Building ASxxxx for Android

Martin Taylor

December 2014

Contents

Introduction

The purpose of this document is to provide guidance in the compilation of the ASxxxx cross assemblers for use as native applications running on Android operating system.

Two methods of compilation will be presented. The first method will compile ASxxxx on the Android device itself using Terminal IDE from Spartacus Rex. The resulting binaries are intended for use within the Terminal IDE environment that runs on the Android device. The second method involves cross compiling the sources on a Linux host for use on an Android device within a terminal emulator. Neither method involves any modification (i.e. rooting) of the Android device or special privileges on either the host or target system.

Due to the many versions and OEM customizations of the Android OS these guidelines may require some adaption by the end user.

Building ASxxxx for Android Using Terminal IDE

Terminal IDE is an expandable terminal application, with a full Java, C/C++, HTML, Android development kit, that runs on an Android device. The entire application runs without needing root permissions; gcc, make, ctags, javac, java, dx, aapt, apkbuilder, signer, ssh, sshd, telnetd, bash, busybox, vim and a nice terminal emulator are all available.

This document assumes that you have already downloaded, installed and are familiar with Terminal IDE.

1. Download the ASxxxx Source Code

Download the source code for ASxxxx to the Android target using the device's browser. These instructions will assume the default destination directory of `/sdcard/Download` is used.

2. Relocate the Source Code

Android's security mechanisms give each application a unique user id. A consequence of this is that it may not be possible to unpack the sources in the location where it was downloaded. To resolve this we will produce a second copy with the necessary permissions within Terminal IDE.

Start Terminal IDE and enter the following commands on the command line:

```
cd $HOME
cat /sdcard/Download/asxs5p10.zip > ~/asxs5p10.zip
```

Do not use `cp` (copy) as this often leaves the file permissions unaltered on Android devices.

3. Unpack the Source Code

Within Terminal IDE type the following on the command line:

```
unzip asxx5p10.zip
```

No command line options are required.

4. Add Symbolic Links

The gnu c compiler within Terminal IDE is called terminal-gcc not gcc. To simplify the compilation process we will create symbolic links for the compiler and strip commands .

Within Terminal IDE type the following on the command line:

```
cd ~/system/bin
ln -s terminal-gcc gcc
ln -s terminal-g++ g++
ln -s ~/android-gcc-4.4.0/bin/arm-eabi-strip strip
```

NB: Tab completion is your friend here.

5. Build ASxxxx

Building the source code is now exactly the same as on a standard Linux host.

Within Terminal IDE type the following on the command line:

```
cd ~/asxv5pxx/asxmak/linux/build
make
```

For reference the build time on an unbranded 10in Android tablet powered by a dual core ARMv7 cpu (BogoMIPS: 1429.28/1823.53) with 1GB DDR3 RAM was 7m 47sec.

5. Installation

To install the binaries simply move them to a directory within your $PATH to which you have write access.

```
mv ~/asxv5pxx/asxmak/linux/exe/* ~/local/bin
```

Note: Terminal IDE installs to `/data/data/com.spartacusrex.spartacuside` with the default $HOME pointing to `/data/data/com.spartacusrex.spartacuside/files`. The directory `~/local/bin` should already exist and be in your path. If not you will need to create it and update your path in `~/.init` using vim.

6. Verify Installation

Within Terminal IDE type the following on the command line:

```
cd ~
aslink –h
as8051 -h
asavr –h
etc.
```

Congratulations! You now have a working copy of ASxxxx installed on your Android device.

7. Usage

The terminal IDE environment is easy to use for anyone coming from a linux background with cli tools experience, as the vim editor allows users to invoke make without leaving the editor, but it's not to everyone's liking.

It is possible to edit the files externally using an Android text editor of your choice, for example 920 Text Editor or DroidEdit, then assemble and link the files from the command line. Switch between the editor and the terminal window using the icon or recent apps history. Terminal IDE will suspend, but not unload from memory, while using the editor.

It is also possible to telnet into the Android device from a computer, consult the Terminal IDE documentation for details, and use the PC's screen and keyboard. This option is of most use when the user cannot install software onto the PC.

