Android System Development

By Team Emertxe

Table of Content

What is Android?

“Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in Java language environment”

- Android is an OS **primarily** designed for mobile devices
- Built on Linux Kernel
- Java, the main **programming** language
- Android is **Open** and **Free**
- Most of the code available under **Apache License**
- Developed and distributed by Google
- Managed by Open Handset Alliance (OHA)

History
(Version And Features)

Version	Release Date	Features
1.6 (Donut)	15 Sep 2009	• Improved Android Market experience • Gallery now enables users to select multiple photos for deletion • Support for CDMA and GSM • Support for more screen sizes and resolutions • Quick search box with updated Voice Search, integration with native applications • Updated technology support text-to-speech engine

History
(Version And Features)

Version	Release Date	Features
2.0/2.1 (Eclair)	26 Oct 2009	• New capabilities in Accounts, contacts & Sync Management • Quick Contact • Optimized hardware speed • New Browser UI and HTML 5 support • Improved Google Maps 3.1.2 • MicrosoftExchage support • Built in flash support for Camera • MotionEvent class enhanced to track multi-touch events • Improved virtual keyboard • Bluetooth 2.1 • Live Wallpapers/Folder

History
(Version And Features)

Version	Release Date	Features
2.2 (Froyo)	20 May 2010	• OS speed, memory, and performance optimizations • Dalivk JIT Complier • Improved application launcher with shortcuts to Phone and Browser applications • USB tethering and Wi-Fi hotspot functionality • Automatic App update • Quick switching between multiple keyboard languages and their dictionaries • Support for numeric and alphanumeric passwords • Support for file upload fields in the Browser application • Adobe Flash10.1 support

History
(Version And Features)

Version	Release Date	Features
2.3 (Gingerbread)	6 Dec 2010	• Updated user interface design • New audio effects • Support for Near Field Communication • Redesigned multi-touch software keyboard • Enhanced support for native code development • Audio, graphical, and input enhancements for game developers • Concurrent garbage collection for increased performance • Native support for more sensors (such as gyroscopes and barometers) • A download manager for long running downloads • Improved power management and application control

History
(Version And Features)

Version	Release Date	Features
3.1 and 3.3 (Honeycomb)	22 Feb 2011	• It was the first Android OS version which was specifically designed for tablets • It adds toolbars at top and bottom and incorporates tabbed browsing and other desktop features • • Honeycomb uses Microsoft's Media Transfer Protocol (MTP) for file transfer rather than connecting as a USB mass storage device

EMERTXE

History
(Version And Features)

Version	Release Date	Features
4.0(Ice Cream Sandwich)	19 Oct 2011	• New typeface called Roboto • New Face unlock feature • Android Beam – A secure NFC powered content sharing platform • Re-arrangeable folders, Favorites Tray, Screenshots • Swipe to dismiss notifications, tasks and browser tabs • W-Fi Direct • Manage apps running in background • Revamped Gmail interface

History
(Version And Features)

Version	Release Date	Features
4.1, 4.2 & 4.3 (Jelly Bean)	27 Jun 2012	• User interface has been made faster and smoother under Project Butter • Notification center has been improved with expandable and actionable notifications • Off line voice recognition and typing facility • Better Google Voice Search • Support for external Braille input in enhanced Accessibility options • Android Beam with enhanced option to transfer photos and videos • Google Now • App encryption and Smart App updates

History
(Version And Features)

Version	Release Date	Features
4.4 (KitKat)	Nov 2013	• Faster Multi-tasking • A smarter caller id • Print where ever, when ever. • Pick a file, any file

History
(Version And Features)

Version	Release Date	Features
5.0 (Lollipop)	Nov 2014	• Support for 64-bit CPUs • Support for print previews • Audio input and output through USB devices • Android Runtime (ART) with ahead-of-time (AOT) compilation and improved garbage collection (GC), replacing Dalvik that uses just-in-time (JIT) compilation

History
(Version And Features)

Version	Release Date	Features
6.0 (Marshmallow)	17 Aug 2015	• Android Pay • Fingerprint Scan Standardization • App Permission Control • App Links • Doze • Home Screen Rotation • Chrome Custom Tab • Auto backup and Store • Status bar icon removal

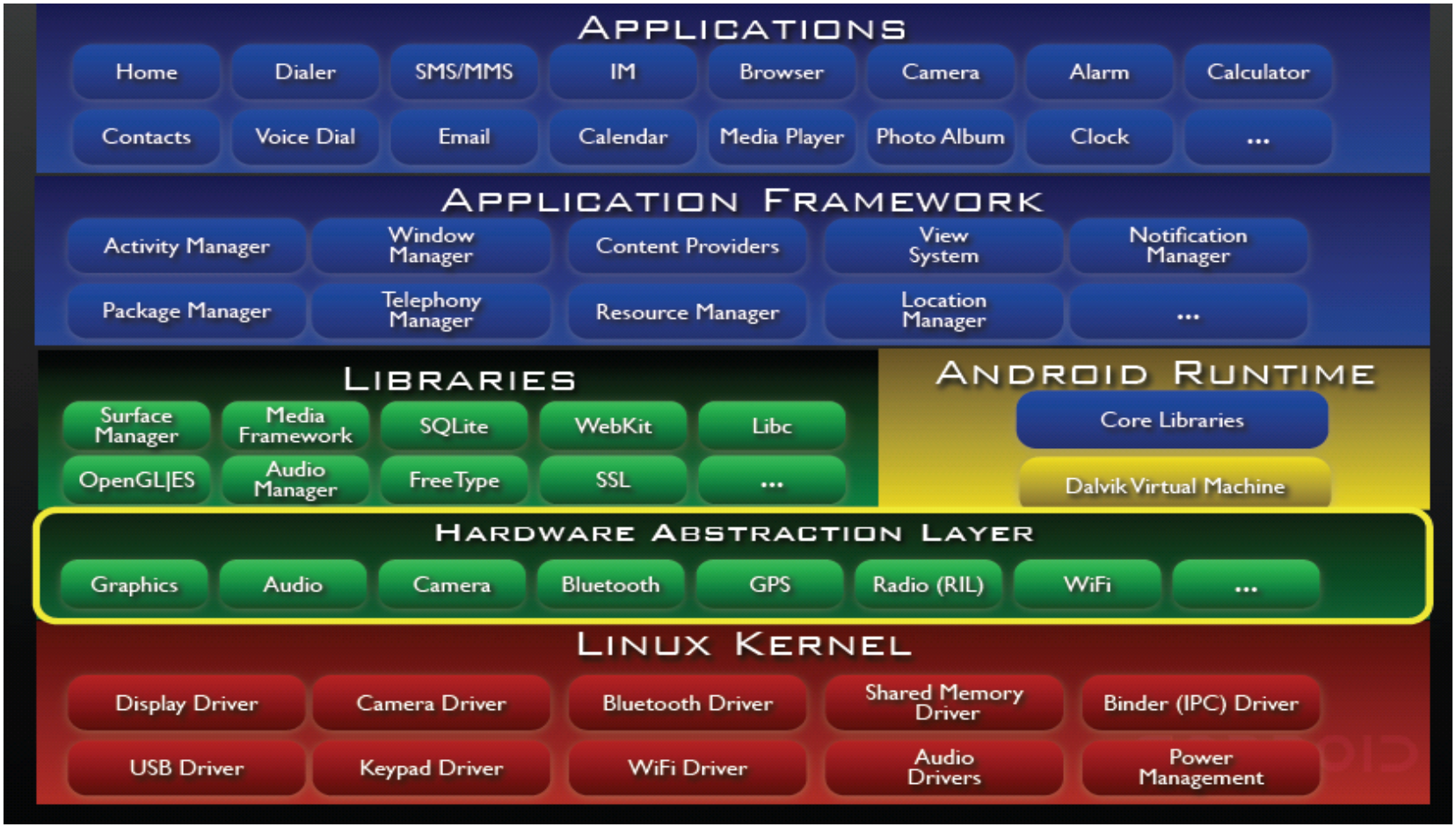
History
(Version And Features)

