WIND RIVER

Wind River Framework for Automated Software Testing

Wind River Framework for Automated Software Testing (FAST) was developed to help customers overcome the challenges of testing complex open source software distributions for mobile devices and automotive systems. Wind River FAST provides a system for combining multiple test frameworks into a single meta-framework that can be run with "a single button press." It helps automate, execute, and manage thousands of tests and their results.

Wind River leverages many open source and existing test frameworks so the test scope is comprehensive and respective of real-world quality requirements, while ensuring time-to-market needs. Wind River FAST makes executing tests much simpler because they are all wrapped in FAST scripts and run from the FAST graphical user interface (GUI) and an available command-line interface (CLI).

Challenge of Testing

Testing very large open source software distributions containing thousands of packages is a tremendous challenge. There are typically dozens of subsystems included in a distribution, with many applications built using those packages. Differences in device types, peripherals used, and user interface variations complicate the test and validation process. To properly validate the distribution, tests for the following are needed:

- Requirements
- Performance
- Stress and stability
- Device driver
- Middleware
- Application
- User interface
- Multi-device
- Peripherals
- Board support packages (BSP)
- Version checking
- Telephony
- Network

Each of these types of tests needs to be performed as often as possible on each new release, and the results of these tests need to be reported and managed.

Because test programs are pulled from so many different sources, and they all have diverse ways of being run and generating and reporting results, executing the tests and tracking and evaluating the test results can complicate and delay the entire testing process. The amount of information can be staggering, particularly when test cases have different reporting formats.

As new original device manufacturers (ODMs) develop hardware and wish to have their products tested with a certain distribution of the software, the developer or quality assurance (QA) engineer needs to adapt tests to run on each device. Each target may have a unique hardware configuration, peripherals, file storage systems, and so on. Tests that were developed for one device frequently need to be adapted to work on different devices.

In addition, user interfaces (UI) that come with different devices are often changing. Even if the functionality behind the UI is the same, the developer or QA engineer must adapt tests to the different layouts and widget styles.

With Wind River FAST, each individual test framework or application is encapsulated within a "wrapper" script executed by the FAST system. The results are all stored in a consistent manner in a centralized database. This provides a flexible and comprehensive system for handling large amounts of test results from heterogeneous test frameworks, and vastly simplifies the problem of wading through and comparing the various individual test reports.

The database contains all test artifacts and logs from the original test programs, and additional metrics are collected. Wind River FAST can be thought of as a meta-test framework that can run other test frameworks and individual tests.

Using the Wind River FAST database allows a variety of test reports to be created. Customizable formal reports (for QA managers), test run comparison reports (for developers), and interactive regression analysis reports (for engineering managers) are all provided. Additional reports can easily be created using the reporting tools of your choice.

Table of Contents

Wind River's Solution

Wind River FAST offers significant benefits to both developers and QA engineers:

- **Integrates testing efforts in one application:** Once the different test frameworks and applications are "wrapped" by a FAST script, they are accessible from a test GUI. Users may select which tests to execute, and with one button execute them all. There is no need to know how the test applications work or what report formats they use; everything is handled by scripts.
- **Integrates other testing frameworks:** Any existing test framework or test application can be integrated into Wind River FAST using a FAST script. The script uses advanced parsing routines to decode the test results from any log file or program output. No matter where the tests come from and how they present test results, FAST scripts can integrate them into the system.
- **Creates custom test scripts:** Often there are no open source or existing test assets available. Wind River or the customer can create custom scripts in Wind River FAST that can execute commands on both a test server and targets and analyze their output. With this capability, a whole suite of tests that did not previously exist can be created.
- **Applies identical tests to different targets with minimal changes:** Wind River FAST is structured so that test scripts/wrappers never directly communicate with the targets. They always go through a "board" class that abstracts the test framework from the specific requirements of a board. The board class can send commands to the target, read the output, install a payload, and reboot the target. There are also special methods that perform common operations that are repeated in many tests (i.e., setting up network interfaces). When a new target is introduced, Wind River FAST only needs to modify the board class. Only the communication and interface methods are rewritten. The actual test scripts and wrappers need not change and can be used to test both target types. Once a script is written, it most likely will work on multiple targets. Occasionally minor changes are needed if the functionality of an application/package changes from one target to another; but for the most part, Wind River test libraries do not require many changes.

