GOWARRIOR

GoDroid

Application Notes

How to Build Customized Projects

v1.1

Copyright Statement

For the latest version of this document refer to:

www.gowarriorosh.com

Table of Contents

List of Tables

List of Figures

Preface

Overview

This manual mainly describes the build system of GoDroid v1.1, and how to use GoDroid build system to develop customized projects, libraries and applications. This manual is organized into the following chapters:

- **Chapter 1: Build Architecture**

This chapter provides information on the build architecture and method of GoDroid building.

- **Chapter 2: Custom Build**

This chapter gives compact description on customized build commands in GoDroid build system.

- **Chapter 3: GoDroid Secondary Development**

This chapter discusses how to add devices and applications to GoDroid system.

- **Chapter 4: FAQ**

This chapter describes range of common questions and their solutions which can be arise during the process of building GoDroid customized projects.

Audience

This manual is applicable for the users who wish to learn how to build customized projects 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

- http://developer.Android.com
- http://developer.android.com/tools/sdk/ndk/index.html

- https://source.android.com/source/building-running.html

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 Build Architecture

This chapter mainly introduces GoDroid build architecture, and gives detailed description for each part.

1.1 Introduction

Figure 1 shows GoDroid build architecture. The *build/core* directory includes most files of the build system, where *main.mk* is the entry of the whole build system.

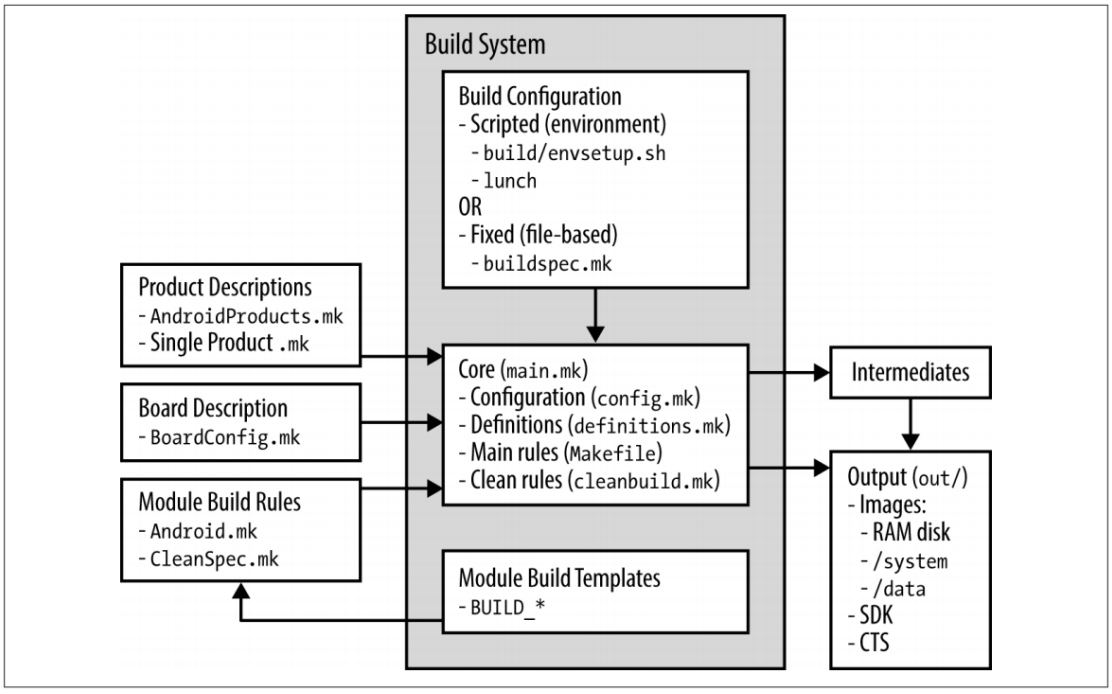

Figure 1. GoDroid Build Architecture Diagram

GoDroid build system integrates all contents into an individual `Makefile`. This is different from Linux Kernel that uses the recursive `Makefile` mechanism.

