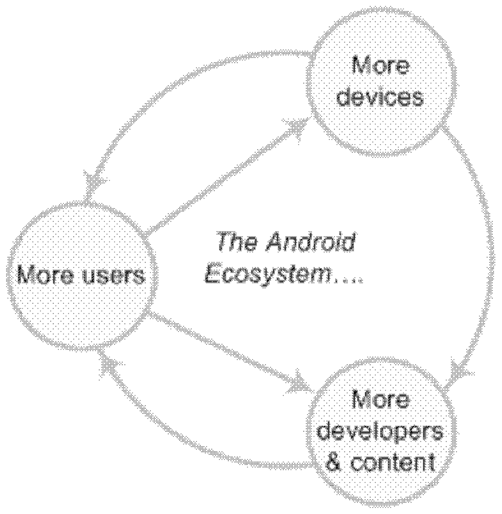
android compatibility program

Compatibility Test Suite (CTS) Framework User Manual

Android 1.6 CTS r4
Open Handset Alliance

Contents

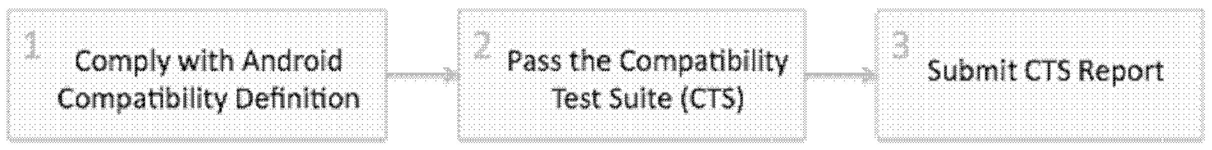
1. Why be compatible?



1. *Give your users the best possible experience with the applications they run.* When a device is compatible with Android, users can choose from among many high-quality applications. Applications that take full advantage of Android's features are likely to perform best on compatible devices.
2. *Make it easy for developers to write top-quality applications for your device.* Developers want to streamline their applications for Android, and this is easiest for them when they are writing for a predictable platform.
3. *Take advantage of the Android Market.* Compatible handsets can give users access to the Android Market.

Android compatibility is free, and it's easy.

2. How can I become compatible?


1 Comply with Android Compatibility Definition
2 Pass the Compatibility Test Suite (CTS)
3 Submit CTS Report

2.1. Comply with Android Compatibility Definition document

To start, read the Android compatibility definition for the Android platform version that you want. This document enumerates the software and the hardware features in a compatible Android device. Except where noted, the features are all required for Android compliance. To learn more about Android compatibility definition in general, and to locate and download a particular definitions document, see the current Compatibility Definition. Archived versions of older Compatibility Definitions may be found on the Downloads page.

2.2. Pass the Compatibility Test Suite (CTS)

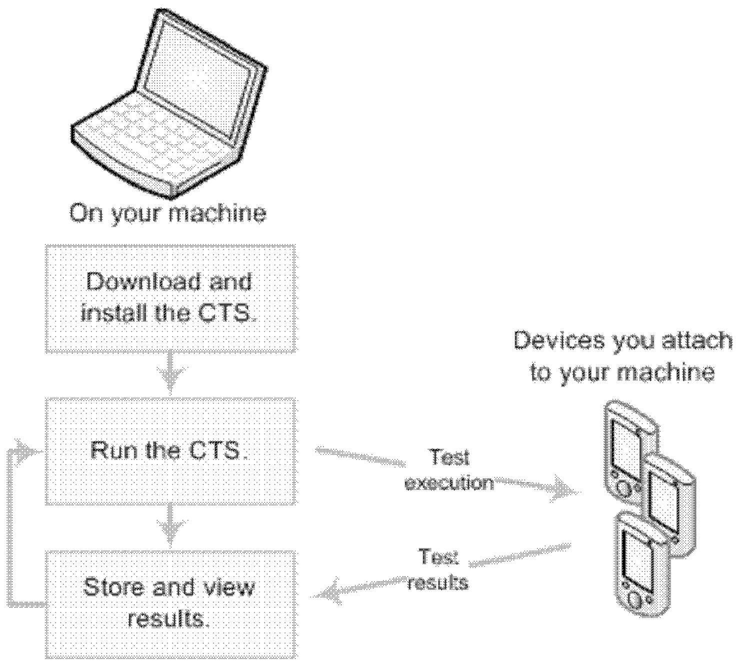
The Compatibility Test Suite (CTS) is a downloadable open-source testing harness that you can use in any way you like as you develop your handset; for example, you could use the CTS to do continuous self-testing during your development work. For more about the CTS and the compatibility report that it generates, see the Compatibility Test Suite page. For instructions on using the CTS, see the CTS User Guide.