With any of the methods used it is preferable to edit the files in a location that can be seen by the editor, file manager and Terminal IDE. This generally means using the internal sdcard. Files created by Terminal IDE and text editors usually grant everyone read/write permissions, some user experimentation may be necessary. All other aspects of using the asxxxx assembler and linker are as described in the documentation.

8. Install the Documentation

To install the documentation simply move the asxhtml folder to a location of your choice from within Terminal IDE by typing:

```
mv ~/asxv5pxx/asxhtml /sdcard/documents/asxhtml
```

The directory /sdcard is located on the internal flash of most tablets, it is often a symbolic link to /mnt/sdcard. The external storage device may be /mnt/extsd or /mnt/external_sd but there may be no write access to this from user installed applications on devices running Android 4.0 (Ice Cream Sandwich) and later.

Alternatively, your device may have a vendor supplied file manager or other utilities capable of unzipping the archive and copying the documentation to a location of your choice. The restrictions on writing to external sdcards do not normally apply to vendor supplied file managers, however, Android 4.4 (KitKat) has issues with this. Terminal IDE is currently incompatible with Android 5.0 (Lollipop).

9. Viewing the Documentation

The documentation may be viewed by navigating to it using the file manager and associated viewer. Alternatively, the documentation may be bookmarked in a browser of your choice by using a url in the following format:

```
file:///sdcard/documents/asxhtml/asxdoc.html    ;internal sdcard example
file:///mnt/extsd/proglang/asxhtml/asxdoc.html  ;external example
```

Note 1: Many third party browsers do not support the `file://` scheme, however, the vendor supplied browser on all of the tablets I have tried to date have supported it.

Note 2: Terminal IDE includes the text mode web browser 'Links' which does support the `file://` scheme. It was possible to keep Links running in a second terminal window with the ASxxxx documentation loaded and swipe between the terminal sessions. Links was able to load pages from both internal and external storage on the system under test.

10. Clean Up

Once the binaries and documentation are installed and verified we can recover some valuable space by deleting the source code and downloaded archives.

Within Terminal IDE type the following on the command line:

```
rm -r ~/asxv5pxx
rm ~/asxs5p10.zip
rm /sdcard/Download/asxs5p10.zip
```

If there are any problems deleting the archive from the /sdcard/Download directory use the vendor supplied file manager to delete it.

Building ASxxxx for Android Using the NDK

Terminal IDE provides a nice environment for running the ASxxxx assemblers and Android code development in general but may be overkill for users who only wish to deploy ASxxxx on their Android device; it also has a large footprint (212MB). An alternative is to cross compile the assemblers on a Linux PC, using the Android Native Development Kit (NDK), then run them on an Android device under a terminal emulator such as Jack Palevich's '*Terminal Emulator for Android*'.

The procedure contained in this section will detail the cross compilation of the ASxxxx assemblers as dynamically linked, stripped, native binaries for Android using the NDK on a Linux PC.

This document makes the following assumptions:

- the user has already downloaded, installed and is familiar with Terminal Emulator for Android and the limitations of their shell.
- that the user is sufficiently knowledgeable in the use of Linux commands to adapt these guidelines to their own device.
- the user does not have root access on the Linux host PC.
- the user's Android target is not rooted.

On all of the devices tested the default shell was found to be the Bourne shell (sh) not the bash shell as used in Terminal IDE and Linux distributions. For help with some of the peculiarities of using a terminal emulator on an Android device refer to '*Appendix V - Bourne Shell and BusyBox Tips*'.

1. Download the Android NDK

The Android NDK is available from the Android Developer's website, at the time of writing the current version was Revision 10d. Seperate downloads of the NDK are available for Linux (x86), Mac OSX and Windows hosts in both 32-bit and 64-bit editions. See '*Appendix IV - Android NDK Overview*' for more details.

Note: The ndk downloads are 430MB - 500MB in size and the current release (Revision 10d) unpacks to 3.4GB

2. Install the Android NDK

Previous releases of the NDK were supplied as tar.bz2 files, since release 10 self extracting .bin files are used. This guide will show how to install both kinds of packages.

