android Compatibility program

Compatibility Test Suite (CTS)

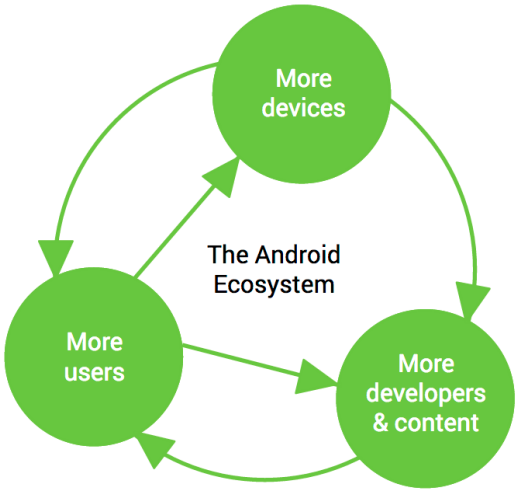
User Manual

Open Handset Alliance

Version 10

Contents

1. Why build compatible Android devices?



Users want a customizable device.

A mobile phone is a highly personal, always-on, always-present gateway to the Internet. We haven't met a user yet who didn't want to customize it by extending its functionality. That's why Android was designed as a robust platform for running after-market applications.

Developers outnumber us all.

No device manufacturer can hope to write all the software that a person could conceivably need. We need third-party developers to write the apps users want, so the Android Open Source Project aims to make it as easy and open as possible for developers to build apps.

Everyone needs a common ecosystem.

Every line of code developers write to work around a particular phone's bug is a line of code that didn't add a new feature. The more compatible phones there are, the more apps there will be. By building a fully compatible Android device, you benefit from the huge pool of apps written for Android, while increasing the incentive for developers to build more of those apps.

Android compatibility is free and easy.

See the Android Compatibility program introduction for more information.

2. Becoming compatible

2.1. Comply with Android Compatibility Definition Document (CDD)

To start, read the Android Compatibility overview, which describes the goals and components of the program.

Then review the Android Compatibility Definition Document (CDD) for the requirements of and policies associated with compatible devices.

The CDD's role is to codify and clarify specific requirements. The CDD does not attempt to be comprehensive. Since Android is a single corpus of open-source code, the code itself is the comprehensive "specification" of the platform and its APIs.

2.2. Pass the Compatibility Test Suite (CTS)

The Android Compatibility Test Suite (CTS) is a downloadable open-source testing harness you can use as you develop your Android device. For example, you could use the CTS to do continuous self-testing during your development work.

For more about the CTS and the compatibility report that it generates, see the Compatibility Test Suite introduction.

3. Running the automated CTS

3.1. Setting up your host machine

Before running CTS, make sure you have a recent version of Android Debug Bridge (adb)

installed and the 'adb' location added to the system path of your machine.

To install adb, download Android SDK tools and set up an existing IDE:
http://developer.android.com/sdk/index.html#ExistingIDE
http://developer.android.com/sdk/installing/index.html

Ensure 'adb' is in your system path:

```
export PATH=$PATH:$HOME/android-sdk-linux_x86/platform-tools
```

Download the CTS package(s) matching the Android version and all the ABIs (Application Binary Interface) your device support.

3.2. Storage requirements

The CTS media stress tests require video clips to be on external storage (`/sdcard`). Most of the clips are from Big Buck Bunny which is copyrighted by the Blender Foundation under the Creative Commons Attribution 3.0 license.

The required space depends on the maximum video playback resolution supported by the device (See section 5 in the compatibility definition document for the platform version of the required resolutions.) Note that the video playback capabilities of the device under test will be checked via the `android.media.CamcorderProfile` APIs for earlier versions of Android and the `android.media.MediaCodecInfo.CodecCapabilities` APIs from Android 5.0

Here are the storage requirements by maximum video playback resolution:

- 480x360: 98MB
- 720x480: 193MB
- 1280x720: 606MB
- 1920x1080: 1863MB

3.3. Setting up your device

CTS should be executed on consumer (user build) devices only.

Follow these instructions to prevent test timeouts and other failures:

1. Your device should be running a **user build (Android 4.0 and later)** from source.android.com.
2. Set up your device per the Using Hardware Devices instructions on the Android developer site.