2.3. Submit report

When you are ready to claim compatibility for your device, you can submit the CTS-generated report to cts@android.com. When you submit a CTS report, you can also request access to the Android Market.

* This is an early preview of CTS. The compatibility site and the service to certify your compatibility reports are work in progress - we will update you when these are ready.

3. How does the CTS work?



The CTS is an automated testing harness that includes two major software components:

- The CTS test harness runs on your desktop machine and manages test execution.
- Individual test cases are executed on attached mobile devices or on an emulator. The test cases are written in Java as JUnit tests and packaged as Android .apk files to run on the actual device target.

3.1. Workflow

1. Use the bundled CTS release or download the CTS from the Android Open Source Project onto your desktop machine.
2. Install and configure the CTS.
3. Attach at least one device (or emulator) to your machine.
4. Launch the CTS. The CTS test harness loads the test plan onto the attached devices. For each test in the test harness:
    - The test harness pushes a .apk file to each device, executes the test through instrumentation, and records test results.
    - The test harness removes the .apk file from each device.
5. Once all the tests are executed, you can view the test results in your browser and use the results to adjust your design. You can continue to run the CTS throughout your development process.

When you are ready, you can submit the report generated by the CTS to cts@android.com. The report is a .zip archived file that contains XML results and supplemental information such as screen captures.

3.2. Types of test cases

The CTS includes the following types of test cases:

- *Unit tests* test atomic units of code within the Android platform; e.g. a single class, such as java.util.HashMap.
- *Functional tests* test a combination of APIs together in a higher-level use-case.
- *Reference application tests* instrument a complete sample application to exercise a full set of APIs and Android runtime services

Future versions of the CTS will include the following types of test cases:

- *Robustness tests* test the durability of the system under stress.
- *Performance tests* test the performance of the system against defined benchmarks, for example rendering frames per second.

3.3. Areas Covered

The unit test cases cover the following areas to ensure compatibility

Area	Description
Signature tests	For each Android release, there are XML files describing the signatures of all public APIs contained in the release. The CTS contains a utility to check those API signatures against the APIs available on the device. The results from signature checking are recorded in the test result XML file.
Platform API Tests	Test the platform (core libraries and Android Application Framework) APIs as documented in the SDK Class Index to ensure API correctness: • correct class, attribute and method signatures • correct method behavior • negative tests to ensure expected behavior for incorrect parameter handling
Dalvik VM Tests	The tests focus on testing the Dalvik VM

Platform Data Model	The CTS tests the core platform data model as exposed to application developers through content providers, as documented in the SDK android.provider package: • contacts • browser • settings • more...
Platform Intents	The CTS tests the core platform intents, as documented in the SDK Available Intents.
Platform Permissions	The CTS tests the core platform permissions, as documented in the SDK Available Permissions.
Platform Resources	The CTS tests for correct handling of the core platform resource types, as documented in the SDK Available Resource Types. This includes tests for: • simple values • drawables • nine-patch • animations • layouts • styles and themes • loading alternate resources

4. Setting up and using the CTS

4.1. Configuring the CTS

To run CTS, make sure you have atleast the Android 1.6 r1 SDK installed on your machine. ****There are changes to adb in 1.6 that will cause CTS to not work correctly with older versions of adb.****

To configure CTS, extract the contents of the zip file and edit the `android-cts/tools/startcts` script - modify the variable `SDK_ROOT` to match your environment.

Example:

```
SDK_ROOT=/home/myuser/android-sdk-linux_x86-1.6_r1
```

This should point to the top-level directory where you unzipped the Android 1.6 SDK to.

4.2. Setting up your device

CTS can be executed only on consumer device since Android 1.6 -- you can run CTS only on developer builds for Android 1.0 and 1.5.