The details on how to install the current package are listed on the Android Developer's NDK page. The procedure on the NDK page instructs the user to unpack the distribution in the location

where it was downloaded, this may be undesireable. The procedure detailed below will assume that all downloaded files are in `$HOME/downloads` and that all packages are to be installed in `$HOME`. No root access or special privaledges are required on the host PC.

A. Binary Package

To install a binary package open a terminal window, give the downloaded package executable permissions then execute the package. For example:

```
cd ~
chmod 755 ~/downloads/android-ndk-r10d-linux-x86.bin
~/downloads/android-ndk-r10d-linux-x86.bin
```

The package containing the NDK extracts itself into a folder below the current directory. In this example it will be `~/android-ndk-r10d`.

Note that you can also use a program like 7z to extract the package.

B. Tarball Package

To install a tarball (.tar.bz2) package open a terminal window and untar it where required. For example:

```
cd ~
tar -xjf ~/downloads/android-ndk-r8e.tar.bz2
```

The package containing the NDK is extracted into a folder below the current directory. In this example it will be `~/android-ndk-r8e`.

3. Build a Standalone Toolchain

The NDK provides multiple compilers, for multiple architectures and multiple APIs. While it is possible to invoke the required compiler directly this involves lengthy command line statements. It is much simpler to build a standalone toolchain for the target device and specific API. The procedure detailed below will provide instructions to build a stand alone toolchain for Android 4.2.2 (API 17, JELLY_BEAN_MR1) and install it into the `$HOME/android-17` directory.

In a terminal window type the following:

```
cd ~/android-ndk-r10d
./build/tools/make-standalone-toolchain.sh --platform=android-17    \\
      --install-dir=$HOME/android-17 --arch=arm
```

Note 1: The `make-standalone-toolchain.sh` script does not like the tilde character in the pathname. Use the shell variable `$HOME` instead.

Note 2: Earlier versions of the make standalone toolchain script created a toolchain for arm cores by default and did not require the `--arch=arm` option.

Note 3: To determine which API version is required see '*Appendix III - Android API Versions*' .

This step may be repeated for each of the NDK API's that you wish to support by replacing the number 17 shown in the command line with the appropriate API number.

4. Create Symbolic Link

The ASxxxx Android NDK Cross Compiler makefile is written with:

```
TOOL_ROOT=$(HOME)/android-toolchain
```

By creating a symbolic link between `$(HOME)/android-17` and `$(HOME)/android-toolchain` we can alter the target device without editing the makefile or rebuilding toolchains by deleting the symbolic link then recreating it to point to a different toolchain.

To create the symbolic link type:

```
ln -s ~/android-17 ~/android-toolchain
```

To delete the link type:

```
rm ~/android-toolchain
```

5. Build ASxxxx

*** DRAFT NOTE ***

I am assuming here that the makefile in Appendix VII is in the asxmak/android/build directory as per asxxxx build conventions

*** DRAFT NOTE ***

This procedure will assume that the ASxxxx source tarball is located in `$HOME/downloads`.

In a terminal window type the following:

```
cd ~
unzip -L $HOME/downloads/asxv5p10.zip
cd ~/asxv5pxx/asxmak/android/build
make clean all
```

The cross assemblers and linker binaries will be found in `~/asxv5pxx/asxmak/android/exe`. To create a tarball containing these binaries type:

```
cd ~/asxv5pxx/asxmak/android/exe
tar -cvzf ../asxv5p10-android-17.tar.gz ./
```

This will create an archive containing all of the binaries located at:

```
~/asxv5pxx/asxmak/android/asxv5p10-android-17.tar.gz
```

Alternatively use:

```
cd ~/asxv5pxx/asxmak/android/exe
tar -cvjf ../asxv5p10-android-17.tar.bz2 ./
```

This will create an archive containing all of the binaries located at:

```
~/asxv5pxx/asxmak/android/asxv5p10-android-17.tar.bz2
```

6. Installation

Installation of the ASxxxx assemblers and linker involves copying the binary executables to the android device and placing them into a folder with executable permissions.