   Note: Any device that does not have an embedded screen need to be connected to a screen.
3. Flash your device with a user build before you run CTS.
4. If the device has a memory card slot, plug in an empty SD card. Use an SD card that supports Ultra High Speed (UHS) Bus with SDHC or SDXC capacity or one with at least speed class 10 or higher to ensure it can pass the CTS.

   Warning: CTS may modify/erase data on the SD card plugged in to the device.
5. Factory data reset the device (Settings > Storage > Factory data reset).

   Warning: This will erase all user data from the device.
6. Set your device's language to English (United States) from Settings > Language & input > Language.
7. Turn on the Location setting if there is a GPS or Wi-Fi / Cellular network available.
8. Connect to a Wi-Fi network (Settings > Wi-Fi) that supports IPv6 and has an internet connection.
   Tip: If you don't have access to a native IPv6 network, an IPv6 carrier network, or a VPN to pass some tests depending on IPv6, you may instead use a Wi-Fi access point and an IPv6 tunnel. See Wikipedia list of IPv6 tunnel brokers.
9. Make sure no lock pattern is set on the device (Settings > Security > Screen Lock = 'None').
10. Check the "USB Debugging" development option (Settings > Developer options > USB debugging).
11. Connect the host machine that will be used to test the device and "Allow USB debugging" for the computer's RSA key fingerprint.
12. Check Settings > Developer options > Stay Awake.
13. Check Settings > Developer options > Allow mock locations.
14. For CTS versions 2.1 R2 through 4.2 R4, set up your device (or emulator) to run the accessibility tests:
    ```
    adb install -r
    android-cts/repository/testcases/CtsDelegatingAccessibilityS
    ervice.apk
    ```
    On the device, enable Settings > Accessibility > Accessibility > Delegating Accessibility Service

15. For CTS 2.3 R4 and beyond on devices that declare the android.software.device_admin feature, set up your device to run the device administration tests as below:
    `adb install -r android-cts/repository/testcases/CtsDeviceAdmin.apk`
    On the device, enable only the two `android.deviceadmin.cts.CtsDeviceAdminReceiver*` device administrators under Settings > Security > Select device administrators. Make sure the `android.deviceadmin.cts.CtsDeviceAdminDeactivatedReceiver` and any other preloaded device administrators stay disabled in the same menu.
16. For CTS 2.3 R12 and beyond, if the device supports video codecs, the CTS media files must be copied to the device. (See section 3.4 for details.)
17. Launch the browser and dismiss any startup/setup screen.
18. Press the home button to set the device to the home screen at the start of CTS.
19. While a device is running tests, it must not be used for any other tasks and must be kept in a stationary position (to avoid triggering sensor activity) with the cameras pointing an object that could be focused.
20. Do not press any keys on the device while CTS is running. Pressing keys or touching the screen of a test device will interfere with the running tests and may lead to test failures.

3.4. Copying media files to the device

Follow these instructions to copy the media files to a device:

1. Download the android-cts-media-X.Y.zip file from http://source.android.com/compatibility/downloads.html and unzip it.
2. Connect the device to the computer and check that adb can connect to it.
3. Navigate (cd) to the unzipped folder.
4. Change the file permissions:
   `chmod u+x copy_media.sh`
5. Run `copy_media.sh`:
   - To copy clips up to a resolution of 720x480, run:
     `./copy_media.sh 720x480`
   - If you are not sure about the maximum resolution, try `all` so that all files are copied.
   - If there are multiple devices under adb, add the -s (serial) option to the end. For example, to copy up to 720x480 to the device with serial 1234567, run:
     `./copy_media.sh 720x480 -s 1234567`

3.5. Using the CTS tradefed

To run a test plan:

1. Connect at least one device.
2. Launch the CTS console by running the *cts-tradefed* script from the folder where the CTS package has been unzipped, e.g.
   `$ ./android-cts/tools/cts-tradefed`
3. You may start the default test plan (containing all of the test packages) by appending:
   `run cts --plan CTS`
   This will kick off all the CTS tests required for compatibility.
   Enter `list plans` to see a list of test plans in the repository.
   Enter `list packages` to see a list of test packages in the repository.
   See the CTS command reference or type help for a complete list of supported commands.
4. Alternately, you may run the CTS plan of your choosing from the command line using:
   `cts-tradefed run cts --plan <plan_name>`
5. View test progress and results reported on the console.
6. If your device is Android 5.0 or later and declares support for an ARM and a x86 ABI, you should run both the ARM and x86 CTS packages.