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

1. Please download and install the Android 1.6 SDK on your machine.
2. Your phone should be running a **user build (Android 1.6 and later)** from source.android.com
3. Please refer to this link on the Android developer site and set up your device accordingly.
4. Make sure that your device has been flashed with a user build (Android 1.6 and later) before you run CTS.
5. You need to download the TTS files via Settings > Speech synthesis > Install voice data before running CTS tests. (Note that this assumes you have Android Market installed on the device, if not you will need to install the files manually via adb)
6. It is advisable to log in to the device with a test Google account, not an account that you actually use.
7. Make sure the device has a SD card plugged in and the card is empty. *Warning: CTS may modify/erase data on the SD card plugged in to the device.*
8. Do a factory data reset on the device (Settings > SD Card & phone storage > Factory data reset). *Warning: This will erase all user data from the phone.*
9. Make sure no lock pattern is set on the device (Settings > Security & location > Require Pattern should be unchecked.

10. Make sure the "Screen Timeout" is set to "Never Timeout" (Settings > Sound & Display > Screen Timeout should be set to "Never Timeout".
11. Make sure the "Stay Awake" development option is checked (Settings > Applications > Development > Stay awake).
12. Make sure Settings > Application > Development > Allow mock locations is set to true.
13. Make sure the device is at the home screen at the start of CTS (Press the home button).
14. While a device is running tests, it must not be used for any other tasks.
15. 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.

4.3. Using the CTS

To run a test plan:

1. Make sure you have at least one device connected (or the emulator running). Launch the CTS console by running the *startcts* script which you modified to match your environment, e.g.
   `$ bash android-cts/tools/startcts`
2. You may start the default test plan (containing all of the test packages) by typing `start --plan CTS`. This will kick off all the CTS tests required for compatibility.
   Type `ls -p` to see a list of test packages in the repository.
   Type `ls --plan` to see a list of test plans in the repository.
   See the CTS command reference or type `help` for a complete list of supported commands.
3. Alternately, you can just run a CTS plan from the command line using
   `startcts start --plan <plan_name>`
4. You should test progress and results reported on the console.

4.4. Selecting CTS Plans

For this release the following 7 test plans are available.

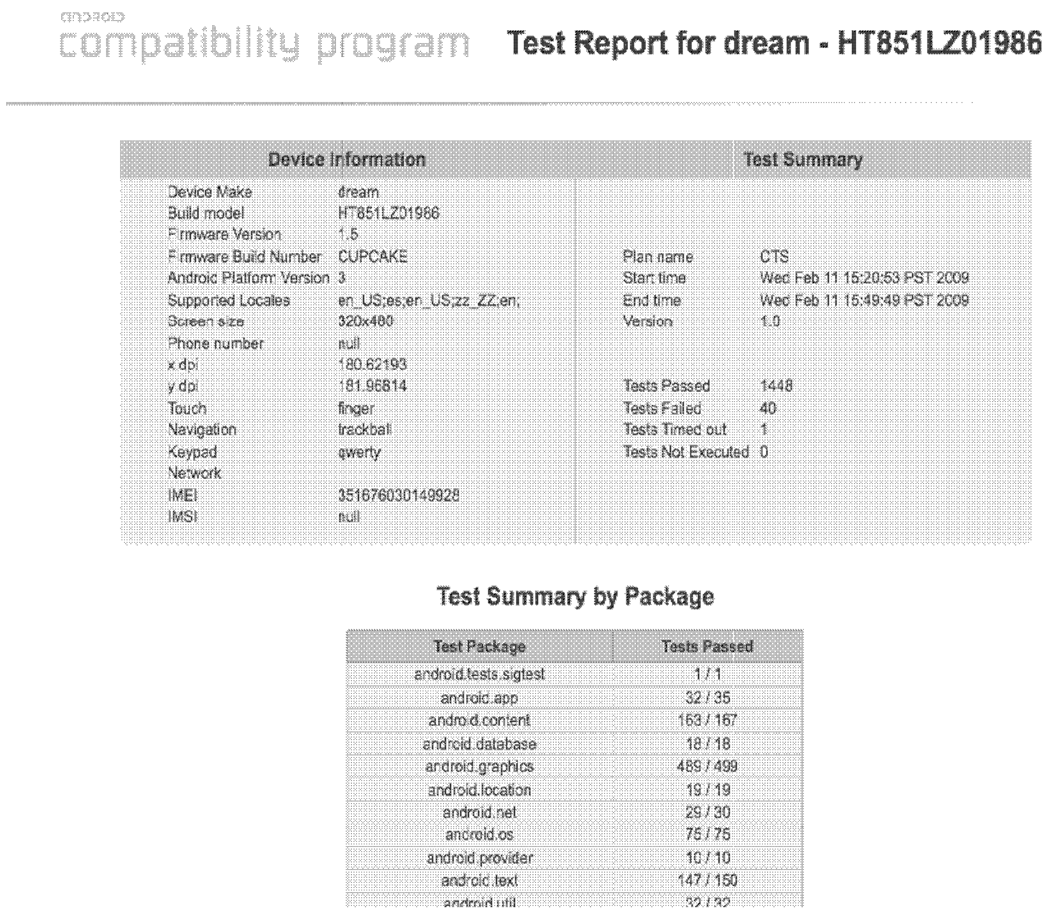
1. `CTS` - contains all tests and will run ~21,000 tests on your device. These tests are required for compatibility. At this point performance tests are not part of this plan (this will change for future CTS releases).
2. `Signature` - contains the signature verification of all public APIs
3. `Android` - contains tests for the android APIs
4. `Java` - contains tests for the Java core library
5. `VM` - contains tests for the Dalvik virtual machine