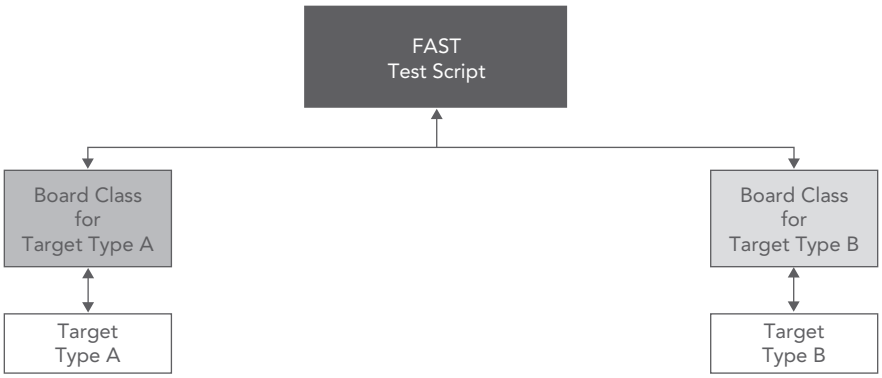


Figure 1: Wind River FAST test script configuration for multiple targets using board class

- **Leverages advanced reporting capabilities:** Because all the results and test artifacts go into one database, it is relatively easy to generate reports. These reports can serve the following functions:
  - Provide customers with a list of tests performed and their results.
  - Compare the current test run to a previous test run. This is useful to the software design team because they can see what new bugs were introduced and what bugs were fixed since the last time tests were run.
  - Illustrate performance results graphically so that managers can quickly see if identified metrics are degrading or improving over time.
  - Track requirements, last test dates, and results. This makes it easy to see where more tests need to be added and which requirements need to be debugged.
  - Summarize test history to show the results and artifacts of a test over the past several runs. This is useful for developers to debug issues that have recently been introduced in the software.

Results from other test-tracking software (such as manual test-tracking software) can be imported into the Wind River FAST database so FAST can track the test effort of the entire test organization.

Wind River UX Test Development Kit

Wind River UX Test Development Kit for Android is an Eclipse-based test authoring environment that enables the rapid creation of automated test scripts for Android devices. The test development kit has been built from the ground up to offer automated testing capabilities that replace human efforts. Tests developed using UX Test Development Kit automatically manipulate the device under test just like a real user would through Android's user interface. The automated execution of these tests provides a significant reduction (up to 70%) in time spent during the manual testing cycles of the complex Android software stacks.

UX Test Development Kit tests automate the interaction with applications and web pages exposed through the native Android browser. It is also able to simulate virtually all user interactions with the target, including the following:

- Clicking widgets in the user interface
- Dragging graphical elements
- Reading the contents of the UI elements
- Typing entries on a physical or virtual keyboard

This is a comprehensive way of replicating and automating the Android user experience while interacting with various graphic interface elements.

FAST for Android includes tests developed using UX Test Development Kit for all standard preloaded Android applications as well as sample test scripts for popular applications such as Google Maps, Facebook, and Twitter.

UX Test Development Kit offers a unique approach to real-world UX testing by making the UI elements accessible for validation and manipulation from automated test scripts. These scripts are automatically executed using FAST for Android.

Test Execution

Configuring a target for testing is easy and straightforward. Wind River FAST uses a board class to specify communication and control for the target. With a

