package.

- Upgrader

```
static char *BootloaderUpdate(int argc, void *argv[])
```

"`bootloader_update`" is the BootloaderUpdate function point defined in `install_extension.c`. The Bootloader will be upgraded if the bootloader and bootloaderbak partition both exist and the bootloader version in the upgrade package meets the upgrade condition. The version comparison rule is also customizable.

2.5.6 Saving MD5

The MD5 value of the upgrade package should be saved after upgrade.

- Upgrade script

```
func_ctrl("md5_save",0);
```

It is always called at the end of the upgrade script. This function saves the MD5 value of the upgrade package and takes no parameters.

- Upgrader

```
static char *Md5Save(int argc, char *argv[])
```

"`md5_save`" is the Md5Save function defined in `install_extension.c`.

2.6 Customized Upgrade Script

The `updater-script` format is edify. Relevant codes to generate this script are as follows:

```
build/tools/releasetools/ota_from_target_filesXXX

build/tools/releasetools/commonXXX.py

build/tools/releasetools/edify_generatorXXX.py
```

2.7 Upgrade Package Signature

An expected key needs to be made for a formal project. It needs to be put in this path first: build/target/product/security. And it needs to be specified during installation of OTA tool. Otherwise, `build/target/product/security/testkey` will be used by default.

Private Key will be used to sign and encrypt an upgrade package when generating an upgrade package. During upgrade recovery will use public key, which is stored recovery instead of the upgrade package, to verify the upgrade package.

3 API Description

3.1 Package android.os.RecoverySystem

```
public static void installPackage(Context context, File packageFile)

* Reboots the device in order to install the given update package.

* Requires the {@link android.Manifest.permission#REBOOT} permission.

* @param context     the Context to use

* @param packageFile  the update package to install.  Must be on a
partition mountable by recovery.  (The set of partitions known to recovery
may vary from device to device.  Generally, /cache and /data are safe.)

* @throws IOException  if writing the recovery command file fails, or if
the reboot itself fails.


public static void rebootWipeCache(Context context)

* Reboot into the recovery system to wipe the /cache partition.

* @throws IOException if something goes wrong.


public static void rebootWipeUserData(android.content.Context)

* Reboots the device and wipes the user data partition.  This is sometimes
called a "factory reset", which is something of a misnomer because the
system partition is not restored to its factory state.

* Requires the {@link android.Manifest.permission#REBOOT} permission.

* @param context  the Context to use

* @throws IOException  if writing the recovery command file fails, or if
the reboot itself fails.


public static void verifyPackage(File packageFile, ProgressListener
listener, File deviceCertsZipFile)

* Verify the cryptographic signature of a system update package before installing it.  Note that the package is also verified separately by the installer once the device is rebooted into the recovery system.  This function will return only if the package was successfully verified; otherwise it will throw an exception.

* Verification of a package can take significant time, so this function should not be called from a UI thread.  Interrupting the thread while this function is in progress will result in a SecurityException being thrown (and the thread's interrupt flag will be cleared).

* @param packageFile  the package to be verified

* @param listener     an object to receive periodic progress updates as verification proceeds.  May be null.

* @param deviceCertsZipFile  the zip file of certificates whose public keys we will accept.  Verification succeeds if the package is signed by the private key corresponding to any public key in this file.  May be null to use the system default file (currently "/system/etc/security/otacerts.zip").

* @throws IOException if there were any errors reading the package or certs files.

* @throws GeneralSecurityException if verification failed
```

3.2 Package android.alisdk.DeviceInfo

Relevant codes

```
frameworks/base/alisdk/java/android/alisdk/DeviceInfo.java

frameworks/base/alisdk/jni/com_ali_DeviceInfo.cpp
```

- public native String nativeGetDeviceSN();

Get the device sequence number

- public native int nativeGetDeviceOUI();

Get the device OUI

- public native String nativeGetDeviceMAC();

Get the device MAC

- public native String nativeGetDeviceHwVersion();

Get the device hardware version

- public native String nativeGetDeviceSwVersion();

Get the device software version

- public native int nativeGetDeviceStatus();

Get the device status

Status definition:

Bit0: factory test enable

Bit1-31: reserved

- public native void nativeSetDeviceStatus(int status);

Set device status

- public native byte[] nativeGetDeviceSwMD5(int len);

Get md5 value. The length is specified by *len* which should not be greater than 128

- public native int nativeSetDeviceSwMD5(byte[] md5, int len);

Set md5 value. The length is specified by *len* which should not be greater than 128.

- public native byte[] nativeGetDeviceSwPrivate(int len);

Get sw private. The length is specified by *len* which should not be greater than 1024*6

- public native int nativeSetDeviceSwPrivate(byte[] pri, int len);

Set sw private. The length is specified by len which should not be greater than 1024*6

3.3 libalideviceinfo for C API

```
frameworks/base/alisdk/libdeviceinfo/
```

- int get_swversion(char *sver);

@sver	[out],	software version

@Return the software version string length; <0 fail

- int get_swmd5(unsigned char *md5);

@md5 [out], MD5 value, md5 buffer length should not be less than 128

@Return MD5 value string length, current is 128bytes; <0 fail

- int save_swmd5(unsigned char *md5, int len);

@md5 [in]

@len [in], md5 length to be saved, should not be less than 128

@ Return <0 fail; otherwise ok

- int get_swprivate(unsigned char *pri);

@pri [out], private software information, pri buffer length should not be less than 1024*6

@Return private software information length, current is 1024*6 bytes; <0 fail

- int save_swprivate(unsigned char *pri, int len);

@pri [in]

@len [in], pri length to be saved, should not be less than 1024*6

@ Return <0 fail; otherwise ok

- int get_mac(unsigned char *mac);

@ mac [out]

@Returnmac length; <0 fail

- int get_devicestatus(unsigned int *status);

@ status [out]

@Return <0 fail; otherwise ok

Status definition:

Bit0: factory test enable

Bit1-31: reserved

- int save_devicestatus(unsigned int status);

@ status [in]

@Return <0 fail; otherwise ok

3.4 Reference Codes

- USB upgrade trigger calling reference:

```
RecoverySystem.installPackage(getApplicationContext(),         new
File("/usb/recovery/update.zip"));
```

- OTA upgrade trigger calling reference:

```
RecoverySystem.installPackage(getApplicationContext(),         new
File("/ota/recovery/update.zip"));
```

- SD upgrade trigger calling reference:

```
RecoverySystem.installPackage(getApplicationContext(),         new
File("/sdcard/update.zip"));
```

- Normal upgrade trigger calling reference: (the default upgrade package path)

```
RecoverySystem.installPackage(getApplicationContext(),         new
File("/cache/update.zip"));
```

Note

After calling the interfaces above, JAVA will write "--update_package=XXX" into /cache/recovery/command. This option will enable recovery to determine the specific upgrade process after reboot. Refer to Chapter 1.6 Recovery Upgrade Process.

Appendix: Glossary

Abbr.	Full Name
OTA	Over-The-Air
TS	Transport Stream

Table 8. List of Abbreviations

Revision History

Document Change History

Revision	Changes	Date
v1.0	Initial Release	September 07, 2015

Table 9. Document Change History

Software Changes

Revision	Changes	Date
v1.0	Initial Release	September 07, 2015

Table 10. Software Change History

www.gowarriorosh.com

Headquarters

Tel: +886-2-8752-2000

Fax: +886-2-8751-1001