6. `RefApp` - contains reference application tests (more coming in future CTS release)
7. `Performance` - contains performance tests for your implementation (more coming in future CTS releases)

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

5. Interpreting the Test Results

The test results are placed in the file:
`$CTS_ROOT/repository/results/<start time>.zip`
Inside the zip, the `testResult.xml` file contains the actual results -- open this file in any web browser (Firefox 3.x recommended) to view the test results.



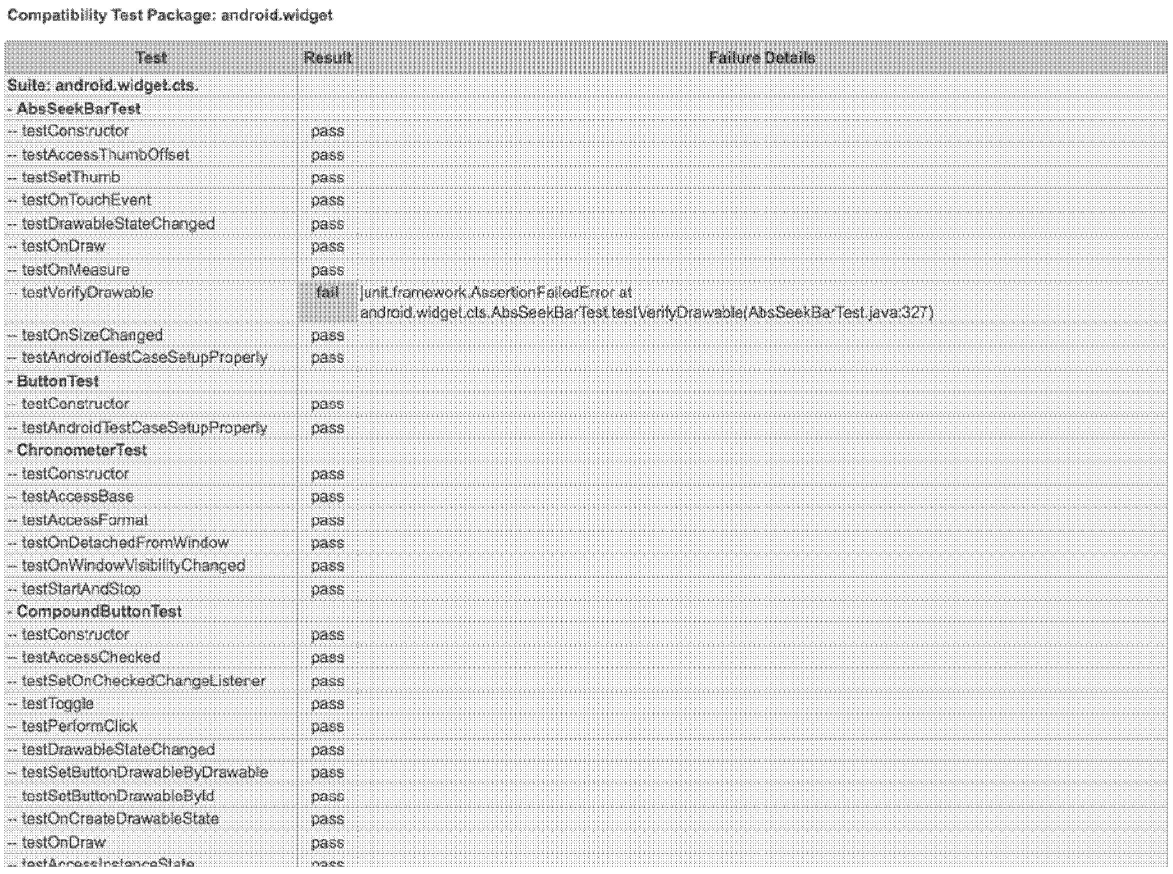
android
compatibility program **Test Report for dream - HT851LZ01986**