3.6. Selecting CTS plans

The following test plans are available:

1. *CTS*—all tests required for compatibility.
2. *Signature*—the signature verification of all public APIs
3. *Android*—tests for the Android APIs
4. *Java*—tests for the Java core library
5. *VM*—tests for ART or Dalvik
6. *Performance*—performance tests for your implementation

These can be executed with the `run cts` command as mentioned earlier.

4. Interpreting the test results

The test results are placed in the file:
`$CTS_ROOT/android-cts/repository/results/<start time>.zip`

$CTS_ROOT will resemble the path “out/host/linux-x86/cts” but will differ by platform.

Inside the zip, the testResult.xml file contains the actual results—open this file in any web browser (HTML5 compatible browser recommended) to view the test results. It will resemble the following screenshots.

Show Device Information

Test Summary	
CTS version	4.4_r3
Test timeout	600000 ms
Host Info	
Plan name	CTS
Start time	Mon Sep 15 18:14:05 PDT 2014
End time	Tue Sep 16 06:39:38 PDT 2014
Tests Passed	28446
Tests Failed	17
Tests Timed out	0
Tests Not Executed	0

Test Summary by Package

Test Package	Passed	Failed	Timed Out	Not Executed	Total Tests
PairingSetup	0	1	0	0	1
android.aadb	11	0	0	0	11
android.acceleration	6	0	0	0	6
android.accessibility	27	0	0	0	27
android.accessibilityservice	54	0	0	0	54
android.accounts	28	0	0	0	28
android.admin	18	0	0	0	18
android.animation	79	0	0	0	79
android.app	294	0	0	0	294
android.bionic	540	0	0	0	540
android.bluetooth	9	0	0	0	9

The Device Information section provides details about the device, firmware (make, model, firmware build, platform), and device hardware(screen resolution, keypad, screen type). To access device information, click the link above Test Summary.

The Test Summary' section provides executed test plan details, like the CTS plan name and execution start and end times. It also presents an aggregate summary of the number of tests that passed, failed, timed out, or could not be executed.

The next section also provides a summary of tests passed per package.

Detailed Test Report

Compatibility Test Package: PairingSetup

Test	Result	Details
com.google.android.clockwork.PairingStress		
-- iteration0	fail	Companion: false, Clockwork: null

Compatibility Test Package: android.aadb

Test	Result	Details
com.android.cts.aadb.TestDeviceFuncTest		
-- testBugreport	pass	
-- testExecuteShellCommand	pass	
-- testGetLogcat_size	pass	
-- testGetScreenshot	pass	
-- testPull_noexist	pass	
-- testPushDir	pass	
-- testPushPull_extStorageVariable	pass	
-- testPushPull_normal	pass	
-- testSyncFiles_extStorageVariable	pass	
-- testSyncFiles_normal	pass	
com.android.cts.aadb.TestDeviceStressTest		
-- testPushFolderWithManyFiles	pass	

Compatibility Test Package: android.acceleration

Test	Result	Details
android.acceleration.cts.HardwareAccelerationTest		
-- testIsHardwareAccelerated	pass	
-- testNotAttachedView	pass	
android.acceleration.cts.SoftwareAccelerationTest		
-- testIsHardwareAccelerated	pass	
-- testNotAttachedView	pass	
android.acceleration.cts.WindowFlagHardwareAccelerationTest		
-- testIsHardwareAccelerated	pass	
-- testNotAttachedView	pass	

This is followed by details of the the actual tests that were executed. The report lists the test package, test suite, test case, and the executed tests. It shows the result of the test execution—pass, fail, timed out, or not executed. In the event of a test failure details are provided to help diagnose the cause.

Further, the stack trace of the failure is available in the XML file but is not included in the report to ensure brevity—viewing the XML file with a text editor should provide details of the test failure (search for the *<Test>* tag corresponding to the failed test and look within it for the *<StackTrace>* tag).