1.2 GoDroid Building Procedure

It is very simple to build Android source code. It needs only 3 steps:

```
$ source build/envsetup.sh

$ lunch aosp_tigerboard-eng

$ make -j8
```

1.3 envsetup.sh

`build/envsetup.sh` is used for initializing build environment variables. `envsetup.sh` mainly completes the following three tasks:

- Define functions including *m*, *mm*, *mmm*, *lunch*, etc.;
- Check current shell environment;
- Add build targets, search `vendor/*/vendorsetup.sh`, `vendor/*/*/vendorsetup.sh`, `device/*/*/vendorsetup.sh`, and then execute the `source` command to add variables of the files to the system environment.

Look at the running result:

```
$ source build/envsetup.sh

including device/generic/armv7-a-neon/vendorsetup.sh

including device/gowarrior/tigerboard/vendorsetup.sh

including sdk/bash_completion/adb.bash
```

Search `vendorsetup.sh` file in the specified directory. Users can see all the commands in `envsetup.sh` using the `hmm` in Android 4.4 or later.

$ hmm

Call "`. build/envsetup.sh`" from shell to add the following functions into the environment:

Command	Details
- lunch:	lunch <product_name>-<build_variant>
- tapas:	tapas [<App1> <App2> ...] [arm|x86|mips|armv5] [eng|userdebug|user]
- croot:	Changes directory to the top of the tree.
- m:	Makes from the top of the tree.
- mm:	Builds all of the modules in the current directory, but not their dependencies.
- mmm:	Builds all of the modules in the supplied directories, but not their dependencies.
- mma:	Builds all of the modules in the current directory, and their dependencies.
- mmma:	Builds all of the modules in the supplied directories, and their dependencies.
- cgrep:	Greps on all local C/C++ files.
- jgrep:	Greps on all local Java files.
- resgrep:	Greps on all local res/*.xml files.
- godir:	Go to the directory containing a file.

Table 3. Functions Added to the Environment

The above mentioned commands can directly run in `shell`.

1.4 Lunch

The next command is `lunch`, which is used to select a target device to build.

```
$ lunch

You're building on Linux


Lunch menu... pick a combo:

    1. aosp_arm-eng

    2. aosp_x86-eng

    3. aosp_mips-eng

    4. vbox_x86-eng

    5. mini_armv7a_neon-userdebug

    6. aosp_tigerboard-eng

    7. aosp_tigerboard-userdebug

    8. aosp_tigerboard-user

Which would you like? [aosp_arm-eng]
```

If you want to select `aosp_tigerboard-eng`, you can input `aosp_tigerboard-eng` or its serial number 6. If you want to directly select without displaying the list then you can directly input the command:

```
$ lunch aosp_tigerboard-eng
```

1.5 Make

You can run `make` to build. The building process will be a long process. `make -j8` builds 8 tasks simultaneously, which can greatly reduce the time.

`make -j` parameter indicates how many building tasks can run simultaneously. The task number is determined by CPU core number. Using `grep -c processor /proc/cpuinfo` can detect the number of tasks processors.

1.6 Output

When building is completed, the output directory is `out/`, which generally has two subdirectories `host/` and `target/`. The specified device output is in the directory `out/target/product/PRODUCT_DEVICE/`. In GoDroid

project, the output is in the directory `out/target/product/tigerboard`.

File/Directory	Content
`data/`	Data directory, for generating userdata.img
`root/`	rootfs directory, for generating ramdisk.img
`system/`	system directory, , for generating system.img
`system.img`	System image, based on system/directory, for burning.
`ramdisk.img`	Ramdisk image, based on root/ directory, for burning.
`userdata.img`	Userdata image, based on data/ directory, for burning.
`symbols/`	Debuggable version of library in root/ and system/.

Table 4. Device Output Directory Content

2 Custom Build

This chapter provides details on the build commands commonly used in Android build system, such as building own SDK in Linux and Windows, CTS building, NDK building, API update, single module building, building recursion, etc.

2.1 Default Target Build

Droid defines the default target in `main.mk`. The following commands are used to define the default target.

```
make droid
```

```
make
```

For general customers, just execute `make` to build.