Version	Release Date	Features
Android N	22 Aug 2016	• Split Screen • Quick Switching beween apps • Notification redesigned • Customizable Quick Setting • Multi-language support • Seamless update

EMERTXE

Software Stack (Architecture)

Android Architecture



EMERTXE

Architecture Detail
(Linux Kernel)

- Android is built on Linux kernel
- Does not include the full set of standard Linux utilities
- It has a core capability like
  - Security
  - Memory management
  - Process management
  - Network stack
  - Driver model
  - Abstraction Layer

Architecture Detail
(Drivers)

- It has a core capability like:
  - Display Driver
  - Camera Driver
  - Bluetooth Driver
  - Flash Memory Driver
  - USB Driver
  - Keypad Driver
  - WiFi Driver
  - Audio Driver
  - Power Management

Architecture Detail
(Libraries)

- Native libraries.
- Written in C/C++ internally
- Called through Java wrapper APIs
- Layer contains
  - Surface Manager (for compositing windows)
  - 2D and 3D graphics
  - Media codecs (MPEG-4, H.264, MP3, etc.)
  - SQL database (SQLite)
  - A native web browser engine (WebKit)

Functional Libraries
(WebKit)

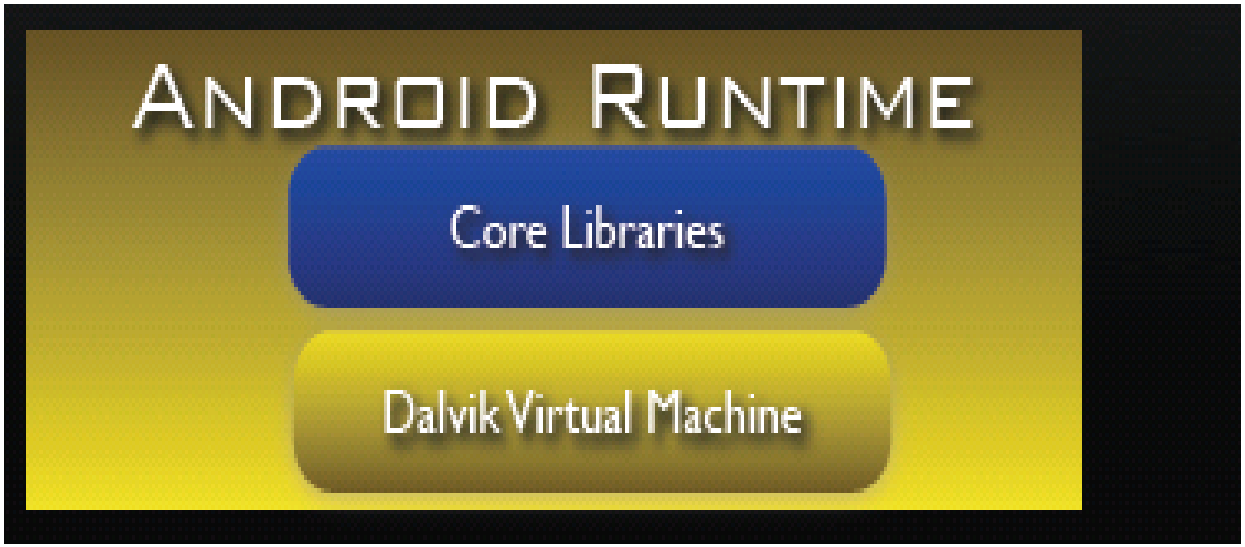
- Open source WebKit browser: http://webkit.org
- Renders pages in full (desktop) view
- Full CSS, Javascript, DOM, AJAX support

Functional Libraries
(SQLite)

- Server less, transactional SQL database engine
- Most widely deployed SQL database engine in the world
- Light-weight transactional data store
- Back end for most platform data storage
- Transactions are **A**tomic, **C**onsistent, **I**solated and **D**urable (ACID) even after system crashes and power failures
- Zero-configuration - no setup or administration needed
- Complete database is stored in single cross-platform disk file
- Cross-platform support :
  - Unix (Linux and Mac OS X)
  - OS/2
  - Windows (Win32 and WinCE)

Dalvik Virtual Machine

- Highly optimized VM, compiled byte is optimized for mobile devices
- **Register** based NOT **Stack** based
- NOT a JVM
- End result is NOT the same byte code as Java, dx in the sdk takes compiled Java class files and converts them into .dex
- Built with security and performances (battery life) in mind

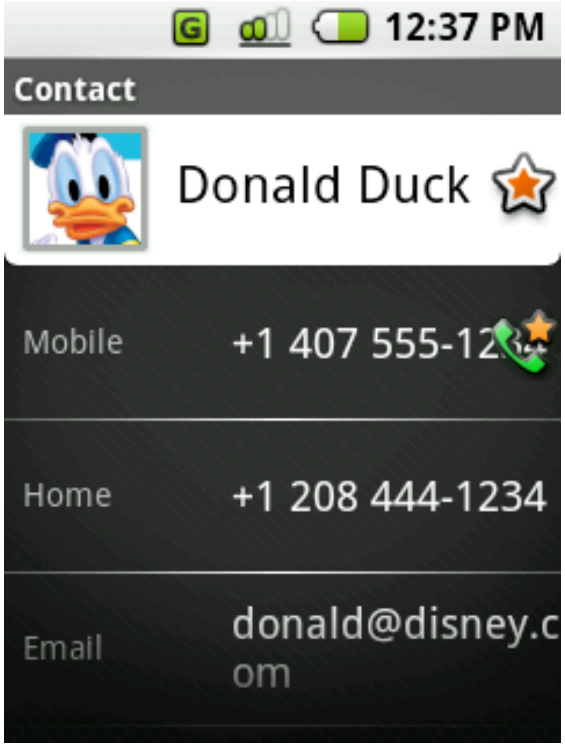


Application Framework

Components	Description
Content Providers	Access, store and share application data
Resource Manager	Access to strings, graphics & layout files
Notification Manager	Display custom alerts in the status bar
Activity Manager	Manages the life cycle of applications
Window Manager	Rendering view on the screen
View System	UI components
Package Manager	Application formation
Location Manager	Location based services