5. CTS Verifier instructions

The Android Compatibility Test Suite Verifier (CTS Verifier) is a supplement to the Compatibility Test Suite (CTS). While CTS checks those APIs and functions that can be automated, CTS Verifier provides tests for those APIs and functions that cannot be tested on a stationary device without manual input, like audio quality, touchscreen, accelerometer, camera, etc.

5.1. Test Preparation

The device must have verified Android API compatibility by successfully passing the Compatibility Test Suite.

5.1.1. Hardware requirements

- A Linux computer with USB 2.0 compatible port
- A second Android device with a known compatible Bluetooth, Wi-Fi direct, and NFC Host Card Emulation (HCE)\ implementation

5.1.2. Setup

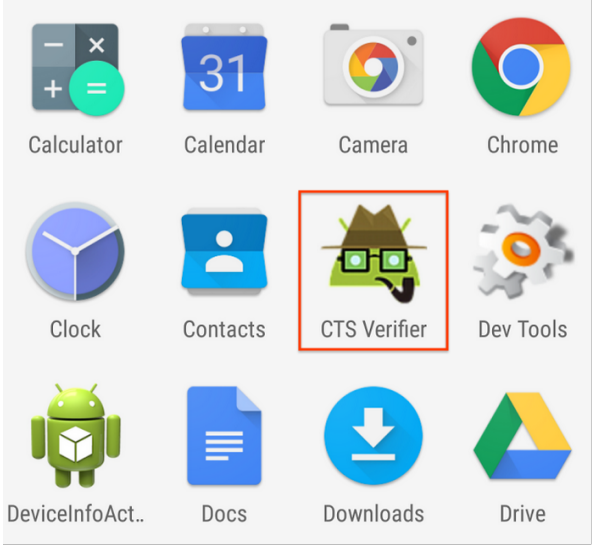
- Install the Android SDK on the Linux computer
- Download the appropriate CTS Verifier.apk for the version of Android under test
- Install CTS Verifier.apk to the *Device Under Test* (DUT).

```
adb install -r CtsVerifier.apk
```

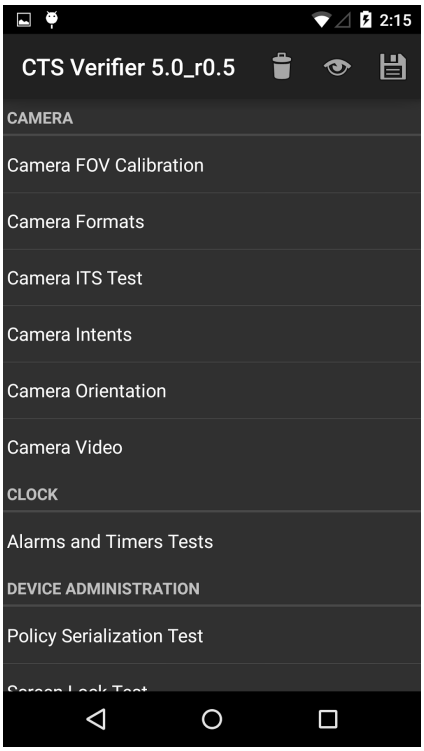
- Ensure that the device has its system data and time set correctly.

5.2. CTS test procedure

- After the CTS Verifier.apk has been installed, launch the CTS Verifier application:

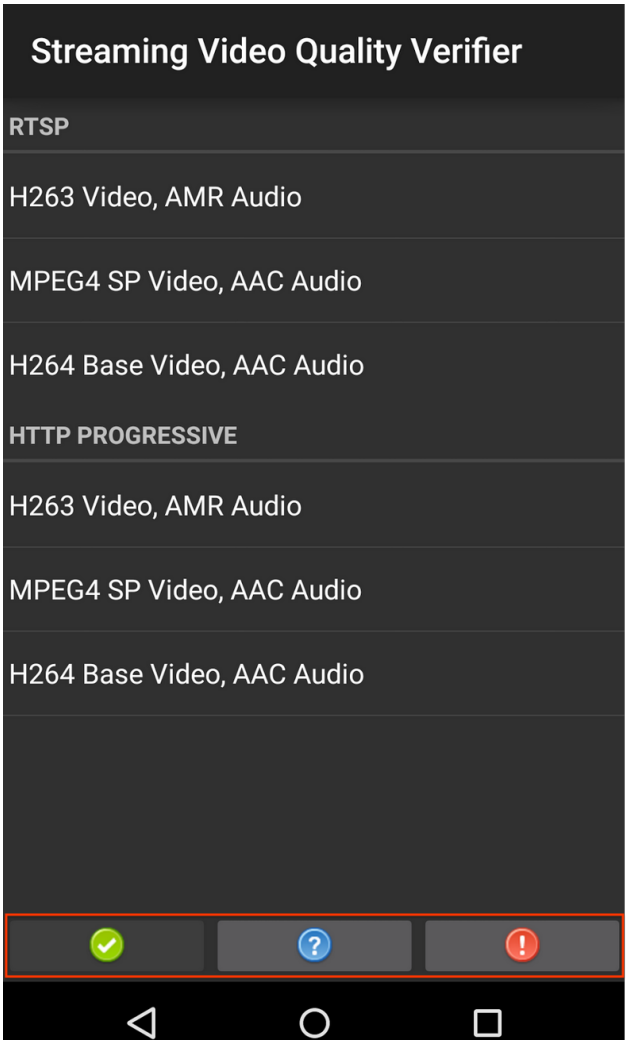


- Once opened, the CTS Verifier displays a list of all test sets available for manual verification:



- Each test contains a set of common elements (in some tests, Pass/Fail is determined automatically):

- *Info*—a set of instructions to run the test. This will appear as a popup the first time each test is opened or whenever the **Info** button (?) is pressed.
- *Pass*—If the DUT meets the test requirements per the instructions from Info, press the **Pass** button (✓).
- *Fail*—If the DUT does not meet the test requirements per the instructions from Info, press the **Fail** button (!).



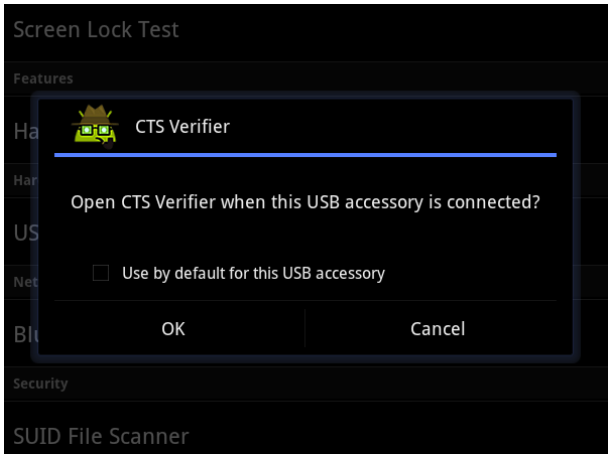
5.3. Specific test requirements

5.3.1. USB Accessory

In order to run the USB Accessory test, you need a Linux computer to run the USB host

program.

1. Connect the DUT to a computer.
2. Execute the cts-usb-accessory program on the computer found in the CTS Verifier package.
3. A popup message will appear on the DUT. Select **OK** and go into the USB Accessory Test in the CTS Verifier application.



4. Output similar to below will appear on the computer's console.

```
out/host/linux-x86/cts-verifier/android-cts-verifier$
./cts-usb-accessory
CTS USB Accessory Tester
Found possible Android device (413c:2106) - attempting to switch to
accessory mode...
Failed to read protocol version
Found Android device in accessory mode (18d1:2d01)...
[RECV] Message from Android device #0
[SENT] Message from Android accessory #0
[RECV] Message from Android device #1
[SENT] Message from Android accessory #1
[RECV] Message from Android device #2
[SENT] Message from Android accessory #2
[RECV] Message from Android device #3
[SENT] Message from Android accessory #3
[RECV] Message from Android device #4
[SENT] Message from Android accessory #4
[RECV] Message from Android device #5
[SENT] Message from Android accessory #5
[RECV] Message from Android device #6
[SENT] Message from Android accessory #6
[RECV] Message from Android device #7
[SENT] Message from Android accessory #7
[RECV] Message from Android device #8
[SENT] Message from Android accessory #8
[RECV] Message from Android device #9
[SENT] Message from Android accessory #9
[RECV] Message from Android device #10
[SENT] Message from Android accessory #10
```