This procedure will assume the following:

* The use of '*Terminal Emulator for Android*' by Jack Palevich

* All required symlinks have been set up in advance.

* The ASxxxx binaries tarball is located at `/sdcard/Download`

For more information on the shell see '*Appendix V - Bourne Shell and BusyBox Tips*' and '*Appendix VI - Terminal Emulator for Android*'.

Start the terminal emulator on the Android device and enter the following commands on the command line:

```
cd /data/data/jackpal.androidterm/local/bin
busybox tar -xvzf /sdcard/Download/asxv5p10-android-17.tar.gz
```

7. Verify Installation

Within Terminal IDE type the following on the command line:

```
cd ~
aslink –h
as8051 -h
asavr –h
etc.
```

Congratulations! You now have a working copy of ASxxxx installed on your Android device.

8. Usage

The shell of the terminal emulator is not the easiest of environments to work in. The commands available are determined by the vendor at compile time and an editor may not even be present.

It is possible to edit the files externally using an android text editor of your choice, such as '920 Text Editor' or 'DroidEdit', then assemble and link the files from the command line. Switching between the editor and the terminal window is achieved by using an icon or recent apps history. 'Terminal Emulator for Android' will suspend, but not unload from memory, while using an android editor application.

It is preferable to edit the files in a location that can be seen by the editor, file manager and the terminal emulator. This generally means using the internal sdcard. Files created by text editors usually grant everyone read/write permissions, some user experimentation may be necessary. All other aspects of using the asxxxx assembler and linker are as described in the asxxx documentation.

9. Install the Documentation

To install the documentation copy the asxhtml folder from the host PC to a location of your choice on the android device using the android file manager. There are restrictions on writing to external sdcards using third party apps which do not normally apply to vendor supplied file managers, however, Android 4.4 (KitKat) has issues with this.

10. Viewing the Documentation

The documentation may be viewed by navigating to it using the file manager and associated viewer. Alternatively, the documentation may be bookmarked in a browser of your choice by using a url in the following format:

```
file:///sdcard/documents/asxhtml/asxdoc.html    ;internal sdcard example
file:///mnt/extsd/proglang/asxhtml/asxdoc.html  ;external example
```

Note 1: Many third party browsers do not support the file:// scheme, however, the vendor supplied browser on all of the tablets I have tried to date have supported it.

11. Android Clean Up

Once the binaries and documentation are installed and verified we can recover some valuable space by deleting the downloaded archives.

Within the android terminal emulator type the following on the command line:

```
rm /sdcard/Download/asxv5p10-android-17.tar.gz
```

If there are any problems deleting the archive from the /sdcard/Download directory use the vendor supplied file manager to delete it.

12. PC Clean Up

This procedure assumes that you no longer wish to retain the NDK but do wish to retain the standalone tool chain.

On the linux host PC open a terminal window and type the following:

```
cd ~
rm -rf ~/android-ndk-r10d
rm -rf ~/asxv5pxx
rm ~/asxs5p10.txt
```

If you no longer wish to retain the standalone toolchain type:

```
rm ~/android-toolchain
rm -rf ~/android-17
```

Congratulations on a job well done.

Building GNU Make for Android Using the NDK

GNU Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files. Make is used to build the asxxxx assemblers and can be used for building your own projects. While *'Terminal IDE*' ships with a version of make your android device most likely does not.

This procedure details how to cross compile gnu make on a linux PC using the android NDK to produce a stripped, binary executable suitable for android devices. It also provides a general insight into the process of cross compiling command line utilities for android.

The normal process of building applications for linux involves obtaining the source code, unpacking the archive, running configure then make all install. The autoconf and automak tools are not present on android nor are they available in '*Terminal IDE*' so the process must be carried out on a linux PC.

This procedure will use the android standalone toolchain and symbolic links created previously during steps 1 to 4 of '*Building ASxxxx for Android Using the NDK*'.

Open a terminal window on the linux box and type the following:

```
cd ~
wget http://ftp.gnu.org/gnu/make/make-3.82.tar.bz2
tar -xvjf make-3.82.tar.bz2
cd make-3.82
mkdir android
cd android
../configure CC=$HOME/android-toolchain/bin/arm-linux-androideabi-gcc \\
             --host=arm-linux
make
```

And the build fails! Welcome to the world of cross-compiling.