Application Layer

- Collection of built-in and 3rd party apps
- Built-in apps - Phone and Web Browser
- Google and 3rd party apps uses same APIs and goes through same approval process
- Built-in applications can be stopped to use 3rd party apps



EMERTXE

Dalvik Runtime

- **Dalvik** is the virtual machine (VM) in Android OS
- Used on mobile devices such as mobile phones, tablets & netbooks
- Android applications are converted into the compact **Dalvik Executable** (**.dex**)
- Dalvik VM is a register-based architecture
- A tool called **dx** is used to convert some (but not all) Java .class files into the .dex format

Dalvik Runtime

- Multiple classes are included in a single .dex file
- As of Android 2.2, Dalvik has a just-in-time compiler
- Being optimized for low memory requirements, Dalvik has some specific characteristics that differentiate it from other standard VMs:
  - The VM was slimmed down to use less space
  - It uses its own bytecode, not Java bytecode
  - Moreover, Dalvik has been designed so that a device can run multiple instances of the VM efficiently

Dalvik vs JVM

- ***Dalvik Virtual machine** (DVM)*
  - Register Architecture
  - Designed to run on low memory
  - Uses its own byte code
  - Runs .Dex file (Dalvik Executable File
- ***Java Virtual Machine** (JVM)*
  - Stack Architecture
  - Uses java byte code
  - Runs .class file having JIT

Android Runtime

- Android application == a process
- A process == an instance of the Dalvik virtual machine
- Dalvik can run multiple VMs efficiently
- Executes classes compiled by a Java language compiler that have been transformed into the .dex format
- DVM executes files in the Dalvik Executable (.dex) format which is optimized

Android - Bionic

- A Google developed C library
- Based on BSD standard C library
- Does not have full support of POSIX APIs
- Does not support full pthreads
- Has been kept simple, fast and lightweight
- GPL & LGPL code has been kept “out” of user space
- BSD license is provided

Android Kernel Changes

- Wakelocks
- Binder IPC
- Klogger
- Anonymous Shared Memory (ashmem)
- Alarm Timers
- Low memory killer
- ION memory allocator
- Network Security
- Varous drivers
- Various fixes

Android Kernel Changes
(Wakelocks)

- Linux has two power management methods
  - APM (Advanced Power Management)
  - ACPI (Advanced Configuration and Power Interface)
- Android uses none of them
- Android has its own Power Manager (app centric)
- Android Apps have to request “wakelock”
- Wakelock prevent host CPU to go to sleep mode
- If no wakelocks are held, PM will put device in sleep mode

Wakelock	CPU	Screen	Keypad
PARTIAL_WAKE_LOCK	ON	OFF	OFF
SCREEN_DIM_WAKE_LOCK	ON	DIM	OFF
SCREEN_BRIGHT_WAKE_LOCK	ON	BRIGHT	OFF
FULL_WAKE_LOCK	ON	BRIGHT	BRIGHT

Android Kernel Changes
(Binder IPC)

Linux System V IPC

- Signals
- Semaphores
- Message Queues
- Pipes
- Sockets
- Shared Memory

Android IPC

- Binder (system wide)
- Sockets
- Ashmem

- App : Android App component can be started by any process
- Stability : No other process shall get affected if a process misbehaves
- Security : Each process is sandboxed and runs under distinct system ID
- Memory : If not needed, processes can be removed to free memory
- System V IPC is prone to resource leak and instability in view of above requirements (Hence binder IPC is rewritten and used)

Android Kernel Changes
(Binder IPC)

- App : Android App component can be started by any process
- Stability : No other process shall get affected if a process misbehaves
- Security : Each process is sandboxed and runs under distinct system ID
- Memory : If not needed, processes can be removed to free memory
- System V IPC is prone to resource leak and instability in view of above requirements (Hence binder IPC is rewritten and used)
- Binder IPC “Link to death” mechanism – built in reference counting of object and notification of death
- When a binder service is not longer referenced by any clients, its owner is automatically notified that it can dispose of it

Android Kernel Changes
(Klogger)

Android

- A kernel driver
- Uses 4 circular buffers in kernel memory area
- Buffers are exposed in /dev/log directory
- Logcat command to access logs

Linux

- syslog daemon to handle userspace logs
- Syslog () call over socket generates expensive task switching
- Exposed in /var/log directory
- dmesg command to get kernel logs (printk())

Android Kernel Changes
(ashmem)

- Asynchronous Shared Memory (ashmem)
- Open /dev/ashmem
- Supports reference counting so that kernel can reclaim resources which are no longer needed
- Discard shared memory units under memory pressure
- Allow kernel to shrink shared memory region under low memory pressure

- Linux IPC based on shared memory is prone to resource leak
- Can’t be shrinked under low memory pressure

Android Kernel Changes
(Alarm timer)

- Linux has High Resolution Timers that can wake up a process but get missed when system is suspended
- Real Time Clock (RTC) can wake the suspended system but can’t wake a process
- Developed alarm timers by adding patches to RTC and High Resolution times so that suspended system can wake up and applications can accomplish its job

Android Kernel Changes
(ION memory allocator)

- Android Requirement :
  - Unified memory management interface for ease of use (allocate memory in the system for most of the cases)
  - Allow efficient (zero copy) sharing of memory between user space, kernel space & hardware devices

Android Kernel Changes
(ION memory allocator)

- Available memory management mechanisms
  - DMABUF – from Linaro
  - CMEM – from TI
  - PMEM – from Qualcomm
  - NVMAP – from NVIDIA
- Linux usually allocates buffer up to 512 pages (4 KiB / page)
- Linux available heaps (3 types) – vmalloc (virtual), kmalloc (physical) and pre-allocated (reserved) at boot

Android Kernel Changes
(ION memory allocator)

- ION is generalized memory manager
- Introduced in Android 4.0 ICS
- ION removes ARM specific dependencies
- Provides a common structure for how memory will be managed and used by GPU, Audio and Camera drivers
- Default ION driver offers three heaps
  - ION_HEAP_TYPE_SYSTEM – allocation via vmalloc_user()
  - ION_HEAP_TYPE_SYSTEM_CONTIG – allocation via kzalloc()
  - ION_HEAP_TYPE_CARVEOUT - carveout memory is physically contiguous and set aside at boot
- Developers may choose to add more ION heaps
- ION_HEAP_TYPE_IOMMU was added by NVIDIA for hardware blocks equipped with an IOMMU
- User space programs shall open /dev/ion device before it can allocate memory from ION

Android Kernel Changes
(ION memory allocator)

- Memory sharing between user space, kernel & devices without copy
- This is achieved by sharing memory pages directly to avoid copying
- Once memory is allocated successfully from a heap, a file descriptor is returned to user space which can be subsequently used to invoke mmap() to map the allocated pages into user space
- Useful scenario, for Instance, OpenGL can manipulate the memory in user space easily and a GPU can also populate the same piece of physical memory with zero copying

Android Kernel Changes
(Paranoid Network Security)