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

6. Release Notes

6.1. General

- This CTS release contains approximately 21,000 tests that you can execute on the device.
- Please make sure all steps in section 4.2 "Setting up your device" have been followed before you kick off CTS. Not following these instructions may cause tests to timeout or fail.

6.2. Known Issues

- The framework restarts the device periodically -- this is expected behavior.
- Concurrent devices are not supported in this release -- CTS can be executed on only one device at a given time.
- The CTS console allows the user to derive a new test plan based on previous results. This is useful for re-running tests that did not pass in a previous run. Successive derivation of test plans (i.e. deriving a test plan from test results of an already derived test plan) may result in the plan including extra tests -- this is a known issue for this release.
- Occasionally while running the tests, a system dialog may pop up informing the user that process 'android.process.acore' is not responding. The user is given the option to kill the process or wait for it to respond. This alert dialog interferes with some tests by grabbing all key and pointer events, causing the tests to fail. Re-running the tests usually fixes the problem.

7. Appendix: CTS Console Command Reference

Host	
help	Display the list of available commands.
exit	Exit the CTS console.
Test Plan	
ls --plan [<test_plan_name>]	Displays the contents of the specified test plan. If no plan is specified, a list of all plans is displayed.
add --plan <new_plan_name>	Create a new test plan. The console will guide you through the test packages to select the tests you want to include in your plan. Note that the plan name must be unique.
add --derivedplan <new_plan_name> [-s <session_id>] [-r [pass | fail | timeout | notExecuted]]	Create a new test plan from an existing result. This test plan will consist of all test with the specified result type in the specified session. If no result type is given, all but the passed tests are included. If no session is given, the latest results are used.
rm --plan <test_plan_name | all>	Remove the specified plan from the plan repository. *all* removes all test plans
start --plan <test_plan_name>	Start the specified test plan and displays progress information. The console will only prompt for further commands when the plan has run to completion. If there are available test sessions for the specified test plan, the CTS console will prompt user to choose between two options: (1) Choose a session from the existing sessions;

	(2) Create a new session. If more than one device connected, CTS host will prompt user to choose one device.
-d, --device <device_id>	Start the specified test plan using the specified device.
-t, --test <test_name>	Start to run the specified test contained in the specified test plan
-p, --package <java_package_name>	Start to run the specified Java package contained in the specified plan.
Test Package	
ls *-p, --package [<package name>]*	List all available test packages in the repository. If package name is specified then lists all its test suites/test cases.
add -p, --package <zip_file_path>	Add new packages to the case repository.
rm -p, --package [<package_name> | all]	Remove the specified package from the case repository. *all* removes all packages
Test Result	
ls *-r, --result* *[pass | fail | timeout | notExecuted]* *[-s <session_id>]*	List the results for all available sessions. If session_id is specified, then lists results for that specific session. If pass, fail, etc. is specified then filters test results based on the specified results.
History	
history | h [<count>] [-e <num>]	List all commands in history. If count is specified, last count commands in history are shown -e allows the command with number num to be executed directly from history
Device	
ls -d, --device	List all attached devices.

android
compatibility program