The line containing the fatal error reads:

```
../arscan.c:257:18: fatal error: ar.h: No such file or directory
```

To fix this we need to edit the autogenerated Makefile (`~/make-3.82/android/Makefile`). Open up the Makefile in your editor of choice and search for the line that begins with:

```
DEFS =   -DLOCALEDIR
```

then add `-DNO_ARCHIVES` at the end of the line, save the Makefile and return to the terminal window. Rerun make:

```
make clean
make
```

Success!

To inspect the file type:

```
file -b make
```

and you should see:

```
ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses
shared libs), not stripped
```

To strip the executable type:

```
~/android-toolchain/bin/arm-linux-androideabi-strip make
```

Running 'file -b make' should now show:

```
ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses
shared libs), stripped
```

The executable may now be copied to the android device and installed into a directory included in the `$PATH`.

Appendix I - Software

ASxxxx Cross Assemblers, Version 5.10, October 2014
http://shop-pdp.net/ashtml/asxxxx.htm
The ASxxxx assemblers are a series of microprocessor assemblers written in the C programming language.

Terminal IDE, Version 2.02
www.spartacusrex.com/terminalide.htm
https://play.google.com/store/apps/details?id=com.spartacusrex.spartacuside
** CURRENTLY INCOMPATIBLE WITH ANDROID 5.0 LOLLIPOP **

Terminal Emulator for Android, Version 1.0.53
Terminal Emulator for Android, Version 1.0.62
https://play.google.com/store/apps/details?id=jackpal.androidterm
Jack Palevich
Terminal Emulator for Android is a VT-100 compatible terminal emulator for Android devices.

920 Text Editor, Version 13.7.18
https://play.google.com/store/apps/details?id=com.jecelyin.editor
Qooyee.com
920 Text Editor is a text and source code editor with syntax highlighting and multi tab interface for Android tablets and phones.

BusyBox, Version 1.11.1
BusyBox, Version 1.16.0
BusyBox, Version 1.18.3
http://busybox.net/
BusyBox combines tiny versions of many common UNIX utilities into a single small executable.

Android NDK, Revision 8e (March 2013)
Android NDK, Revision 9d (March 2014)
Android NDK, Revision 10d (December 2014)
http://developer.android.com/tools/sdk/ndk/index.html
Officially the NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. It is also possible to use the NDK and supplied libraries to compile native command line utilities.

GNU Make, v3.82
http://www.gnu.org/software/make/
GNU Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.

Appendix II - Hardware

This appendix details the hardware used for testing the cross compiled asxxxx binaries.

Wondermedia WM8650 7" Notebook

OS:	Android 2.2 (`FROYO, API-8`)
RAM:	256MB
CPU:	wm8650, arm926ej-s rev 5, armv5tej
BogoMIPS:	797.97

G-pad 7.0 EXplorer 7 Tablet

OS:	Android 4.0.4 (`ICE_CREAM_SANDWICH_MR1, API-15`)
RAM:	512MB
CPU:	Allwinner A13, cortex-a8, armv7 rev 2
BogoMIPS:	1001.88

G-pad 8.0 EXtreme I Tablet

OS:	Android 4.0.3 (`ICE_CREAM_SANDWICH_MR1, API-15`)
RAM:	1GB
CPU:	rk2918, cortex-a8, armv7 rev 2
BogoMIPS:	407.77 - 1199.34 (variable clock 0.4 - 1.2 GHz)

Fusion5 xtra compact 10.1 Tablet

OS:	Android 4.2.2 (`JELLY_BEAN_MR1, API-17`)
RAM:	1GB
CPU:	dual-core cortex-a7, armv7 rev 4
BogoMIPS:	1429.28/1823.53