- In Linux, any app can open the network connection
- Android needs strict polcy for network access control
- Permission based network access
- Filtered with GID

#define	GID	Capability
AID_NET_BT_ADMIN	3001	Can create an RFCOMM, SCO, or L2CAPP Bluetooth socket
AID_NET_BT	3002	Can create a Bluetooth socket
AID_INET	3003	Can create IPv4 or IPv6 socket
AID_NET_RAW	3004	Can create certain kinds of IPv4 sockets??
AID_NET_ADMIN*	3005	Allow CAP_NET_ADMIN permissions for process

Patches in mainline

Following patches/features are added in the mainline kernel (3.10 onward):

- Binder
- Alarm Timers (under the name POSIX Alarm Timers introduced in 2.6.38)
- Ashmem
- Klogger
- Timed GPIOs
- Low Memory Killer
- RAM Console (superseded by pstore RAM backend introduced in 3.5)

Missing Patches

Following patches/features are missing from the mainline kernel (As of 3.10) -

- Paranoid Networking
- ION Memory Allocator
- USB Gadget
- FIQ debugger
- pmem (removed in 3.3)

Directory Structure

Directory Structure
(Android Root)

Root	Sub-directory	Description
android	device	Device specific (Samsung, HTC, LG, etc) stuff
	kernel	Kernel tree for chipsets (msm, mediatek, etc..)
	platform	Android platform source code
	product	For variety of products, currently for brillo
	public-projects	Empty (Android based public projects)
	toolchain	jack, gcc, ndk, llvm, etc..
	tools	repo, aospstats
	trusty	For trusted execution environment components (Google's Trusty OS)

Directory Structure
(Platform)

#	Directory	Description
1	abi	Not very clear, seems to be created for APIs for binary interface; currently it contains C++ files having APIs
2	art	Android Runtime
3	bionic	Android's standard C library
4	bootable	Code samples (bootloader, diskinstaller, recovery)
5	build	The main entry point of the build system
6	cts	contains compatibility test suite
7	dalvik	Dalvik tools and core libraries
8	developers	Not clear, build, demos, samples, docs etc..
9	development	Platform engineering tools and sample apps
10	docs	AOSP documents
11	external	Contains source code for all external open source projects

Directory Structure
(Platform)

#	Directory	Description
12	frameworks	Contains sources of all the frameworks
13	hardware	Hardware related source code such as HAL
14	libcore	Test code for dalvik, DOM, JSON, LUNI (lang util net io) and other support files
15	libnativehelper	VM-agnostic native helper functions
16	ndk	build scripts & helper files for building the NDK
17	out	Build generated directory, output is placed here
18	packages	Standard android apps of AOSP
19	pdk	Reduced set of Android release provided to chipset vendors and OEMs to migrate to new release
20	prebuilts	cross compilations toolchains for different development machines
21	sdk	To compile the SDK. Lot of tools are moved to prebuilts

Directory Structure
(Platform)

#	Directory	Description
22	system	System core files (commands, debugging, bootstat)
23	toolchain	To build and install GNU tools
24	tools	libraries that are sourced in tools/base and tools/swt are converted to prebuilts which are located in prebuilts/devtools
25	vendor	Initially created for vendors (HTC); It seems to be discontinued

Setting up Build Environment

System Requirements

- Intel core i5 or higher
- RAM – 4 ~ 8 GB
  - Recommended 8GB
- HDD – 100 ~ 500 GB
  - Recommended 300GB
- Ubuntu 14.04 LTS or higher
  - Recommended 16.04
- Swap partition - 8 GB
- Optionally enable ccache
  - Recommended for servers

Check Your System

- $ lscpu
- $ free -h
- $ uname -a
- $ lsb_release -a
- $ top
- $ ifconfig

Major Steps

- Choose a branch to build
- Setup Linux build environment
  - Install openJDK (JAVA 8)
  - Installing git
  - Install repo
- Download source
- Install arm toolchain

Installing packages (Ubuntu)

Installing required packages

- $ sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 \
  lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache \
  libgl1-mesa-dev libxml2-utils xsltproc unzip
- $ sudo apt-get install gtkterm
- $ sudo apt-get install adb

*https://source.android.com/source/initializing.html

EMERTXE

Configuring USB Access

- $ wget -S -O - http://source.android.com/source/51-android.rules | sed "s/<username>/$USER/" | sudo tee >/dev/null /etc/udev/rules.d/51-android.rules; sudo udevadm control --reload-rules

*https://source.android.com/source/initializing.html

EMERTXE

Check JAVA version

- Verify Java version (if you already have)
  $ java -version
- You should get following message :-
  openjdk version "1.8.0_111"
  OpenJDK Runtime Environment (build 1.8.0_111-8u111-b14-3~14.04.1-b14)
  OpenJDK 64-Bit Server VM (build 25.111-b14, mixed mode)

Installing openJDK 8

1. $ sudo add-apt-repository ppa:openjdk-r/ppa
2. $ sudo apt-get update
3. $ sudo apt-get install openjdk-8-jdk
   If you have more than one java versions then follow step 4; You would be given options to set the correct version
4. $ sudo update-alternatives --config java
   Set the correct version number
5. $ sudo update-alternatives --config javac
6. Finally, verify the installed version with below command
   $ java -version

***Skip this slide if you already have openJDK 8**

Installing git (ubuntu)

- Run following command

  $ sudo apt-get install git

- Configure git with your real name and email address; you will need an email address that is connected with a registered Google account

  $ git config --global user.email "you@example.com"

  $ git config --global user.name "Your Name"

https://help.ubuntu.com/lts/serverguide/git.html

Installing repo tool

- Repo tool is required to work with Git to work with Android
- Create a bin directory in home folder

  $ mkdir ~/bin

  $ export PATH=~/bin:$PATH

  (or edit .bashrc file and add path)
- Download repo tool and make sure it is executable

  $ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

  $ chmod a+x ~/bin/repo

*https://source.android.com/source/downloading.html

Initializing repo client

- Create a working directory in home folder
  $ mkdir ~/android
  $ cd android
  $ repo init -u https://android.googlesource.com/platform/manifest -b android-7.1.1_r6
- On successful initialization .repo folder shall be created
  $ cd .repo
- Clone git repository
  $ git clone https://github.com/peyo-hd/local_manifests -b nougat
- Download the Android source
  $ repo sync

*https://source.android.com/source/downloading.html

repo sync

- Syncing with repository would take few hours
- Time depneds on your connection speed and server usage
- Ensure there are no or few interruptions in connectivity

*https://source.android.com/source/downloading.html

ΣMERTXE

Verify GCC

- Verify gcc version installed in your machine

  $ gcc --version

- You should get follwing message :-

  gcc (Ubuntu 4.9.4-2ubuntu1~14.04.1) 4.9.4

  Copyright (C) 2015 Free Software Foundation, Inc.