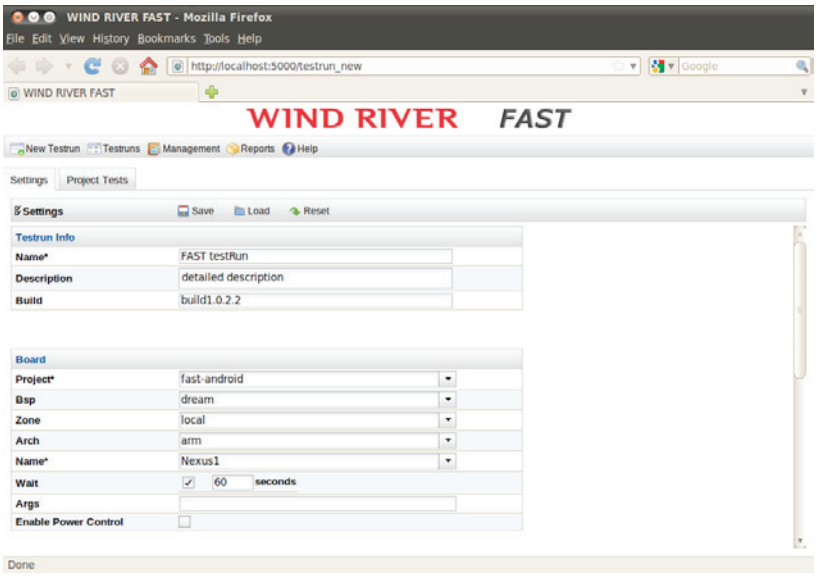


Figure 2: Wind River FAST user interface

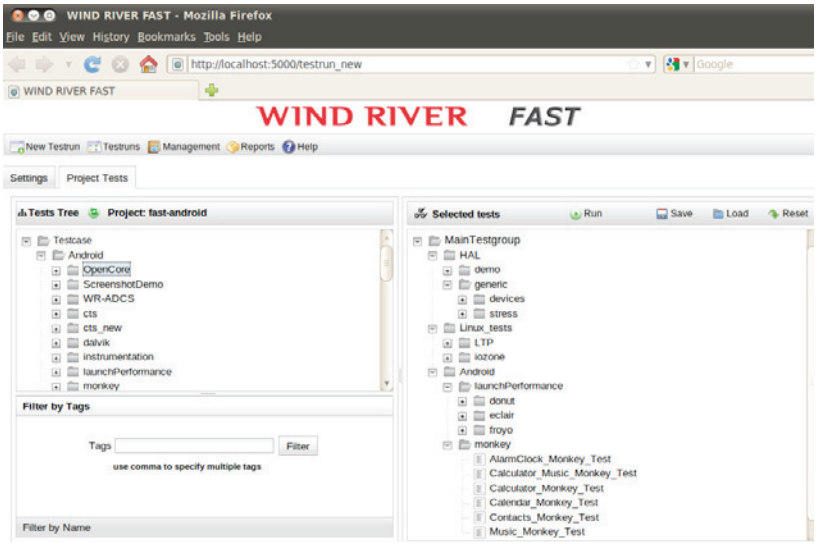


Figure 3: Wind River FAST test lists

board class and network connection to the target, the tester is ready to begin testing.

The FAST user interface offers significant flexibility to automate the testing process:

- Software may be loaded onto the target before testing begins, and these uploaded files may be removed once testing is completed.
- Special arguments may be executed on the command line for boards that require them.
- The target may be configured to power down or reboot during testing or between tests.
- Baseline results may be identified for comparing test results in the test report.

The web-based graphical user interface makes test selection and execution a quick process. The user navigates the test tree at the left and selects individual tests for running by dragging and dropping. Each test may then be configured, including default time-out, count, parameters, and reboot options (reboot before, after, or after the test does not pass). Once a test run is configured, the progress of each test is displayed, including the start and end dates, any test cases run, their status (passed/failed), and any artifacts, such as log files, or key values (performance metrics) that are associated with the test case.

Wind River FAST supports a command-line interface that allows execution of tests from the Linux command line. This is very useful for running tests regularly from a cron job. Running tests regularly is an excellent tool for catching bugs early. A report emailed to the developers every day can show what has changed in the build since the last time it was tested and what bugs were introduced, while recent changes to the software are still fresh in the developers' minds. Because the changes are recent, it is much easier to identify the cause of the bugs rather than getting feedback weeks or even months later on hundreds of changes made to the software. This frequent, even daily, feedback makes it much faster to identify a change that introduced a bug.

By storing performance metrics and running tests regularly, managers can see the impact of changes to the performance of the system. They can see whether performance dropped off radically after a certain check-in or whether performance is gradually dropping or improving over time. This prevents surprises at the end of a release testing cycle.

In the web GUI, the user can specify the test environment and select the desired tests. Instead of creating test runs each time, the user can save an INI file that contains all tests planned for execution. The INI file may be loaded into the system at a later date with settings specific to that file to pass it to FAST for execution. This executes the tests in the same fashion as if they were executed from the GUI.

The command-line interface also has the capability to stop a test run, get its status, and download its results and artifacts.

FAST also offers scripts that can automatically analyze the FAST database to find the best tests to execute, based on the time passed since their last execution and their last execution status (giving higher priority to those tests that have not run recently and those that last failed). Using these scripts, FAST can run tests for a specific amount of time (e.g., eight hours). The system will find the best tests to run that will approximately fill the time (tests typically take the same amount of time to run, but if a bug is introduced, certain tests may take a longer or shorter time to execute). These scripts automatically create an INI file containing the best tests to run.

Test Library

Wind River has a large suite of tests created for mobile platforms, which are run regularly. A dedicated team continues to add more tests to this library, so it is constantly expanding. Tests are grouped into the following categories, with some overlap:

- **Requirements:** Wind River has defined more than 1,000 requirements across our mobile and automotive commercial platforms. All requirements are tested with a combination of manual and automated tests developed by Wind River. Additional tests can easily be added to cover customer-specific requirements.
- **Compliance:** Wind River FAST includes the latest version of the compliance suites for Android Compatibility Test Suite (CTS), MeeGo Core Test Suite (MCTS), and GENIVI.
- **System framework:** There is a complete set of system framework tests specifically for Wind River Platform for Android and Wind River Platform for Infotainment.
- **Test RPMs:** Many of the open source packages used in Mobile Linux include their own tests. These are usually difficult and procedurally cumbersome to run frequently. The Wind River Linux Distribution Assembly Tool (LDAT) build system for Mobile Linux extracts these into test RPMs that can be executed by FAST, providing an ongoing regression test of these components, thus making it much easier to detect failures caused by changes in other areas of the system.
- **Packages:** Wind River has created tests to assure that packages that are part of the Mobile Linux system that do not have their own tests are installed and operational.
- **Open source:** TETware, Linux Test Project (LTP), IOZone, IPerf, and Android CTS are all integrated. Additional third-party test frameworks are straightforward to add.
- **Performance:** These include automated tests for graphics, networking, and boot-time performance.
- **Interoperability (IOT):** IOT tests include both automated and manual tests for Bluetooth and Wi-Fi interoperability.
- **Telephony:** Wind River offers automated tests for its telephony stacks for Android.
- **BSPs:** Tests include IOZone for board support packages (BSPs).
- **Stress/stability/soak:** Wind River FAST provides an easy mechanism for combining multiple tests that stress the system under unusual loads. Most commonly, FAST accomplishes stress tests with LTP, but any automated test can be used into this kind of test scenario.
- **User interface:** Automated test suites for Wind River Platform for Android user interfaces are created using UX Test Development Kit. UX Test Development Kit tests are fully managed and automatically executed by Wind River FAST in the same manner as any other tests.

Reporting

Generating test reports from Wind River FAST is easy. Report generation can be controlled from the FAST GUI or can be executed from the CLI.

There are five different types of reports available in the standard FAST system:

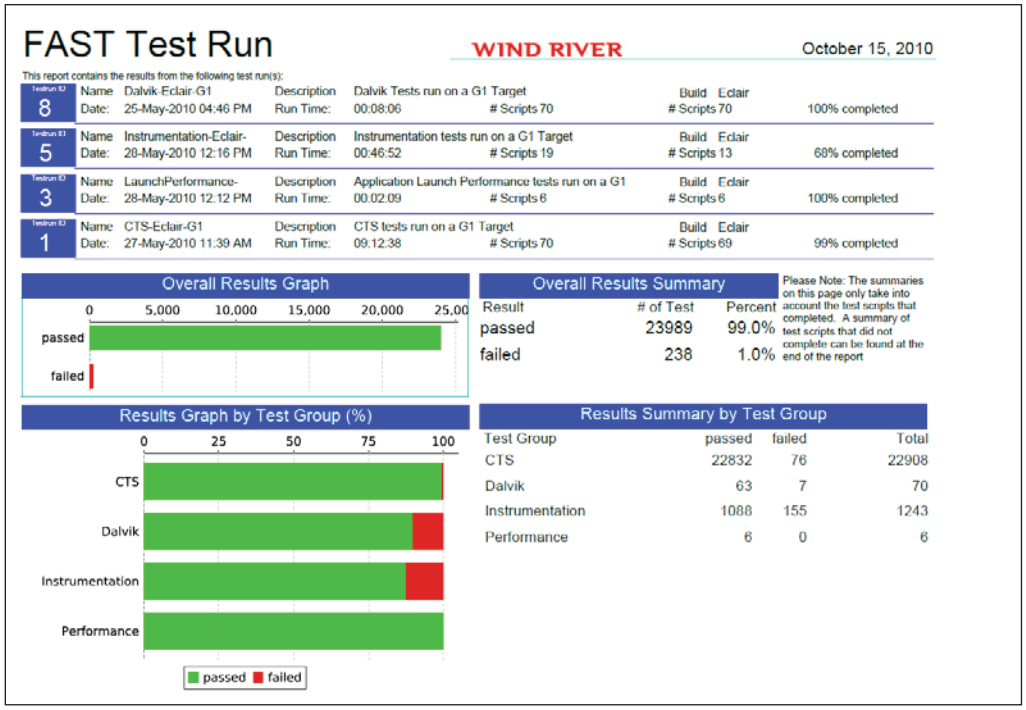
- **Final:** Final reports contain a list of all the test cases executed and their status. See Appendix Figures 1 and 2 for samples.
- **Failure:** Failure reports are very similar to final reports, except they only show the test cases that failed. This is good for seeing a list of all features that need to be fixed in a release.
- **Comparative:** Comparative reports compare two test runs against each other to show what has changed between the two. This report summarizes the pass/fail rates of each and shows tests with different results, tests that are new, and tests that have been removed from the test run. This report is especially useful as a nightly report for designers because it shows only what has changed since the last time the software was tested. See Appendix Figures 3 and 4 for samples.
- **Performance:** Performance reports take all of the performance metrics and display them as graphs over time (e.g., showing the last 30 test runs and highlighting the current test run). This allows developers and managers to see whether a change in the software on a certain date caused a degradation (or improvement) of a particular performance metric. See Appendix Figure 5 for a sample report.
- **Detailed Test Script:** This report allows you to see all the keyvals and artifacts for a given test script instance. It contains an overview of the test script instance (and a graph showing its historical results) and a list of all the results that were recorded during the execution.