Appendix III - Android API Versions

Platform Version	API Level	VERSION_CODE
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2.2	17	JELLY_BEAN_MR1
Android 4.2	17	JELLY_BEAN_MR1
Android 4.1.1	16	JELLY_BEAN
Android 4.1	16	JELLY_BEAN
Android 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0.3	15	ICE_CREAM_SANDWICH_MR1
Android 4.0.2	14	ICE_CREAM_SANDWICH
Android 4.0.1	14	ICE_CREAM_SANDWICH
Android 4.0	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4	10	GINGERBREAD_MR1
Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2	9	GINGERBREAD
Android 2.3.1	9	GINGERBREAD
Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

Source: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels

Appendix IV - Android NDK Overview

The Android Native Development Kit (NDK) includes a set of cross-toolchains (compilers, linkers, etc.) that can generate native ARM binaries on Linux, OS X, and Windows (with Cygwin) platforms. It provides a set of system headers for stable native APIs that are guaranteed to be supported in all later releases of the platform:

- libc (C library) headers
- libm (math library) headers
- JNI interface headers
- libz (Zlib compression) headers
- liblog (Android logging) header
- OpenGL ES 1.1 and OpenGL ES 2.0 (3D graphics libraries) headers
- libjnigraphics (Pixel buffer access) header (for Android 2.2 and above).
- A Minimal set of headers for C++ support
- OpenSL ES native audio libraries
- Android native application APIS

The NDK supports the following instruction sets:

- ARMv5TE, including Thumb-1 instructions
- ARMv7-A, including Thumb-2 and VFPv3-D16 instructions, with optional support for NEON/VFPv3-D32 instructions

ARMv5TE machine code will run on all ARM-based Android devices. ARMv7-A will run only on devices such as the Verizon Droid or Google Nexus One that have a compatible CPU. The main difference between the two instruction sets is that ARMv7-A supports hardware FPU, Thumb-2, and NEON instructions. You can target either of the instruction sets — ARMv5TE is the default

Contrary to the instructions on the Android Developer website it is not necessary to install the Android Software Development Kit (SDK) to use the NDK. The SDK is required to build standard applications that use the Java Native Interface (JNI) and native code to implement part of the application. This document is only concerned with building applications entirely in native code and does not require the SDK.

The two most important points to realize here are that the NDK "provides a set of system headers for stable native APIs that are guaranteed to be supported in all later releases of the platform" and "ARMv5TE machine code will run on all ARM-based Android devices" so the latest NDK may not be necessary. Since this document is concerned with command line utilities which only use the standard c libraries it is quite probable that the NDK version is not critical.

During initial testing for this document NDK r8e was used to build binaries for Android 2.2 (API 8, FROYO) on an armv5te cpu and Android 4.0.3 (API 15, ICE_CREAM_SANDWICH_MR1) on an armv7 core. Despite the fact that NDK r8e support only extends to Android 4.0.2 (API 14, ICE_CREAM_SANDWICH) the binaries produced were found to be identical. The binaries produced by the Android 2.2 standalone toolchain were loaded into the Android 4.0.2 device and executed without incident.

NDK Revisions

Android NDK, Revision 10d (December 2014)
Android NDK, Revision 10c (October 2014)
Android NDK, Revision 10b (September 2014)
Android NDK, Revision 10 (July 2014)
Android NDK, Revision 9d (March 2014)
Android NDK, Revision 9c (December 2013)
Android NDK, Revision 9b (October 2013)
Android NDK, Revision 9 (July 2013)
Android NDK, Revision 8e (March 2013)
Android NDK, Revision 8d (December 2012)
Android NDK, Revision 8c (November 2012)
Android NDK, Revision 8b (July 2012)
Android NDK, Revision 8 (May 2012)
Android NDK, Revision 7c (April 2012)
Android NDK, Revision 7b (February 2012)
Android NDK, Revision 7 (November 2011)
Android NDK, Revision 6b (August 2011)
Android NDK, Revision 6 (July 2011)
Android NDK, Revision 5c (June 2011)
Android NDK, Revision 5b (January 2011)
Android NDK, Revision 5 (December 2010)
Android NDK, Revision 4b (June 2010)
Android NDK, Revision 3 (March 2010)
Android NDK, Revision 2 (September 2009)
Android NDK, Revision 1 (June 2009)

Source: http://developer.android.com/tools/sdk/ndk/index.html#Revisions

There is no definative list of which API levels are supported in each NDK release. Neither is there a list detailing which compilers are included and which one is the default. For this and other details the user is required to either read through all of the change lists online or download and unpack a release to gain access to the documentation for that release.