Installing gcc/g++

1. $ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
2. $ sudo apt-get update
3. $ sudo apt-get install gcc-4.9 g++-4.9

Still the gcc will have old symbolic link to gcc-4.8 or lesser

4. $ sudo rm /usr/bin/gcc
5. $ sudo rm /usr/bin/g++

6. $ ln -s /usr/bin/gcc-4.9 /usr/bin/gcc
7. $ ln -s /usr/bin/g++-4.9 /usr/bin/g++

Skip this slide if you already have gcc 4.9 installed

Get gcc-arm-linux-androidgnueabi

1. Check following directory (you must have toolchain)

   $ ls android/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin/

2. Otherwise, get the prebuilt toolchain from Android Developer web site

   https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/+/android-7.1.1_r13

3. Set the toolchain path (if stored in custom user directory)

   $ export PATH=<user directory> : $PATH

   or edit .bashrc and add above export command

Install python mako

- Install python mako module

  $ sudo apt-get install python-mako
- Python is used by build system

Make Heirarchy

Makefile	Root working directory	Top level make file
main.mk	build/core/	Top level build configuration file
config.mk	build/core/	Includes all other sub .mk file
definitions.mk		Standard flag definitions
dex_preopt.mk		Dalvik default options

Make configuration

- **Android.mk** – Platform configuration; to include & build sources for your device
- **AndroidBoard.mk** – Kernel configuration; build system uses that to drop the kernel image in place
- **AndroidProducts.mk** - specify appropriate product make file to be used for building.

  Example – device/brcm/rpi3/AndroidProducts.mk
- **device_codename.mk** - Every device has a codename, and there is a file named as the codename of device. It specifies the properties and extras to copy over into the final output.

  Example: device/brcm/rpi3/rpi3.mk
- **BoardConfig.mk** - This is where compiler conditional flags are set, partition layouts, boot addresses, ramdisk size etc..

  Example : device/brcm/rpi3/BoardConfig.mk

Build Commands

- $ make <package>
  - Builds only the package, instead of going through the entire build
- $ make clean
  - Cleans all the files generated by previous compilations
- $ make clean-<package>
  - Removes all the files generated by the compilation of the given package

Compiling Sources

Kernel Compilation

1. $ cd kernel/rpi
2. $ ARCH=arm scripts/kconfig/merge_config.sh arch/arm/configs/bcm2709_defconfig android/configs/android-base.cfg android/configs/android-recommended.cfg
3. $ ARCH=arm CROSS_COMPILE=arm-linux-androideabi- make zImage
4. $ ARCH=arm CROSS_COMPILE=arm-linux-androideabi- make dtbs

Android Compilation
(envsetup.sh)

Modifies the current environment, it adds many useful shell macros; these macros will serve several purposes:

- Configure and set up the build system
- Ease the navigation in the source code
- Ease the development process
- Some macros will modify the environment variables, to be used by the build system later on
- lunch is a shell function defined in build/envsetup.sh; it sets the environment variables needed for the build

Android Compilation
(lunch)

- lunch is a shell function defined in build/envsetup.sh; it sets the environment variables needed for the build
- TARGET_PRODUCT
  - Which product to build. To build for the emulator, you will have aosp_<arch>
- TARGET_BUILD_VARIANT
  - Select which set of modules to build, among
    - user: Includes modules tagged user (Phone)
    - userdebug: Includes modules tagged user or debug (strace)
    - eng: Includes modules tagged user, debug or eng: (e2fsprogs)
- TARGET_BUILD_TYPE
  - Either release or debug; If debug is set, it will enable some debug options across the whole system

Android Compilation

1. $ source build/envsetup.sh
2. $ lunch rpi3-eng
3. Jack setting (reducing number of jack simultaneous compilations)

   $ vim $HOME/.jack-server/config.properties

   change jack.server.max-service from 4 to 2

   $ export ANDROID_JACK_VM_ARGS="-Xmx4096 -Xms4096 -Dfile.encoding=UTF-8 -XX:+TieredCompilation"

Fire build now

```
$ make
```

Let's see output

- System image
  - $ ls out/target/product/rpi3
- Boot files
  - $ ls device/brcm/rpi3/boot
  - $ ls kernel/rpi/arch/arm/boot/
  - $ ls kernel/rpi/arch/arm/boot/dts/bcm2710-rpi-3-b.dtb
  - $ ls kernel/rpi/arch/arm/boot/dts/overlays/vc4-kms-v3d.dtbo
- Ramdisk image
  - $ ls out/target/product/rpi3/ramdisk.img

Prepare SD card

- Partitions of the card should be set-up like followings
  - p1 512MB for BOOT : Do fdisk : W95 FAT32(LBA) & Bootable, mkfs.vfat
  - p2 512MB for /system : Do fdisk, new primary partition
  - p3 512MB for /cache : Do fdisk, mkfs.ext4
  - p4 remainings for /data : Do fdisk, mkfs.ex4
- Set volume label for each partition - system, cache, userdata
- Use -L option of mkfs.ext4, e2label command, or -n option of mkfs.vfat

Prepare SD card
(check mount points)

- Check mount points for sdcard (Usually /dev/sdb<n>)

  $ mount

- Unmount the partitions

  $ sudo umount /dev/sdb1

  $ sudo umount /dev/sdb2

- Validate whether all partitions are unmounted

  $ mount

Prepare SD card
(Create partitions)

- Create new partitions

  $ sudo fdisk /dev/sdb

- List partions using **p** command
- Delete partions using **d** command
- Create partions using **n** command (Create all as primary (use **p**))
- Sizes (P1, P2, P3, P4): +512M, +3G, +512M, rest will be default
- Change partion type using **t** command (select **c** (FAT32 format) for p1 here)
- Define p1 as active using **a** command
- !!! Make sure to save the partion table using **w**

Prepare SD card
(Format partitions)

- Partion 1 is FAT32 format

  $ sudo mkfs.vfat -n boot /dev/sdb1

- All others are ext4

  $ sudo mkfs.ext4 -L system /dev/sdb2

  $ sudo mkfs.ext4 -L cache /dev/sdb3

  $ sudo mkfs.ext4 -L userdata /dev/sdb4

Copy images to SD card

- Copy system image

  $ cd out/target/product/rpi3

  $ sudo dd if=system.img of=/dev/<p2> bs=4M

- Copy Boot partition, kernel & ramdisk

  $ cp device/brcm/rpi3/boot/* to p1:/

  $ cp kernel/rpi/arch/arm/boot/zImage to p1:/

  $ cp kernel/rpi/arch/arm/boot/dts/bcm2710-rpi-3-b.dtb to p1:/

  $ cp kernel/rpi/arch/arm/boot/dts/overlays/vc4-kms-v3d.dtbo to p1:/overlays/vc4-kms-v3d.dtbo