In case if you only use `make` to build, only brief build information can be seen in the building process. If you need detailed build information, such as checking build parameter, then execute the following command:

```
make showcommands
```

2.2 SDK Build in Linux

For Android official SDK, please go to Google official website: *http://developer.android.com*. Users can release their own SDK according to their devices, especially when the Android core API is modified. The commands for building SDK are:

```
$ . build/envsetup.sh
$ lunch sdk-eng
$ make sdk
```

After completing building, SDK will be generated in `out/host/linux-x86/sdk/`.

2.3 SDK Build in Windows

Building SDK in Windows is similar to that in Linux. Execute the following command:

```
$ . build/envsetup.sh

$ lunch sdk-eng

$ make win_sdk
```

After completing building, SDK will be generated in `out/host/windows/sdk/`.

2.4 CTS Build

`envsetup.sh` and `lunch` are not required for CTS building. To build CTS, just execute the following commands:

```
$ make cts

...

Generating test description for package Android.sax

Generating test description for package Android.performance

Generating test description for package Android.graphics

Generating test description for package Android.database

Generating test description for package Android.text

Generating test description for package Android.webkit

Generating test description for package Android.gesture

Generating test plan CTS

Generating test plan Android
```

2.5 NDK Build

NDK is an acronym for Native Development Kit. NDK is a toolset that allows developers to implement parts of your app using native-code languages such as C and C++.

For detailed documents of Android NDK, please go to *http://developer.android.com/tools/sdk/ndk/index.html*.

The Android NDK r9d is suggested for the GoWarrior projects. Users could download it from the following URLs:

http://dl.google.com/android/ndk/android-ndk-r9d-windows-x86.zip

http://dl.google.com/android/ndk/android-ndk-r9d-windows-x86_64.zip

http://dl.google.com/android/ndk/android-ndk-r9d-darwin-x86.tar.bz2

http://dl.google.com/android/ndk/android-ndk-r9d-linux-x86.tar.bz2

http://dl.google.com/android/ndk/android-ndk-r9d-linux-x86_64.tar.bz2

2.6 API Update

Generally speaking, it's not recommended to modify the core API of AOSP system. If modified, system will display a warning message when building and the build will fail.

```
******************************

You have tried to change the API from what has been previously approved.

To make these errors go away, you have two choices:

1) You can add "@hide" javadoc comments to the methods, etc. listed
in the

errors above.

2) You can update current.xml by executing the following command:

make update-api
```

It is recommended use `make update-api` to fix the error in the above warning information. In this case, just execute `make update-api`, and then execute `make`. This error will not occur any more.

2.7 Single Module Build

Building the whole AOSP project may take a long time. However, sometimes we only need build one module:

```
make Launcher2
```

Or we can clear a module that has been built individually by executing the following command:

```
make clean-Launcher2
```

2.8 Cross-Compilation Out-of-Tree

Cross-Compiling Out-of-Tree means that some modules are separated from the AOSP build system and do not participate in AOSP project building. As cross-compiling out-of-tree is based on `Makefile` instead of `Android.mk`, it is essentially the same with cross-compiling for Linux. Therefore, these modules will not be compiled when the make command of AOSP is executed separately.