From what I have been able to deduce from my own observations it would appear that a release number change indicates a change in the APIs supported by a release, while a suffix letter change indicates changes to the compilers included in the release and bug fixes.

android-ndk-r8e-linux-x86.tar.bz2 (Linux 32-bit x86 host) (March 2013)

Filesize: 440M
Footprint: 1.4G
APIs: 3, 4, 5, 8, 9, 14
Targets: arm, mips, x86
Compilers: gcc v4.4.3, 4.6 (default), 4.7; clang v3., 3.2

android-ndk-r9d-linux-x86.tar.bz2 (Linux 32-bit x86 host) (March 2014)

Filesize: 440M
Footprint: 1.4G
APIs: 3, 4, 5, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19
Targets: arm, mips, x86
Compilers: gcc v4.6 (default), v4.8; clang v3.3, 3.4

android-ndk-r10d-linux-x86.bin (Linux 32-bit x86 host) (December 2014)

Filesize: 429M
Footprint: 3.4G
APIs: 3, 4, 5, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 21
Targets: arm, mips, x86
Compilers: gcc v4.6, 4.8 (default), v4.9; clang v3.4, 3.5 (32-bit and 64-bit compilers)

Obtaining Older NDK Releases

The Android Developer site only includes a link to the current release of the NDK. There is no obvious link to any previous releases. After searching many forums the following link was revealed:

```
https://dl.google.com/android/ndk/<filename>
```

where filename is of the format:

```
android-ndk-<rel>-<host_os>-<host_cpu>.<ext>
```

`rel` = release number i.e r8e
`host_os` = linux | darwin | windows
`host_cpu` = x86 | x86_64
`ext` = tar.bz2 | zip | bin | exe

The extensions changed at or just after r10. Earlier releases were distributed as tar.bz2 for linux and OSX (Mac), zip was used for windows (Cygwin 1.7) releases. Later releases were distributed as bin for linux and OSX, exe was used for windows releases. The bin and exe files are self extracting archives.

Appendix V - Bourne Shell and BusyBox Tips

On all of the devices tested the default shell was found to be the Bourne shell (sh) not the bash shell as used in Terminal IDE and Linux distributions. This shell and it's default configuration are not part of the terminal emulator, they are device specific and configured by the vendor at compile time.

Many of the common Linux commands are often provided by multitools such as busybox or toolbox and symbolic links for these command are not always present. To determine which version of busybox, if any, you are running type 'busybox' in the terminal window on your android device...

```
BusyBox v1.11.1 (2009-03-11 09:17:49 CST) multi-call binary
Copyright (C) 1998-2008 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: function [arguments]...

    BusyBox is a multi-call binary that combines many common Unix
    utilities into a single executable.  Most people will create a
    link to busybox for each function they wish to use and BusyBox
    will act like whatever it was invoked as!

Currently defined functions:
    [, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, brctl,
    bunzip2, bzcat, bzip2, cal, cat, catv, chgrp, chmod, chown, chpasswd,
    .
    .
    .
    uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, wc,
    wget, which, who, whoami, xargs, yes, zcat, zcip
```

Just because a function is defined does not mean it will be useable. Android devices lack many of the features and files of a full linux distribution. Furthermore, the Android security layers enforce additional restrictions on port use (ports 0 to 1024 are reserved for system use only)and storage access.

If no symbolic link (symlink) is installed for a command any attempt to execute the command will fail i.e.

```
whoami
/system/bin/sh: whoami: not found
```

Try running the command again by typing busybox in front of it:

```
busybox whoami
whoami: unknown uid 10070
```

If a symlink is created to the Busybox executable using the name of one of the commands it

supports, it will act as that command automatically:

```
cd ~
ln -s busybox whoami
./whoami
whoami: unknown uid 10070
```

This allows you to create a bunch of symlinks or hardlinks to the Busybox executable, add them to your $PATH, and let a single Busybox provide a standard set of command line tools .

```
ln -s /system/bin/busybox /data/data/jackpal.androidterm/local/bin/cp
```

Note: Not all devices have busybox installed. Toolbox is a similar tool but with far fewer commands supported.