  $ cp out/target/product/rpi3/ramdisk.img to p1:/

  - To preserve permissions

    $ sudo cp -a /path-to-working-system-part/* /media/emertxe/system/

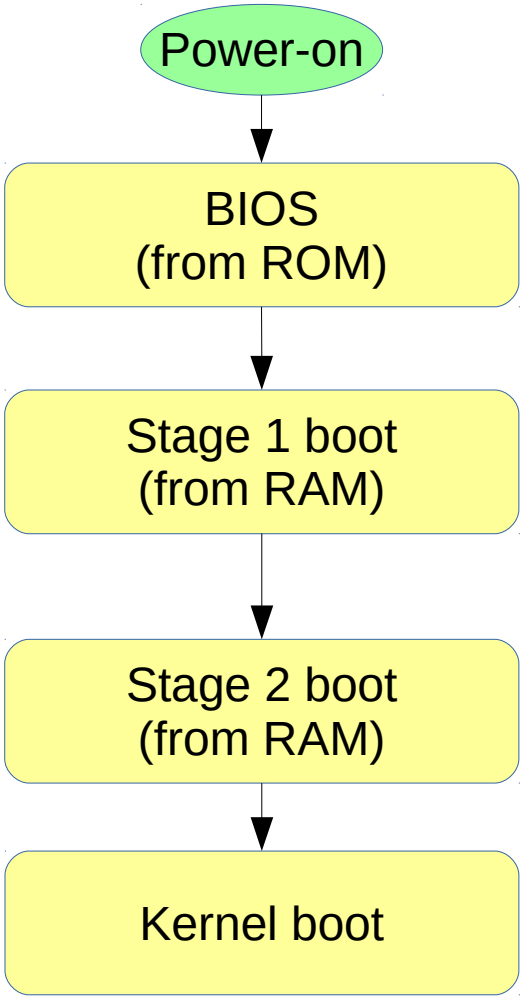
Bootloader

- Is a small piece of code executed before any operating system becomes ready to run
- Bootloader start running before any other software; therefore it is highly processor specific and board specific
- Bootloaders usually starts from ROM
- When a board is switched on CPU jumps to a hardcoded/predecided address which is the begining of bootloader instructions
- First task of bootloader (generally) to map RAM to predefined addresses; After RAM is mapped Stack Pointer (SP) is setup

Bootloader

- Bootloader is responsible for
  - Basic hardware initialization (RAM, Serial port, Screen, Keyboard etc..)
  - Loading kernel image (from flash, network or other storage) to RAM
  - Decomprerss kernel image
  - Execute kernel

Boot Sequence
(PC)



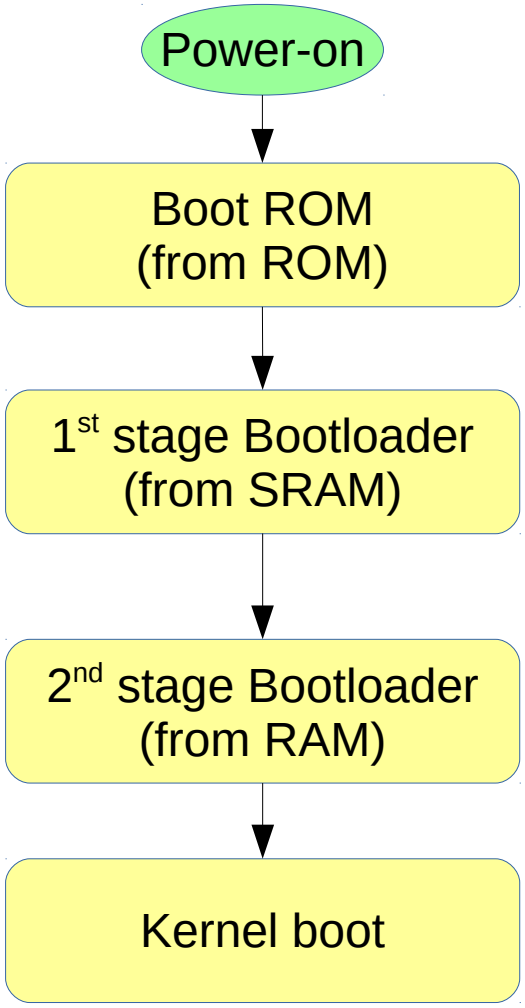
1. Basic Input/Output System
2. Runs POST stands for Power on Self Test
3. Initializes monitor and keyboard
4. Loads 1st stage bootloader from HDD/CD-ROM/Floppy in RAM

1. Example MBR (Master Boot Record)
2. Typically, stored in 1st sector of HDD or CD-ROM
3. Size is less than 512 bytes
4. Loads 2nd stage bootloader in RAM

1. Example GRUB (Grand Unified Bootlader)
2. Has knowledge of file system
3. Decompress & loads Kernel image & initrd in RAM

1. Mounts RFS
2. Executes /sbin/init program

Boot Sequence
(Embedded)



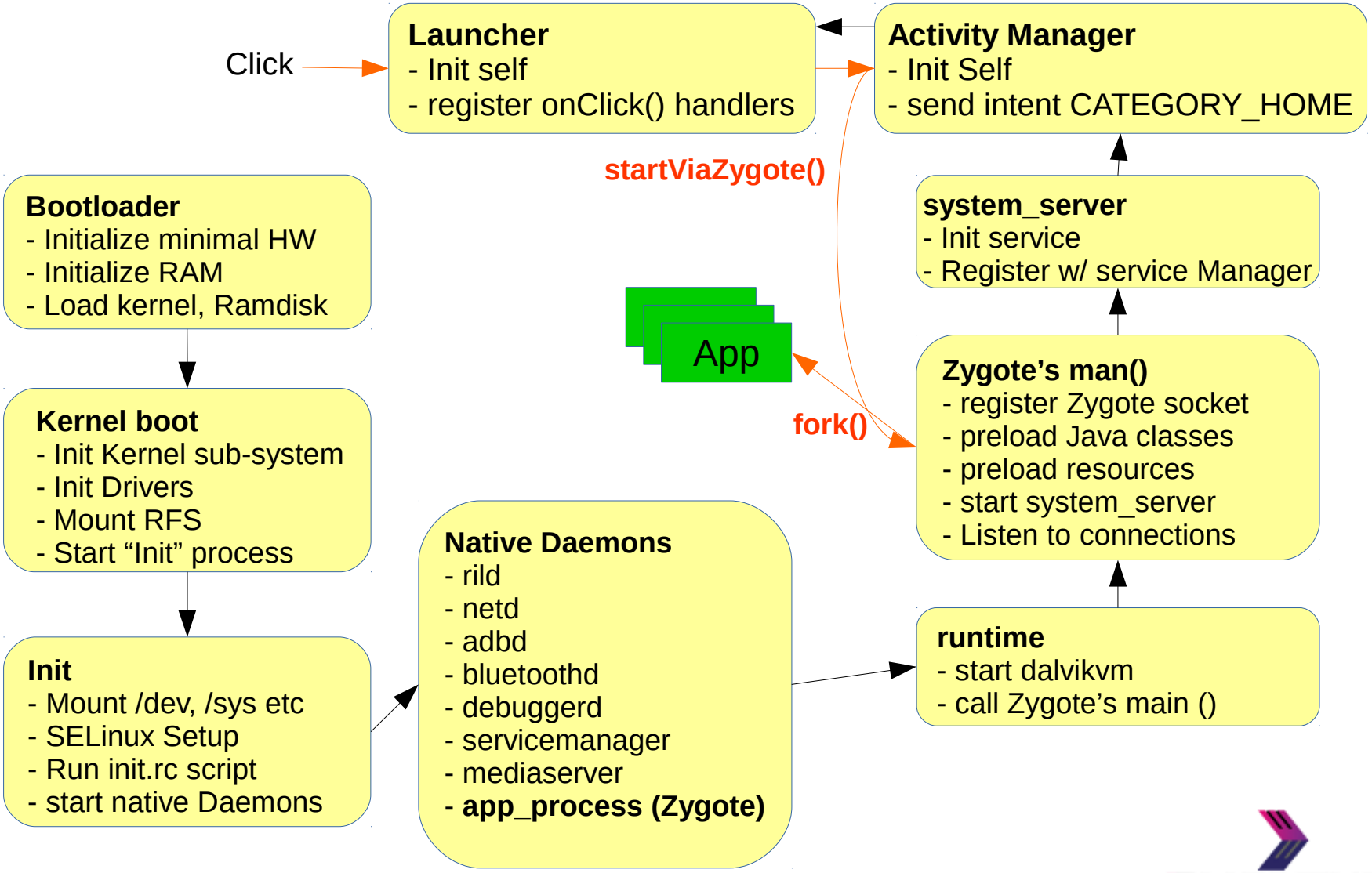
1. Board specific bootloader
2. Initialize minimal harware
3. Load first stage bootloader (MMC, SPI flash) in SRAM

