

# Compatibility Test Suite (CTS)

*User Manual*

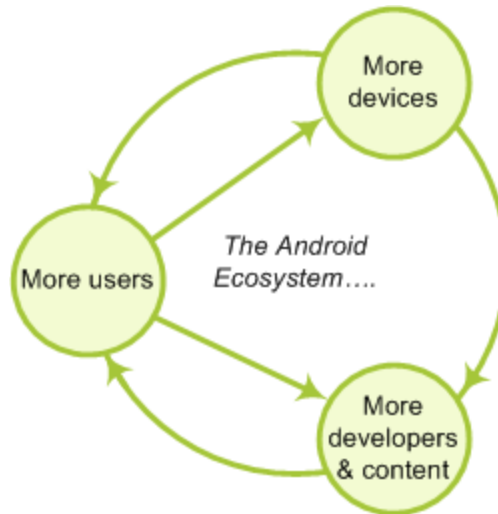
Open Handset Alliance

revision 7

## Contents

1. Why build compatible Android devices?
2. How can I become compatible?
  - 2.1. Comply with Android Compatibility Definition document
  - 2.2. Pass the Compatibility Test Suite (CTS)
3. Running the automated CTS
  - 3.1. Setting up your host machine
  - 3.2. Storage requirements
  - 3.3. Setting up your device
  - 3.4. Copying media files to the device
  - 3.5. Using the CTS
  - 3.6. Selecting CTS plans
4. Interpreting the test results
5. CTS Verifier instructions
  - 5.1. Test Preparation
    - 5.1.1. Hardware requirements
    - 5.1.2. Setup
  - 5.2. CTS test procedure
  - 5.3. Specific test requirements
    - 5.3.1. USB Accessory
    - 5.3.2. Camera field of view calibration
  - 5.4. Exporting test reports
6. Release notes
7. Appendix: CTS Console command reference

## 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 it's easy.**

See the Android Compatibility program introduction for more information:

<http://source.android.com/compatibility/index.html>

## 2. How can I become compatible?

### 2.1. Comply with Android Compatibility Definition document

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

<http://source.android.com/compatibility/overview.html>

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

<http://source.android.com/compatibility/android-cdd.pdf>

The CDD's role is to codify and clarify specific requirements, and eliminate ambiguity. 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 handset; 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:

<http://source.android.com/compatibility/cts-intro.html>

For the latest instructions on using the CTS, visit the following link:

<http://source.android.com/compatibility/android-cts-manual.pdf>

## 3. Running the automated CTS

### 3.1. Setting up your host machine

**Note:** the steps to configure and run CTS have changed in the 4.0 release.

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/myuser/android-sdk-linux_x86/platform-too
ls
```

---

### 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 ( <http://www.bigbuckbunny.org>) under the Creative Commons Attribution 3.0 license: <http://creativecommons.org/licenses/by/3.0/>

The required space depends on the maximum video playback resolution supported by the device. By default, 176x144 and 480x360 SHOULD be supported. Note that the video playback capabilities of the device under test will be checked via the android.media.CamcorderProfile APIs.

Here are the storage requirements by maximum video playback resolution:

- 480x360: 91.4MB
- 720x480: 151.9MB
- 1280x720: 401.6MB
- 1920x1080: 1008.2MB

### 3.3. Setting up your device

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

This section is important as not following these instructions will lead to test timeouts and other failures:

1. Your device should be running a **user build (Android 4.0 and later)** from [source.android.com](http://source.android.com).
2. Set up your device per the Using Hardware Devices instructions on the Android developer site:  
<http://developer.android.com/tools/device.html>
3. Make sure your device has been flashed with a user build before you run CTS.
4. If the device has a memory card slot, make sure the device has an SD card plugged in and the card is empty. *Warning: CTS may modify/erase data on the SD card plugged in to the device.*
5. Do a factory data reset on the device (Settings > Storage > Factory data reset).  
*Warning: This will erase all user data from the device.*
6. Make sure your device is set up with English (United States) as the language (Settings > Language & input > Language).
7. Make sure the device is connected to a functioning Wi-Fi network (Settings > Wi-Fi).
8. Make sure no lock pattern is set on the device (Settings > Security > Screen Lock = 'None').
9. Make sure the "USB Debugging" development option is checked (Settings > Developer options > USB debugging).
10. Connect the host machine that will be used to test the device, and "Allow USB debugging" for the computer's RSA key fingerprint.
11. Make sure Settings > Developer options > Stay Awake is checked.
12. Make sure Settings > Developer options > Allow mock locations is checked.
13. Make sure the device is at the home screen at the start of CTS (by pressing the home button).
14. Set up your device (or emulator) to run the accessibility tests per the Workflow section of the CTS introduction:  
<http://source.android.com/compatibility/cts-intro.html#workflow>
15. 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).
16. 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. Use 'chmod' to change the file permissions like so:  
`chmod 544 copy_media.sh`
5. Run `copy_media.sh` like so:
  - To copy clips for just the default resolutions, 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 1920x1080 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

To run a test plan:

- Make sure you have at least one device connected. 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`
- You may start the default test plan (containing all of the test packages) by typing `run cts --plan CTS`. This will kick off all the CTS tests required for compatibility. Type `list plans` to see a list of test plans in the repository. Type `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.
- Alternately, you can just run a CTS plan from the command line using `cts-tradefed run cts --plan <plan_name>`
- You should see test progress and results reported on the console.

### 3.6. Selecting CTS plans

For this release the following test plans are available:

1. *CTS* - all tests and will run ~18,000 tests on your device. These tests are required for compatibility. At this point performance tests are not part of this plan (this will change for future CTS releases).
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 the Dalvik virtual machine
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
```