But for Android system, if an application wants to run in Android, it must depend on Android toolchain and bionic library. Just modify `makefile` of this module, and add the environment variables required by building to it:

```
# Paths and settings

TARGET_PRODUCT = generic

ALiDroid_ROOT = /home/karim/ALiDroid/aosp-4.3

BIONIC_LIBC = $(ALIDROID_ROOT)/bionic/libc

PRODUCT_OUT = $(ALIDROID_ROOT)/out/target/product/$(TARGET_PRODUCT)

CROSS_COMPILE = \

$(ALIDROID_ROOT)/
prebuilts/gcc/linux-x86/arm/arm-linux-ALiDroideabi-4.7/bin/
arm-linux-ALiDroideabi-

# Tool names

AS = $(CROSS_COMPILE)as

AR = $(CROSS_COMPILE)ar

CC = $(CROSS_COMPILE)gcc

CPP = $(CC) -E

LD = $(CROSS_COMPILE)ld

NM = $(CROSS_COMPILE)nm

OBJCOPY = $(CROSS_COMPILE)objcopy

OBJDUMP = $(CROSS_COMPILE)objdump
```

```
RANLIB = $(CROSS_COMPILE)ranlib

READELF = $(CROSS_COMPILE)readelf

SIZE = $(CROSS_COMPILE)size

STRINGS = $(CROSS_COMPILE)strings

STRIP = $(CROSS_COMPILE)strip

export AS AR CC CPP LD NM OBJCOPY OBJDUMP \

RANLIB READELF SIZE STRINGS STRIP

CFLAGS = -O2 -Wall -fno-short-enums

HEADER_OPS = -I$(BIONIC_LIBC)/arch-arm/include \

-I$(BIONIC_LIBC)/kernel/common \

-I$(BIONIC_LIBC)/kernel/arch-arm

LDFLAGS = -nostdlib -Wl,-dynamic-linker,/system/bin/linker \

$(PRODUCT_OUT)/obj/lib/crtbegin_dynamic.o \

$(PRODUCT_OUT)/obj/lib/crtend_ALiDroid.o \

-L$(PRODUCT_OUT)/obj/lib -lc -ldl

# Installation variables

EXEC_NAME = example-app

INSTALL = install

INSTALL_DIR = $(PRODUCT_OUT)/system/bin

# Files needed for the build

OBJS = example-app.o

# Make rules

all: example-app

.c.o:

$(CC) $(CFLAGS) $(HEADER_OPS) -c $<

example-app: ${OBJS}

$(CC) -o $(EXEC_NAME) ${OBJS} $(LDFLAGS)

install: example-app
```

```
test -d $(INSTALL_DIR) || $(INSTALL) -d -m 755 $(INSTALL_DIR)

$(INSTALL) -m 755 $(EXEC_NAME) $(INSTALL_DIR)

clean:

rm -f *.o $(EXEC_NAME) core

distclean:

rm -f *~

rm -f *.o $(EXEC_NAME) core
```

In this case, this module can also be built even if `build/envsetup.sh` and `lunch` are not executed. However, AOSP build system will not build it, so building an out-of-tree module needs to be manually executed each time.

2.9 Cross-Compilation in-the-Tree

The advantage of cross-compiling out-of-tree lies in that it can easily port a Linux project to Android without caring too much about Android intrinsic build system, which is the same as cross-compile in Linux. Android build system uses `Android.mk` as definition file of each module building, which cannot identify general `makefile` in Linux. If users want to add a module to AOSP build system, they must provide the `Android.mk` file of this module.

The cross-compiling in-the-tree method can not only use `Makefile` same as Linux cross-compiling, but also can add new modules to AOSP build system. Refer to the `external/x264` in `linaro` for the handling method.

```
include $(CLEAR_VARS)

X264_TCDIR := $(realpath $(shell dirname $(TARGET_TOOLS_PREFIX)))

X264_TCPREFIX := $(shell basename $(TARGET_TOOLS_PREFIX))

# FIXME remove -fno-strict-aliasing once the code is fixed

COMPILER_FLAGS    :=    $(subst    -I    ,-I../../,$(subst    -include
system/core/include/arch/linux-arm/AndroidConfig.h,,$(        subst
-include
build/core/combo/include/arch/linux-arm/AndroidConfig.h,,$(TARGET_
GLOBAL_CFLAGS)))) -fno-strict-aliasing
```

```
.phony: x264


droid: x264


systemtarball: x264


x264:        $(TARGET_CRTBEGIN_DYNAMIC_O)        $(TARGET_CRTEND_O)
$(TARGET_OUT_SHARED_LIBRARIES)/libm.so
$(TARGET_OUT_SHARED_LIBRARIES)/libc.so
$(TARGET_OUT_SHARED_LIBRARIES)/libdl.so

   cd $(TOP)/external/x264 && \

   export PATH=$(X264_TCDIR):$(PATH) && \

   ./configure \

       --host=arm-linux \

       --prefix=/system \

       --bindir=/system/bin \

       --libdir=/system/lib \

       --enable-shared \

       --disable-thread \

       --cross-prefix=$(X264_TCPREFIX) \

       --extra-ldflags="-nostdlib
-Wl,-dynamic-linker,/system/bin/linker
-L../../$(PRODUCT_OUT)/system/lib
-L../../$(TARGET_OUT_SHARED_LIBRARIES) -lgcc -ldl -lc" \

       --extra-cflags="$(COMPILER_FLAGS)
-I../../bionic/libc/include       -I../../bionic/libc/kernel/common
-I../../bionic/libc/kernel/arch-arm
-I../../bionic/libc/arch-arm/include -I../../bionic/libm/include" &&
\

   $(MAKE) && \

   $(MAKE) install DESTDIR=../../$(PRODUCT_OUT)/
```