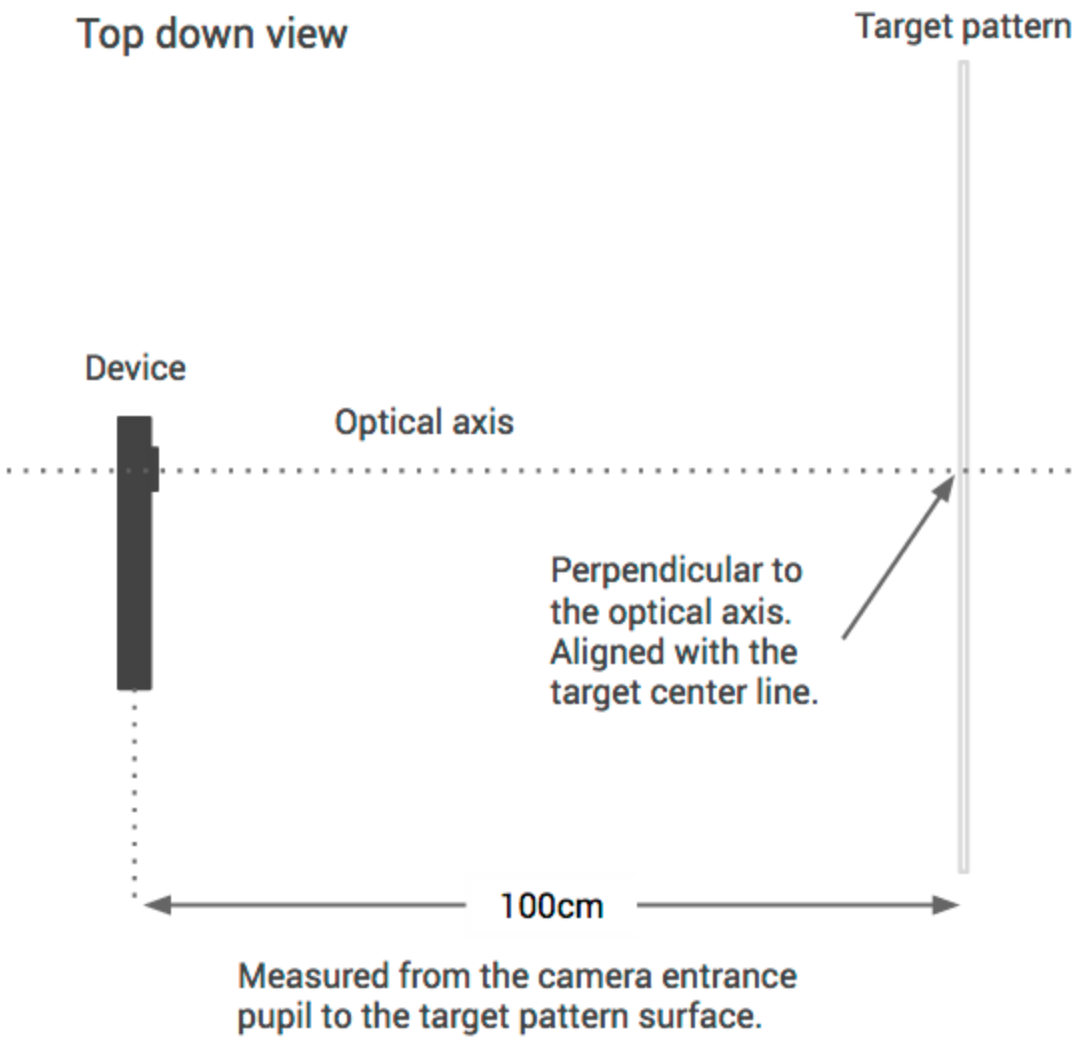
5.3.2. Camera field of view calibration

This field of view calibration procedure is designed to be a quick way to determine the device field of view with moderate accuracy.

Setup

Print the calibration-pattern.pdf target file and mount it on a rigid backing. Print on 11” x 17” or A3.

Orient the camera device and the printed target as shown in the diagram below:



Setting the target width

Measure the distance between the solid lines on the target pattern in centimeters to account for printing inaccuracies (~38 cm).