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.



android  
compatibility program **Test Report for dream - HT851LZ01986**

Device Information		Test Summary	
Device Make	dream	Plan name	CTS
Build model	HT851LZ01986	Start time	Wed Feb 11 15:20:53 PST 2009
Firmware Version	1.5	End time	Wed Feb 11 15:49:49 PST 2009
Firmware Build Number	CUPCAKE	Version	1.0
Android Platform Version	3	Tests Passed	1448
Supported Locales	en_US;es;en_US;zz_ZZ;en;	Tests Failed	40
Screen size	320x480	Tests Timed out	1
Phone number	null	Tests Not Executed	0
x dpi	180.62193		
y dpi	181.96814		
Touch	finger		
Navigation	trackball		
Keypad	qwerty		
Network			
IMEI	351676030149928		
IMSI	null		

**Test Summary by Package**

Test Package	Tests Passed
android.tests.sigtest	1 / 1
android.app	32 / 35
android.content	163 / 167
android.database	18 / 18
android.graphics	489 / 499
android.location	19 / 19
android.net	29 / 30
android.os	75 / 75
android.provider	10 / 10
android.text	147 / 150
android.util	32 / 32

The '*device information*' section provides details about the device and the firmware (make, model, firmware build, platform) and the hardware on the device (screen resolution, keypad, screen type).

The details of the executed test plan are present in the '*test summary*' section which provides the CTS plan name and execution start and end times. It also presents an aggregate summary of the number of tests that passed, failed, time out or could not be executed.

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

Compatibility Test Package: android.widget

Test	Result	Failure Details
Suite: android.widget.cts.		
- AbsSeekBarTest		
-- testConstructor	pass	
-- testAccessThumbOffset	pass	
-- testSetThumb	pass	
-- testOnTouchEvent	pass	
-- testDrawableStateChanged	pass	
-- testOnDraw	pass	
-- testOnMeasure	pass	
-- testVerifyDrawable	fail	junit.framework.AssertionFailedError at android.widget.cts.AbsSeekBarTest.testVerifyDrawable(AbsSeekBarTest.java:327)
-- testOnSizeChanged	pass	
-- testAndroidTestCaseSetupProperly	pass	
- ButtonTest		
-- testConstructor	pass	
-- testAndroidTestCaseSetupProperly	pass	
- ChronometerTest		
-- testConstructor	pass	
-- testAccessBase	pass	
-- testAccessFormat	pass	
-- testOnDetachedFromWindow	pass	
-- testOnWindowVisibilityChanged	pass	
-- testStartAndStop	pass	
- CompoundButtonTest		
-- testConstructor	pass	
-- testAccessChecked	pass	
-- testSetOnCheckedChangeListener	pass	
-- testToggle	pass	
-- testPerformClick	pass	
-- testDrawableStateChanged	pass	
-- testSetButtonDrawableByDrawable	pass	
-- testSetButtonDrawableById	pass	
-- testOnCreateDrawableState	pass	
-- testOnDraw	pass	
-- testAccessInstantState	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 (e.g. 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:  
<http://developer.android.com/sdk/index.html>
- Download the appropriate CTS Verifier.apk for the version of Android under test:  
<http://source.android.com/compatibility/downloads.html>
- Install CTS Verifier.apk to the *Device Under Test* (DUT).

```
adb install -r CTS Verifier.apk
```