Adding new reports is relatively easy. Since all the information is stored in a MySQL database, you can use your preferred reporting engine to generate a report or engage Wind River to create new reports that meet your needs. Wind River FAST uses the open source JasperReports to create and generate the reports as part of the standard delivered product.

Figure 4: Wind River FAST test report window

Appendix

The following images are samples of the types of reports that can be generated with the FAST system. In addition to these reports, others can be created.



FAST Test Run

WIND RIVER

October 15, 2010

This report contains the results from the following test run(s):

Testrun ID						
8	Name: Dalvik-Eclair-G1	Description: Dalvik Tests run on a G1 Target		Build: Eclair		
	Date: 25-May-2010 04:46 PM	Run Time: 00:08:06	# Scripts: 70	# Scripts: 70	100% completed	
5	Name: Instrumentation-Eclair-	Description: Instrumentation tests run on a G1 Target		Build: Eclair		
	Date: 28-May-2010 12:16 PM	Run Time: 00:46:52	# Scripts: 19	# Scripts: 13	68% completed	
3	Name: LaunchPerformance-	Description: Application Launch Performance tests run on a G1		Build: Eclair		
	Date: 28-May-2010 12:12 PM	Run Time: 00:02:09	# Scripts: 6	# Scripts: 6	100% completed	
1	Name: CTS-Eclair-G1	Description: CTS tests run on a G1 Target		Build: Eclair		
	Date: 27-May-2010 11:39 AM	Run Time: 09:12:38	# Scripts: 70	# Scripts: 69	99% completed	

Overall Results Graph

Overall Results Summary

Result	# of Test	Percent
passed	23989	99.0%
failed	238	1.0%

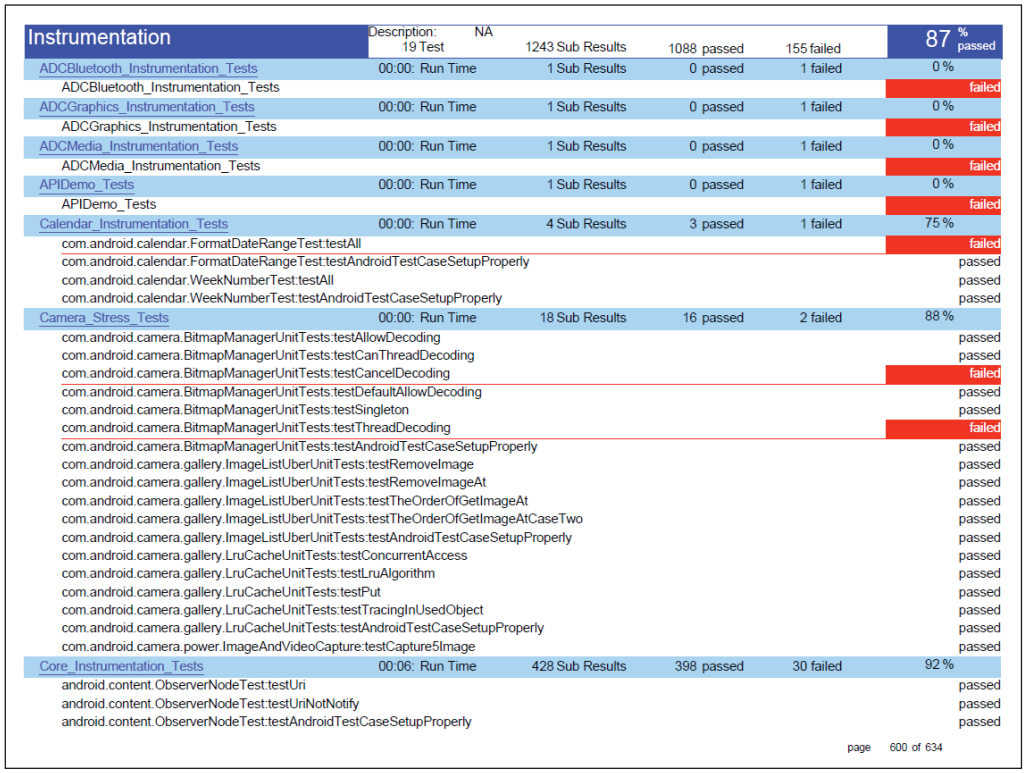
Please Note: The summaries on this page only take into account the test scripts that completed. A summary of test scripts that did not complete can be found at the end of the report

Results Graph by Test Group (%)

Results Summary by Test Group

Test Group	passed	failed	Total
CTS	22832	76	22908
Dalvik	63	7	70
Instrumentation	1088	155	1243
Performance	6	0	6