1. Start the calibration application.
2. Press the setup button and select “Marker distance” to enter the distance.
3. Measure and enter the distance to the target pattern (~100 cm).
4. Press the back button to return to the calibration preview.

Calibration process

Verify that the device and target are placed as shown in the figure and the correct distances have been entered into the setup dialog.

The preview will display the image with a vertical line overlaid onto it. This line should

align with the center line of the target pattern. The transparent grid can be used with the other vertical lines to ensure that the optical axis is orthogonal to the target.

- Select an image resolution to test from the selector at the bottom left.
- Tap the screen to take a photo and enter the calibration mode (described below).
- Hit the back button and repeat for all supported image resolutions.

Calibration test (per resolution)

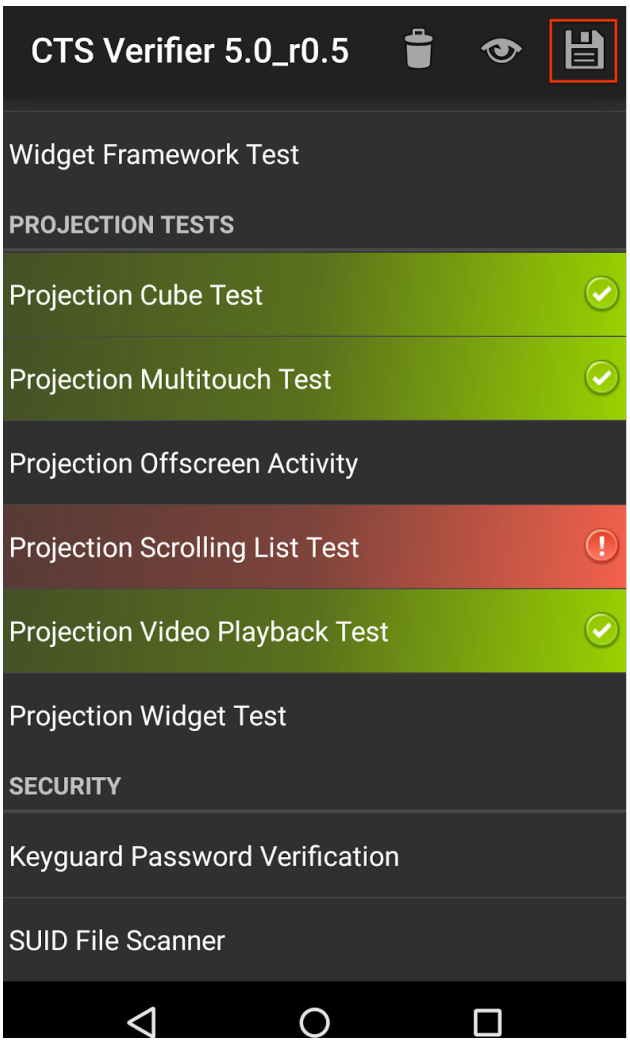
In the calibration mode, the photo will be displayed with two vertical lines overlaid onto the image.

These lines should align with the vertical lines on the target pattern within a few pixels. If they do not, then the reported field of view for that mode is inaccurate (assuming the setup is correct).