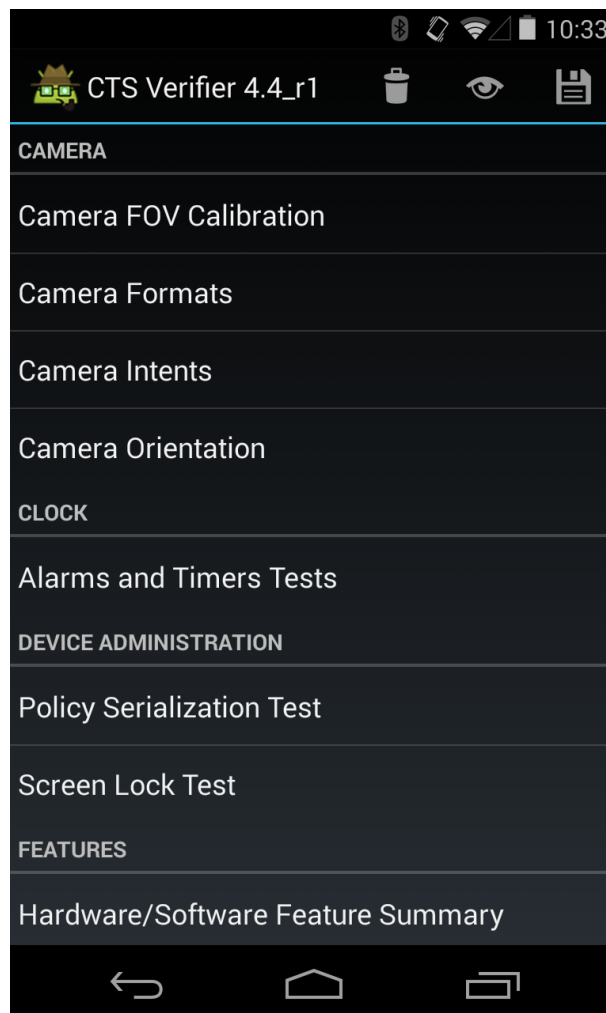
- Ensure that the device has its system data and time set correctly.

## 5.2. CTS test procedure

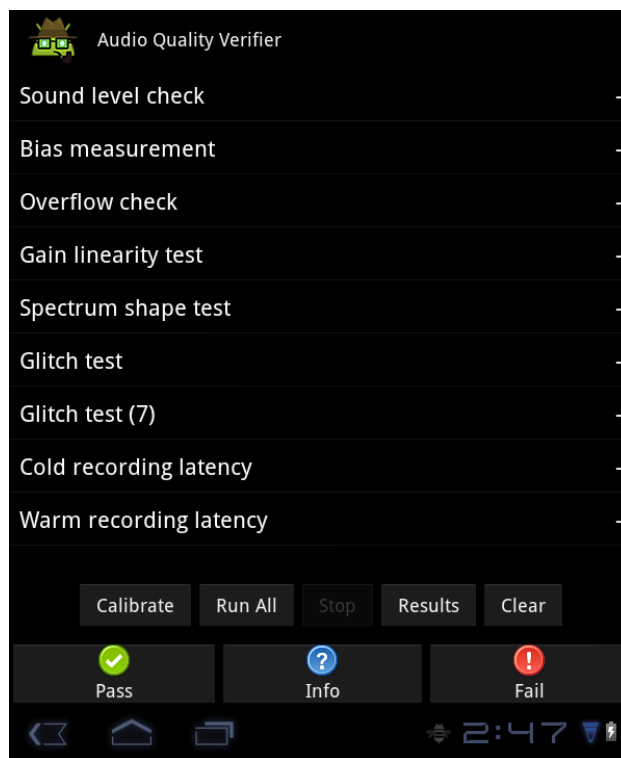
- After the CTS Verifier.apk has been installed, launch the CTS Verifier application:



- 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', then select the Pass button.
  - Fail - If the DUT does not meet the test requirements per the instructions from 'Info', then select the Fail button.