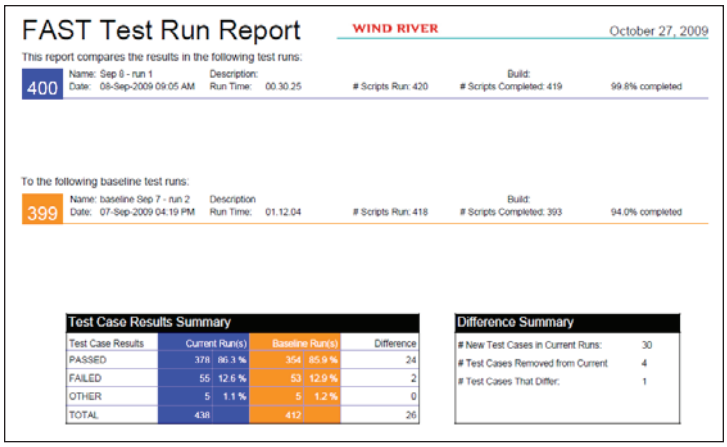
Appendix Figure 1: Sample final report—summary



Instrumentation	Description: NA				
	19 Test	1243 Sub Results	1088 passed	155 failed	87 % passed
ADCBluetooth_Instrumentation_Tests	00:00: Run Time	1 Sub Results	0 passed	1 failed	0 %
ADCBluetooth_Instrumentation_Tests					failed
ADCGraphics_Instrumentation_Tests	00:00: Run Time	1 Sub Results	0 passed	1 failed	0 %
ADCGraphics_Instrumentation_Tests					failed
ADCMedia_Instrumentation_Tests	00:00: Run Time	1 Sub Results	0 passed	1 failed	0 %
ADCMedia_Instrumentation_Tests					failed
APIDemo_Tests	00:00: Run Time	1 Sub Results	0 passed	1 failed	0 %
APIDemo_Tests					failed
Calendar_Instrumentation_Tests	00:00: Run Time	4 Sub Results	3 passed	1 failed	75 %
com.android.calendar.FormatDateRangeTest:testAll					failed
com.android.calendar.FormatDateRangeTest:testAndroidTestCaseSetupProperly					passed
com.android.calendar.WeekNumberTest:testAll					passed
com.android.calendar.WeekNumberTest:testAndroidTestCaseSetupProperly					passed
Camera_Stress_Tests	00:00: Run Time	18 Sub Results	16 passed	2 failed	88 %
com.android.camera.BitmapManagerUnitTests:testAllowDecoding					passed
com.android.camera.BitmapManagerUnitTests:testCanThreadDecoding					passed
com.android.camera.BitmapManagerUnitTests:testCancelDecoding					failed
com.android.camera.BitmapManagerUnitTests:testDefaultAllowDecoding					passed
com.android.camera.BitmapManagerUnitTests:testSingleton					passed
com.android.camera.BitmapManagerUnitTests:testThreadDecoding					failed
com.android.camera.BitmapManagerUnitTests:testAndroidTestCaseSetupProperly					passed
com.android.camera.gallery.ImageListUberUnitTests:testRemoveImage					passed
com.android.camera.gallery.ImageListUberUnitTests:testRemoveImageAt					passed
com.android.camera.gallery.ImageListUberUnitTests:testTheOrderOfGetImageAt					passed
com.android.camera.gallery.ImageListUberUnitTests:testTheOrderOfGetImageAtCaseTwo					passed
com.android.camera.gallery.ImageListUberUnitTests:testAndroidTestCaseSetupProperly					passed
com.android.camera.gallery.LruCacheUnitTests:testConcurrentAccess					passed
com.android.camera.gallery.LruCacheUnitTests:testLruAlgorithm					passed
com.android.camera.gallery.LruCacheUnitTests:testPut					passed
com.android.camera.gallery.LruCacheUnitTests:testTracingInUsedObject					passed
com.android.camera.gallery.LruCacheUnitTests:testAndroidTestCaseSetupProperly					passed
com.android.camera.power.ImageAndVideoCapture:testCapture5Image					passed
Core_Instrumentation_Tests	00:06: Run Time	428 Sub Results	398 passed	30 failed	92 %
android.content.ObserverNodeTest:testUri					passed
android.content.ObserverNodeTest:testUriNotNotify					passed
android.content.ObserverNodeTest:testAndroidTestCaseSetupProperly					passed

page 600 of 634

Appendix Figure 2: Sample final report—detail



FAST Test Run Report

WIND RIVER

October 27, 2009

This report compares the results in the following test runs:

400 Name: Sep 8 - run 1 Description: Build:
Date: 08-Sep-2009 09:05 AM Run Time: 00.30.25 # Scripts Run: 420 # Scripts Completed: 419 99.8% completed

To the following baseline test runs:

399 Name: baseline Sep 7 - run 2 Description Build:
Date: 07-Sep-2009 04:19 PM Run Time: 01.12.04 # Scripts Run: 418 # Scripts Completed: 393 94.0% completed

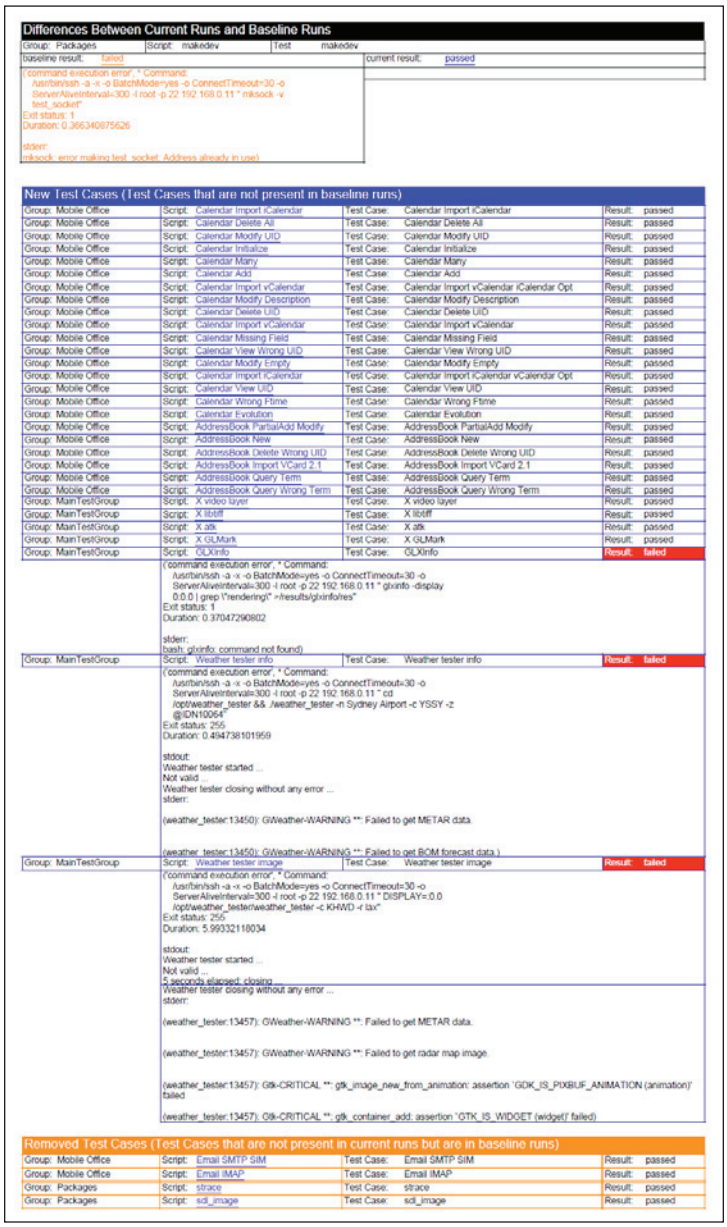
Test Case Results Summary

Test Case Results	Current Run(s)		Baseline Run(s)		Difference
PASSED	378	86.3 %	354	85.9 %	24
FAILED	55	12.6 %	53	12.9 %	2
OTHER	5	1.1 %	5	1.2 %	0
TOTAL	438		412		26

Difference Summary

# New Test Cases in Current Runs:	30
# Test Cases Removed from Current	4
# Test Cases That Differ:	1

Appendix Figure 3: Sample comparative report—summary



Differences Between Current Runs and Baseline Runs

Group: Packages Script: makedev Test makedev

baseline result: failed

("command execution error", " Command:
/usr/bin/ssh -a -x -o BatchMode=yes -o ConnectTimeout=30 -o
ServerAliveInterval=300 -l root -p 22 192.168.0.11 " mksock -v
test_socket"
Exit status: 1
Duration: 0.366340875626

stderr:
mksock: error making test_socket: Address already in use)

current result: passed

New Test Cases (Test Cases that are not present in baseline runs)