1. Initialize DRAM, NAND, SPI controller
2. Load 2nd stage bootloader

1. Example U-boot or Barebox
Initialize other hardware (USB, NW)
2. Load Kernel image and Ramdisk

1. Mounts RFS
2. Executes /sbin/init program

Android Bootup

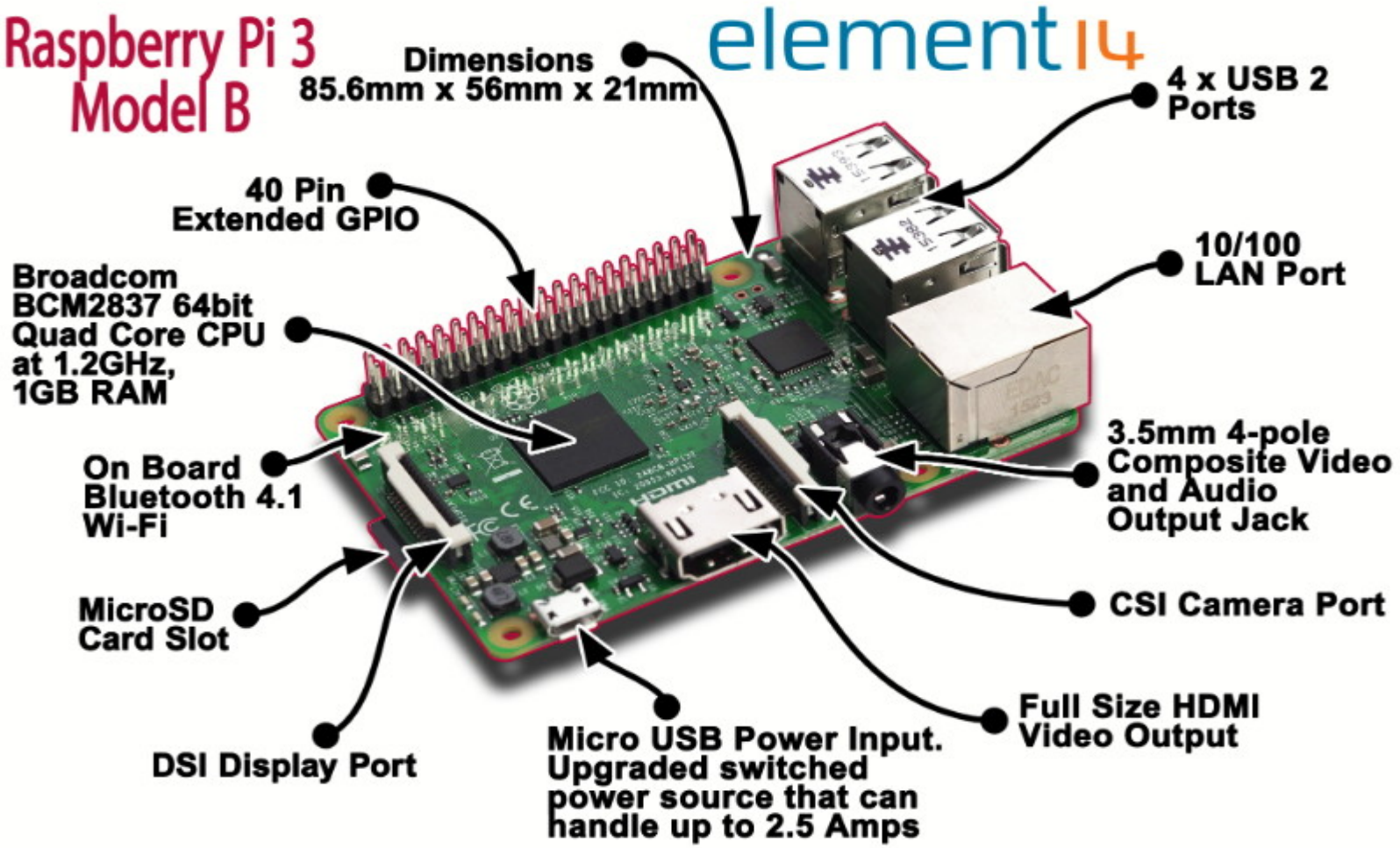


Hardware Details

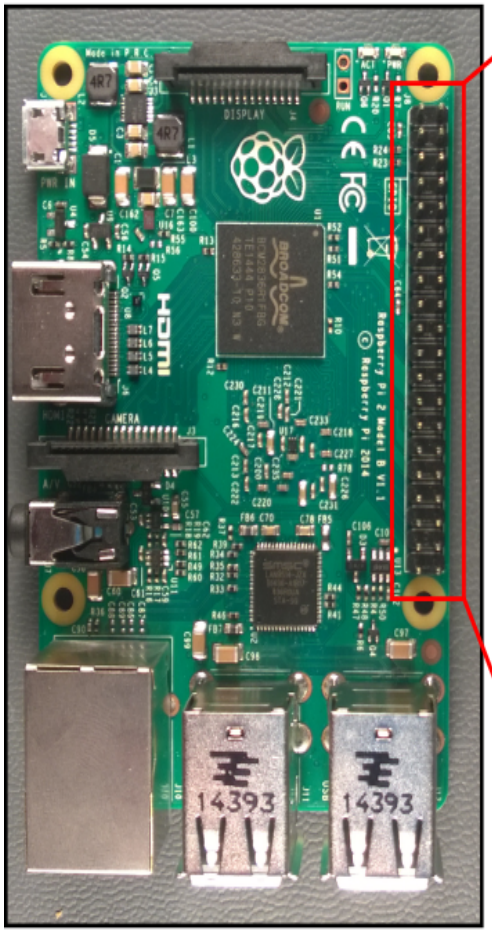
Raspberry Pi3
(specifications)