## 5.3. Specific test requirements

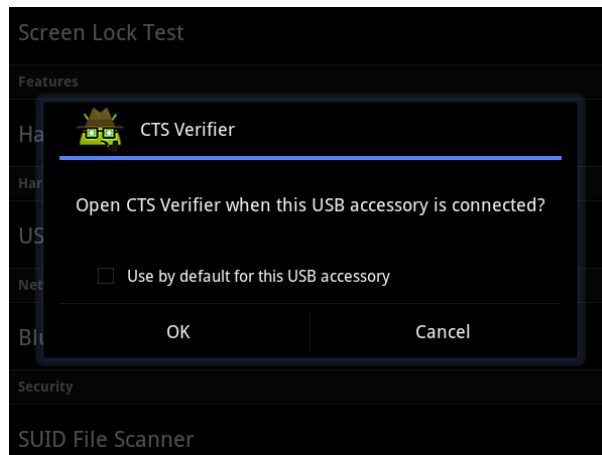
### 5.3.1. USB Accessory

In order to run the USB Accessory test, a Linux computer will be needed in order to run the USB host program.

- Connect the DUT to a computer
- Execute the cts-usb-accessory program on the computer found in the CTS

Verifier package.

- A popup message will appear on the DUT. Select 'OK' and go into the USB Accessory Test in the CTS Verifier application.



- Console 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

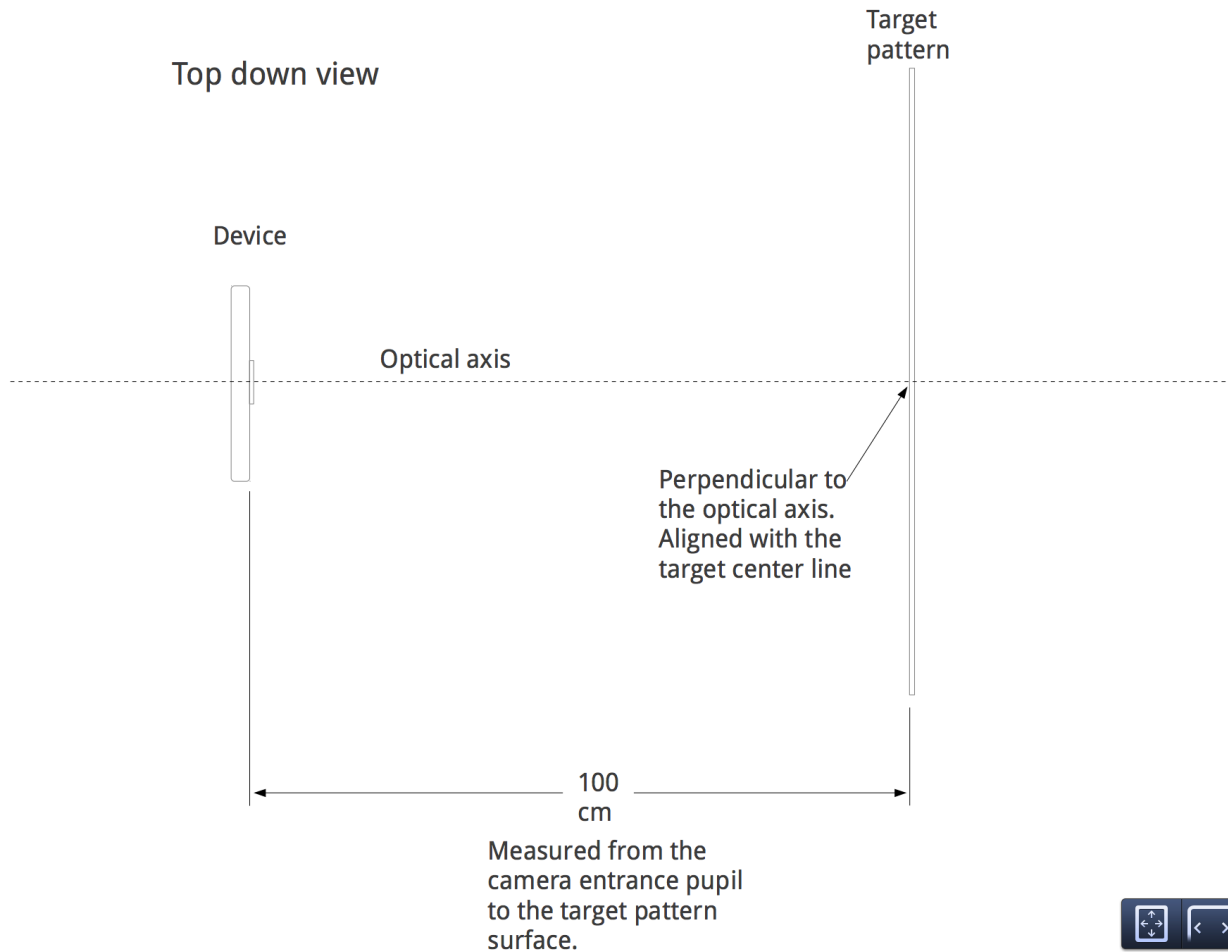
This field of view calibration procedure is designed to be a quick way to determine the device field of view with moderate accuracy.