3 GoDroid Secondary Development

This chapter discusses how to use Android open source codes for secondary development to meet users' demand, such as adding a device, application, native tools or daemon process.

3.1 Adding a Device

Take GoDroid Board as example:

- Create a new directory in the format of `device/company/my_device`. In the example of DemoBoard, we create the directory `device/ali/tigerboard` to save all contents related to device. The following operations are performed in this directory.
- Create a new file in this directory, named as `vendorsetup.sh`. Add the following content to it:

```
add_lunch_combo my_device-eng
```

This example is named as `aosp_tigerboard-eng`. Users can name it according to their device name. Please pay attention to the suffix of the symbol "-", it includes eng, user, etc. For details, please refer to the website *https://source.android.com/source/building-running.html*

- Create the `Android.mk` file, and add the following to it:

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

include $(CLEAR_VARS)

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

- Create the **`AndroidProducts.mk`** `file`, and add the following to it:

```
PRODUCT_MAKEFILES := \

       $(LOCAL_DIR)/my_device_name.mk \
```

In GoDroid project, it is named as `aosp_tigerboard.mk`. Users can change it to the name of their personal device.

- Create the `my_device_name.mk` file, and add the following to it:

```
$(call inherit-product, $(SRC_TARGET_DIR)/product/full_base.mk)

$(call inherit-product, device/company/my_device/device.mk)

PRODUCT_BRAND   := device_brand

PRODUCT_DEVICE  := device_name

PRODUCT_NAME    := device_name
```

Please be noted `device_name` needs to be modified accordingly.

- Create the `device.mk` file, with the following commands:

```
PRODUCT_COPY_FILES    +=
device/company/my_device/init.rc:root/init.rc

PRODUCT_TAGS      += dalvik.gc.type-precise

$(call inherit-product,
frameworks/base/build/tablet-dalvik-heap.mk)
```

`device.mk` is very important. The variables in it define lots of board related contents. The project settings are almost contained in this file.

For the specific variables in this file, please refer to file `build/core/product.mk`.

- Create the `BoardConfig.mk` file.

This file defines the parameters related with Board when building, such as IC type, supported instruction set and some Board definitions. The following is an example of TIGER Board:

```
TARGET_NO_BOOTLOADER := true
```

```
TARGET_NO_KERNEL := true

TARGET_USE_UBOOT := false

 TARGET_NO_RECOVERY := true

 TARGET_NO_RADIOIMAGE := true


 TARGET_CPU_ABI := armeabi-v7a

 TARGET_CPU_ABI2 := armeabi

 TARGET_CPU_SMP := true

 TARGET_ARCH := arm

 TARGET_ARCH_VARIANT := armv7-a-neon

 TARGET_CPU_VARIANT := cortex-a9


 TARGET_BOARD_PLATFORM := ali3921
```

- After adding these basic files, execute the following commands in AOSP directory:

```
. build/envsetup.sh

lunch
```

And the added device will be listed:

```
You're building on Linux


Lunch menu... pick a combo:

    1. aosp_arm-eng

    2. aosp_x86-eng

    3. aosp_mips-eng

    4. vbox_x86-eng
```

```
    5. mini_armv7a_neon-userdebug


    6. aosp_tigerboard-eng

    7. aosp_tigerboard-userdebug

    8. aosp_tigerboard-user


Which would you like? [aosp_arm-eng] 6
```

3.2 Adding an Application

Generally, Android application uses Eclipse+ADT ADT (Android Developer Tools) to develop. After completing application development, if users want to add applications of AOSP:

1. Copy a project into *packages/apps/*:

```
cp -raf my_project packages/apps/
```

2. Create the `Android.mk` file in the *packages/apps/my_project* directory.

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

include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := optional

LOCAL_SRC_FILES := $(call all-java-files-under, src)

LOCAL_PACKAGE_NAME := my_project

include $(BUILD_PACKAGE)
```

`include $(BUILD_PACKAGE)` is used for generating java executable program. In the given examples, there are many similar commands. Android will use different building rules to build apps, libraries, etc.

3.3 Modifying an Application

Sometimes, users do not want to add a new app, but to modify existing ones in Android to meet their requirement, such as modifying the default value of settings. In AOSP, overlays mechanism allows device manufacturers to modify the resource of app on the premise that they do not directly modify the original app resource s. To realize this function, we need to create an overlay tree first, and notice the build system. For overlay tree, the simplest location is the relevant path we have created:

```
cd device/ali/tigerboard

mkdir overlay
```

Then notice the build system to use this overlay, and add the following content to **device.mk**:

```
DEVICE_PACKAGE_OVERLAYS := device/ali/tigerboard/overlay
```

Now we can put the overlay of each app into this directory. For example, if we want to modify some character string of *Launcher2*, we need to take the following steps:

```
mkdir -p overlay/packages/apps/Launcher2/res/values

cp  aosp-root/packages/apps/Launcher2/res/values/strings.xml \

overlay/packages/apps/Launcher2/res/values/
```

Then modify:

```
overlay/packages/apps/Launcher2/res/values/strings.xml
```

Delete the value with no need to overlay and leave the value to be modified only. After building, the character resource of Launcher2 itself does not change, but the value of its character resources has been changed by overlaying.

3.4 Adding a Tool or Daemon Process

Adding a new tool or daemon process is similar with generating an app. We just need to create file `Android.mk`.

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

include $(CLEAR_VARS)

LOCAL_MODULE := hello-world

LOCAL_MODULE_TAGS := optional

LOCAL_SRC_FILES := hello-world.cpp

LOCAL_SHARED_LIBRARIES := liblog

include $(BUILD_EXECUTABLE)
```

`include $(BUILD_ EXECUTABLE)` is used for generating local executable programs written by C/C++.

3.5 Adding a Library

Adding a native library is similar with that of tools and app. The most important is also the file `Android.mk`.

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

include $(CLEAR_VARS)

LOCAL_MODULE := libmylib

LOCAL_MODULE_TAGS := optional

LOCAL_PRELINK_MODULE := false

LOCAL_SRC_FILES := mylib.c

include $(BUILD_SHARED_LIBRARY)
```

`include $(BUILD_SHARED_LIBRARY)` is used for generating DLL. If some app or other library needs to use this library, just add the following command to the file `Android.mk`.

`LOCAL_SHARED_LIBRARIES := libmylib`

4 FAQ

No.	Description	Solution
1	If some projects can run in Linux platform stably, can they directly run in Android platform?	Even for the same CPU and the same toolchain, the biggest problem of running a Linux application on Android is libc. Android uses bionic as libc library. Compared with gnu libc, bionic lacks implementation of many functions. Therefore, to make applications normally run in Android, please rebuild in AOSP build system.
2	In Linux platform, open source projects usually use cross building. How to build these projects in Android platform?	Firstly, it is recommended to create the Android.mk file for building the project. Please refer to the project in external directory of AOSP. Secondly, adopt the method of building out-of-tree and building recursively, which is very simple and efficient.

Table 5. FAQ List

Appendix: Glossary

Abbr.	Full Name
ADT	GoWarrior Developer Tools
BGA	Ball Grid Array
NDK	Native Development Kit
QFP	Quad Flat Package

Table 6. List of Abbreviations

Revision History

Document Change History

Revision	Changes	Date
v1.1	Updated document version to 1.1.	February 29, 2016
v1.0	Initial Release	September 07, 2015

Table 7. Document Change History

Software Changes

Revision	Changes	Date
v1.1	Install and start GoDroid with a MicroSD card.	February 29, 2016
v1.0	Initial Release	September 07, 2015

Table 8. Software Change History

www.gowarriorosh.com

Headquarters

Tel: +886-2-8752-2000

Fax: +886-2-8751-1001