- Broadcom BCM2837 chipset running at 1.2 GHz
- 64-bit quad-core ARM Cortex-A53
- 802.11 b/g/n Wireless LAN
- Bluetooth 4.1 (Classic & Low Energy)
- Dual core Videocore IV® Multimedia co-processor
- 1 GB LPDDR2 memory
- microUSB connector for 2.5 A power supply
- 1 x 10/100 Ethernet port
- 1 x HDMI video/audio connector
- 1 x RCA video/audio connector
- 1 x CSI camera connector
- 4 x USB 2.0 ports
- 40 GPIO pins
- microSD card slot

Raspberry Pi3
(Hardware)



Raspberry Pi3
(Serial pin out)



Alternate Function						Alternate Function
	3.3V PWR	1	2	5V PWR		
I2C1 SDA	GPIO 2	3	4	5V PWR		
I2C1 SCL	GPIO 3	5	6	GND		
	GPIO 4	7	8	UART0 TX		
	GND	9	10	UART0 RX		
	GPIO 17	11	12	GPIO 18		
	GPIO 27	13	14	GND		
	GPIO 22	15	16	GPIO 23		
	3.3V PWR	17	18	GPIO 24		
SPI0 MOSI	GPIO 10	19	20	GND		
SPI0 MISO	GPIO 9	21	22	GPIO 25		
SPI0 SCLK	GPIO 11	23	24	GPIO 8		SPI0 CS0
	GND	25	26	GPIO 7		SPI0 CS1
	Reserved	27	28	Reserved		
	GPIO 5	29	30	GND		
	GPIO 6	31	32	GPIO 12		
	GPIO 13	33	34	GND		
SPI1 MISO	GPIO 19	35	36	GPIO 16		SPI1 CS0
	GPIO 26	37	38	GPIO 20		SPI1 MOSI
	GND	39	40	GPIO 21		SPI1 SCLK

EMERTXE

Debugging Essentials

Android Debug Bridge

- A command line tool that lets you communicate with an emulator or connected Android device
- ADB commands cane be categorized as follows -
  - General
  - Debug
  - Connection
  - Package Manager
  - File Manage
  - Network
  - Logcat
  - System

Android Debug Bridge
(General)

Command	Description
$ adb device	To list all the attached device instances
$ adb help	Prints a list of supported adb commands
$ adb version	Prints the adb version number

Android Debug Bridge
(Debug)

forward	
Syntax	$ adb forward <local> <remote>
Example	$ adb forward tcp:8000 tcp:9000
Description	set up forwarding of host port 8000 to emulator/device port 9000
Pre-rquisite	Enable USB debugging on the device
kill-server	
Syntax	$ adb kill-server
Example	-
Description	Terminates the adb server process
Pre-rquisite	-

Android Debug Bridge
(Connection)

Command	Description
$ adb connect <ip > [:port]	To connect to device specified with IP
$ adb usb	Restarting ADB in USB mode

```
Example:
   $ adb connect 192.168.32.199
```

Android Debug Bridge
(Package Manager)

install	
$ adb install [option] <path>	Installing a package
Options	
-l	Forward lock application
-r	Replace existing application
-t	Allow test packages
-s	Install application on sdcard
-d	Allow version code downgrade
-p	Partial application install

```
Example:
   $ adb install -t mytest.apk
```

Android Debug Bridge
(Package Manager)

uninstall	
$ adb install [option] <path>	Installing a package
Options	
-l	Forward lock application
-r	Replace existing application
-t	Allow test packages
-s	Install application on sdcard
-d	Allow version code downgrade
-p	Partial application install

```
Example:
   $ adb uninstall -t mytest.apk
```

EMERTXE

Android Debug Bridge
(Package Manager)

shell	
$ adb shell pm list packages	Installing a package
$ adb shell pm path <package>	Display path of package
$ adb shell pm clear <package>	Clearing app data, cache

```
Example:
   $ adb shell pm path com.android.phone
   $ adb shell pm clear com.android.phone
```

Android Debug Bridge
(File Manager)

shell	
$ adb pull <remote> [local]	Download a specified file from an emulator/device to your computer. By default pulls in platform-tools directory
$ adb push <local> [remote]	Upload a specified file from your computer to an emulator/device. By default pushes from platform-tools directory.

```
Example:
   $ adb pull /sdcard/demo.mp4 .
   $ adb push d:\test.apk /sdcard
```

Android Debug Bridge
(File Manager)

Command	Description
$ adb shell ls <path>	List files in a directory
$ adb shell cd <path>	Change directory
$ adb shell rm <file>	Remove a file
$ adb shell mkdir <dir>	Make directory
$ adb shell touch [Options] <file>	Create empty file or change file timestamps
$ adb shell pwd	Print current working directory location
$ adb shell cp <s-file> <d-file>	Copy source file to destination file
$ adb shell mv <s-file> <d-file>	Move source file to destination file

```
Example:
    $ adb ls
    $ adb mkdir test
    $ adb rm -r test
```

EMERTXE

Android Debug Bridge
(Network)

Command	Description
$ adb shell netstat	
$ adb shell ping <ip>	
$ adb shell netcfg	
$ adb shell ip	

```
Example:
    $ adb shell ping 192.168.32.71
```

Android Debug Bridge
(Logcat)

Command	Description
$ adb logcat [option] [filter-specs]	Prints logs to screen
*:V (Verbose) lowest priority, *:D (Debug), *:I (Info),	
*:W (warning), *:E (Error), *:F (Fatal), *:S (Silent) highest priority	
$ logcat -b <buffer : radio, event, main (default)>	
$ adb logcat -c : Clears the entire log and exits	
$ adb logcat -d : Dumps the log to the screen and exits	
$ adb logcat -f test.logs : Writes log message output to test.logs	
$ adb logcat -g : Prints the size of the specified log buffer and exits	
$ adb logcat -v [option]	

```
Example:
    $ adb logcat *:V
    $ logcat -b radio
    $ adb logcat -v brief
```

EMERTXE

Android Debug Bridge
(Screenshot)

Command	Description
$ adb shell screencap <file>	Taking a screenshot of a device display
$ adb shell screenrecord [OPT] <file>	Record video; press Ctrl-C to stop recording

```
Example:
    $ adb shell screencap /sdcard/screen.png
    $ adb shell screenrecord /sdcard/demo.mp4
    $ adb shell screenrecord --time-limit 180
    * 180 => 3 minute
```

Android Debug Bridge
(System)

Command	Description
$ adb root	Reboot the device
$ adb sideload	
$ adb reeboot	
$ adb shell ps	
$ adb shell top	
$ adb shell getprop [OPTION]	Get property via the android property service
$ adb shell setprop <key> <value>	Set property service
$ adb shell dumpsys [options]	Dumps system data.
$ adb shell dumpstate	Dumps state

```
Example:
    $ adb shell setprop service.adb.tcp.port 5555
    $ adb shell getprop ro.build.version.sdk
    $ adb dumpsys battery
    $ adb dumpstate > state_log.txt
```

THANK YOU