#### Setup

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

<http://source.android.com/compatibility/calibration-pattern.pdf>

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. Hit the setup button and select "Marker distance" to enter the distance.
3. Measure and enter the distance to the target pattern (~100 cm).
4. Hit 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)

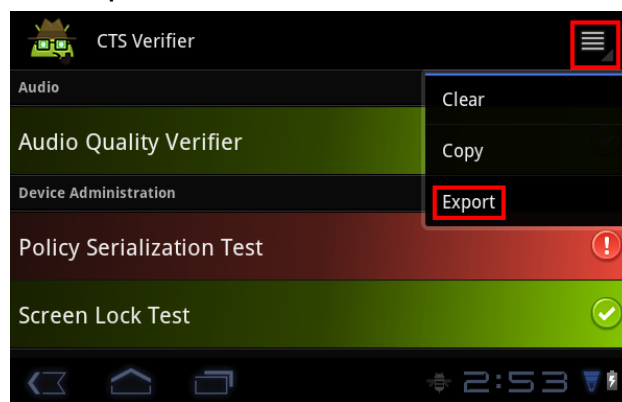
In the calibration mode, the photo will be displayed with two vertical lines overlaid onto the image.

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).

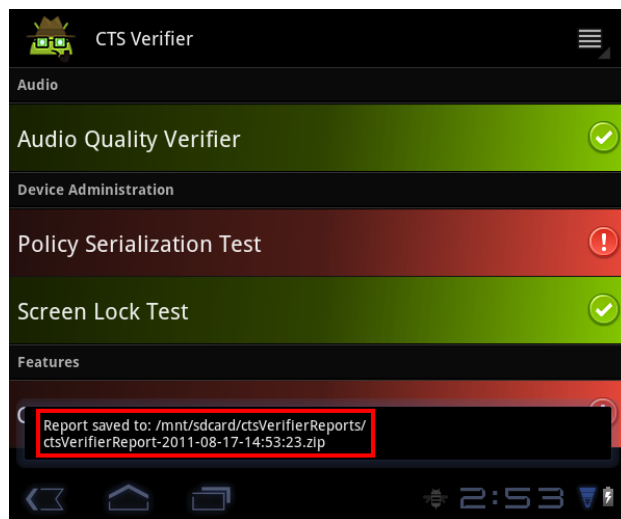
Adjust the slider at the bottom of the screen until the overlay aligns with the target pattern as closely as possible. The displayed field of view will be a close approximation to the correct value when the overlay and the target pattern image are aligned. The reported field of view should be within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

## 5.4. Exporting test reports

- After all tests are completed, tap the MENU button from the CTS Verifier home screen, and select "Export".



- A path to the saved report will be displayed in pop-up (e.g. `/mnt/sdcard/ctsVerifierReports/ctsVerifierReport-date-time.e.zip`). Record the path. *Future releases of CTS Verifier will support reporting directly from the device.*



- 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

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

### Run

<i>run cts</i>	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.
<i>--plan &lt;test_plan_name&gt;</i>	Run the specified test plan
<i>--package/-p &lt;test_package_name&gt; [-package/-p &lt;test_package2&gt;...]</i>	Runs the specified test packages.
<i>--class/-c &lt;class_name&gt; [-method/-m &lt;test_method_name&gt;]</i>	Runs the specified test class and/or

<i>--continue-session-id</i>	method
<i>--shards &lt;number_of_shards&gt;</i>	Runs all not executed tests from previous CTS session. The sessions testResult.xml will be updated with the new results.
<i>--serial/-s &lt;deviceId&gt;</i>	Shard a CTS run into given number of independent chunks, to run on multiple devices in parallel.
	Run CTS on the specific device

## List

<i>list packages</i>	List all available test packages in the repository.
<i>list plans</i>	Lists all available test plans in the repository
<i>list invocations</i>	Lists 'run' commands currently being executed on devices.
<i>list commands</i>	List all 'run' commands currently in the queue waiting to be assigned to devices
<i>list results</i>	List CTS results currently stored in repository
<i>list devices</i>	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.