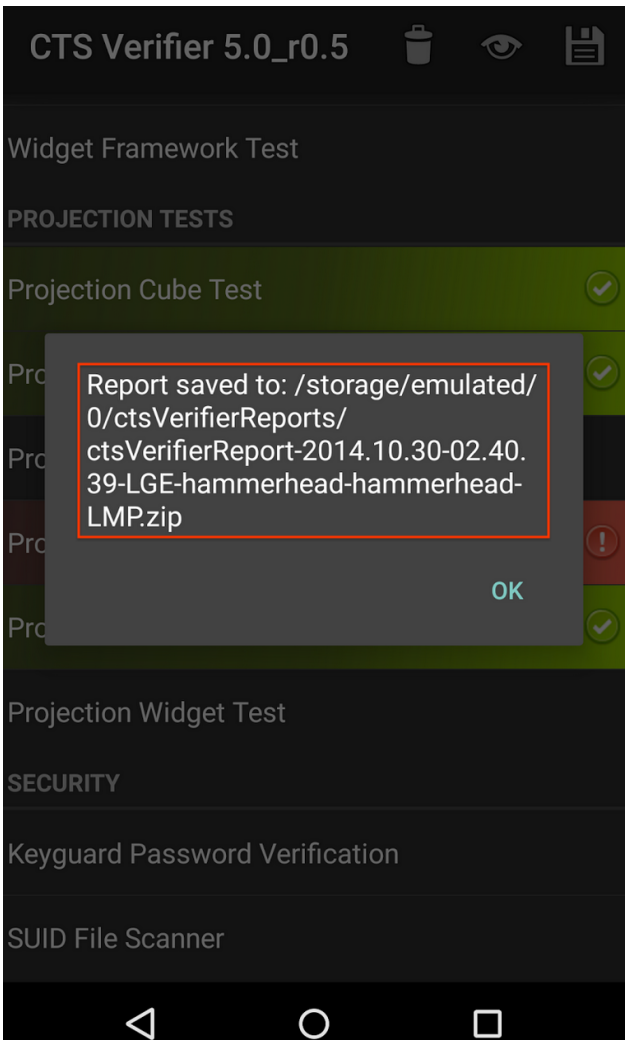
Adjust the slider at the bottom of the screen until the overlay aligns with the target pattern as closely as possible. The displayed field of view will be a close approximation to the correct value when the overlay and the target pattern image are aligned. The reported field of view should be within +/-1 degree of the calibration value.

5.4. Exporting test reports

- After all tests are completed, tap the Save icon  .



- A path to the saved report will be displayed in pop-up (e.g. `/mnt/sdcard/ctsVerifierReports/ctsVerifierReport-date-time.zip`). Record the path.



- Connect the device via USB to a computer with the SDK installed.
- From the computer's SDK installation, run `adb pull <CTS Verifier report path>` to download the report from the device.
  - To download all reports run :
    `adb pull /mnt/sdcard/ctsVerifierReports/ .`
  - The name of the reports are time-stamped based on the DUT's system time.
- To clear results after they have been selected, select Menu > Clear. This will clear the Pass/Fail results.

6. Release notes

- Note the CTS test harness has changed significantly in the Android 4.0 release. Some new features have been added included support for sharding a CTS test run onto multiple concurrent devices, as well as general faster performance.
- This CTS release contains approximately 18,000 tests that you can execute on the device.
- Please make sure all steps in section 3.3 "Setting up your device" have been followed before you kick off CTS. Not following these instructions may cause tests to timeout or fail.

7. Appendix: CTS Console command reference

Host

help	Display a summary of the most commonly used commands.
help all	display the complete list of available commands
exit	Gracefully exit the CTS console. Console will close when all currently running tests are finished

Run

run cts	Run the specified tests and displays progress information. One of --plan, --package, --class or --continue-session-id needs to be specified. The CTS console can accept other

	commands while tests are in progress. If no devices are connected, the CTS host will wait for a device to be connected before starting tests. If more than one device is connected, CTS host will choose a device automatically.
--plan <test_plan_name>	Run the specified test plan
-- package/-p <test_package_name> [--package/-p <test_package2>...]	Runs the specified test packages.
--class/-c <class_name> [--method/-m <test_method_name>	Runs the specified test class and/or method
--continue-session-id	Runs all not executed tests from previous CTS session. The sessions testResult.xml will be updated with the new results.
--shards <number_of_shards>	Shard a CTS run into given number of independent chunks, to run on multiple devices in parallel.
--serial/-s <deviceID>	Run CTS on the specific device
-t <class_name>#<test_method_name>	Run a specific test method
--abi 32|64	On 64-bit devices, run the test against only the 32-bit or 64-bit ABI
List	
list packages	List all available test packages in the repository.

list plans	Lists all available test plans in the repository
list invocations	Lists 'run' commands currently being executed on devices.
list commands	List all 'run' commands currently in the queue waiting to be assigned to devices
list results	List CTS results currently stored in repository
list devices	List currently connected devices and their state. 'Available' devices are functioning, idle devices, available for running tests. 'Unavailable' devices are devices visible via adb, but are not responding to adb commands and won't be allocated for tests. 'Allocated' devices are devices currently running tests.

Add

add derivedplan --plan <plan_name> --result/-r [pass | fail | timeout | notExecuted] [--session/-s <session_id>]	Create a plan derived from given result session.