Appendix VI - Terminal Emulator for Android

Terminal Emulator for Android is a VT-100 compatible terminal emulator for Android devices. To get the best experience from the terminal emulator a certain amount of configuration and customization is required.

For the terminal emulator to invoke a program from anywhere within the system several conditions must be met:

- the location of the program must have executable permissions
- the shell must have executable permissions to the program
- the program must be in the PATH

A. Executable Locations

From the perspective of the terminal emulator the only location(s) where we can create executable files is within the terminal emulator's own private directories. For Jack Pavelich's 'Android Terminal Emulator' this is located at:

```
/data/data/jackpal.androidterm
```

and any of its subdirectories. For convenience it is recommended that 'local/bin' folders are created from this point to hold symbolic links and user programs:

```
mkdir /data/data/jackpal.androidterm/local
mkdir /data/data/jackpal.androidterm/local/bin
ln -s /system/bin/busybox /data/data/jackpal.androidterm/local/bin/cp
```

Binaries located on the sdcard cannot be executed by the shell. The exception to this rule is bash script files which can be invoked using the 'dot' command e.g.

```
. filename
. /sdcard/documents/testme.sh
```

such scripts are not marked as executable, and do not need to be.

B. Executable Permissions

Files copied over to an android device are generally owned by system groups and not by the terminal emulator. Attempting to run chmod or chown on them will often fail. The most common solution to this problem is to 'cat' the file into the application's directories, this gives the emulator ownership of the file, then use 'chmod' on it as usual. Copying does not work on some devices as the owner does not get changed.

```
cd ~
cat /sdcard/Download/myprog > myprog
chmod 755 myprog
./myprog
```

C. Setting the $PATH variable

The terminal emulator environment can be configured from the preferences screen accessable from the menu. Select menu -> Preferences then scroll down to the 'SHELL' options:

Command Line: /system/bin/sh

Initial Command: export PATH=/data/data/jackpal.androidterm/local/bin:$PATH

HOME folder: /data/data/jackpal.androidterm/app_HOME

After making changes to these settings it is advisable to exit and restart the terminal emulator.

Appendix VII - Android NDK Makefile

```
android directory created under asxmak by copying linux directory

Changes made to /asxv5pxx/asxmak/android/build/makefile

Find and replace 'strip' with $(STRIP)'
Replace 'DSTEXE:= $(ASXBAS)asxmak/linux/exe/'
with 'DSTEXE:=    $(ASXBAS)asxmak/android/exe/'

Changes to CC= and LD=
CC=   $(TOOL_ROOT)/bin/arm-linux-androideabi-gcc
LD=   $(TOOL_ROOT)/bin/arm-linux-androideabi-gcc

New definitions:
TOOL_ROOT=  $(HOME)/android-toolchain
STRIP=            $(TOOL_ROOT)/bin/arm-linux-androideabi-strip

#########################################################################
#  Makefile
#
#           ANDROID NDK CROSS COMPILER Version
#           ==================================
#
#   12 December 2014               Created for LINUX     / make /
#
#########################################################################

# TOOL_ROOT specifies the location on the linux host of the
# Android NDK standalone toolchain

TOOL_ROOT=  $(HOME)/android-toolchain

CC=               $(TOOL_ROOT)/bin/arm-linux-androideabi-gcc
CCOPT=            -O3
CCFLAGS=    -Wall -funsigned-char $(CCOPT)

LD=               $(TOOL_ROOT)/bin/arm-linux-androideabi-gcc
LDFLAGS=    -Wall

STRIP=            $(TOOL_ROOT)/bin/arm-linux-androideabi-strip

#  Set ASXBAS relative to this make file.

ASXBAS:=    ../../../

DSTEXE:=    $(ASXBAS)asxmak/android/exe/

==8><=== snipped =======

### end ################################################################
```