Group: Mobile Office	Script: Calendar Import iCalendar	Test Case: Calendar Import iCalendar	Result: passed
Group: Mobile Office	Script: Calendar Delete All	Test Case: Calendar Delete All	Result: passed
Group: Mobile Office	Script: Calendar Modify UID	Test Case: Calendar Modify UID	Result: passed
Group: Mobile Office	Script: Calendar Initialize	Test Case: Calendar Initialize	Result: passed
Group: Mobile Office	Script: Calendar Many	Test Case: Calendar Many	Result: passed
Group: Mobile Office	Script: Calendar Add	Test Case: Calendar Add	Result: passed
Group: Mobile Office	Script: Calendar Import vCalendar	Test Case: Calendar Import vCalendar iCalendar Opt	Result: passed
Group: Mobile Office	Script: Calendar Modify Description	Test Case: Calendar Modify Description	Result: passed
Group: Mobile Office	Script: Calendar Delete UID	Test Case: Calendar Delete UID	Result: passed
Group: Mobile Office	Script: Calendar Import vCalendar	Test Case: Calendar Import vCalendar	Result: passed
Group: Mobile Office	Script: Calendar Missing Field	Test Case: Calendar Missing Field	Result: passed
Group: Mobile Office	Script: Calendar View Wrong UID	Test Case: Calendar View Wrong UID	Result: passed
Group: Mobile Office	Script: Calendar Modify Empty	Test Case: Calendar Modify Empty	Result: passed
Group: Mobile Office	Script: Calendar Import iCalendar	Test Case: Calendar Import iCalendar vCalendar Opt	Result: passed
Group: Mobile Office	Script: Calendar View UID	Test Case: Calendar View UID	Result: passed
Group: Mobile Office	Script: Calendar Wrong Ftime	Test Case: Calendar Wrong Ftime	Result: passed
Group: Mobile Office	Script: Calendar Evolution	Test Case: Calendar Evolution	Result: passed
Group: Mobile Office	Script: AddressBook PartialAdd Modify	Test Case: AddressBook PartialAdd Modify	Result: passed
Group: Mobile Office	Script: AddressBook New	Test Case: AddressBook New	Result: passed
Group: Mobile Office	Script: AddressBook Delete Wrong UID	Test Case: AddressBook Delete Wrong UID	Result: passed
Group: Mobile Office	Script: AddressBook Import VCard 2.1	Test Case: AddressBook Import VCard 2.1	Result: passed
Group: Mobile Office	Script: AddressBook Query Term	Test Case: AddressBook Query Term	Result: passed
Group: Mobile Office	Script: AddressBook Query Wrong Term	Test Case: AddressBook Query Wrong Term	Result: passed
Group: MainTestGroup	Script: X video layer	Test Case: X video layer	Result: passed
Group: MainTestGroup	Script: X libtiff	Test Case: X libtiff	Result: passed
Group: MainTestGroup	Script: X atk	Test Case: X atk	Result: passed
Group: MainTestGroup	Script: X GLMark	Test Case: X GLMark	Result: passed
Group: MainTestGroup	Script: GLXinfo	Test Case: GLXinfo	Result: failed

("command execution error", " Command:
/usr/bin/ssh -a -x -o BatchMode=yes -o ConnectTimeout=30 -o
ServerAliveInterval=300 -l root -p 22 192.168.0.11 " glxinfo -display
0.0.0 | grep \"rendering\" >/results/glxinfo/res"
Exit status: 1
Duration: 0.37047290802

stderr:
bash: glxinfo: command not found)

Group: MainTestGroup	Script: Weather tester info	Test Case: Weather tester info	Result: failed

("command execution error", " Command:
/usr/bin/ssh -a -x -o BatchMode=yes -o ConnectTimeout=30 -o
ServerAliveInterval=300 -l root -p 22 192.168.0.11 " cd
/opt/weather_tester && ./weather_tester -n Sydney Airport -c YSSY -z
@IDN10064"
Exit status: 255
Duration: 0.494738101959

stdout:
Weather tester started ...
Not valid ...
Weather tester closing without any error ...
stderr:

(weather_tester:13450): GWeather-WARNING **: Failed to get METAR data.

(weather_tester:13450): GWeather-WARNING **: Failed to get BOM forecast data.)

Group: MainTestGroup	Script: Weather tester image	Test Case: Weather tester image	Result: failed

("command execution error", " Command:
/usr/bin/ssh -a -x -o BatchMode=yes -o ConnectTimeout=30 -o
ServerAliveInterval=300 -l root -p 22 192.168.0.11 " DISPLAY=:0.0
/opt/weather_tester/weather_tester -c KHWD -r lax"
Exit status: 255
Duration: 5.99332118034

stdout:
Weather tester started ...
Not valid ...
5 seconds elapsed: closing ...
Weather tester closing without any error ...
stderr:

(weather_tester:13457): GWeather-WARNING **: Failed to get METAR data.

(weather_tester:13457): GWeather-WARNING **: Failed to get radar map image.

(weather_tester:13457): Gtk-CRITICAL **: gtk_image_new_from_animation: assertion 'GDK_IS_PIXBUF_ANIMATION (animation)' failed

(weather_tester:13457): Gtk-CRITICAL **: gtk_container_add: assertion 'GTK_IS_WIDGET (widget)' failed)

Removed Test Cases (Test Cases that are not present in current runs but are in baseline runs)

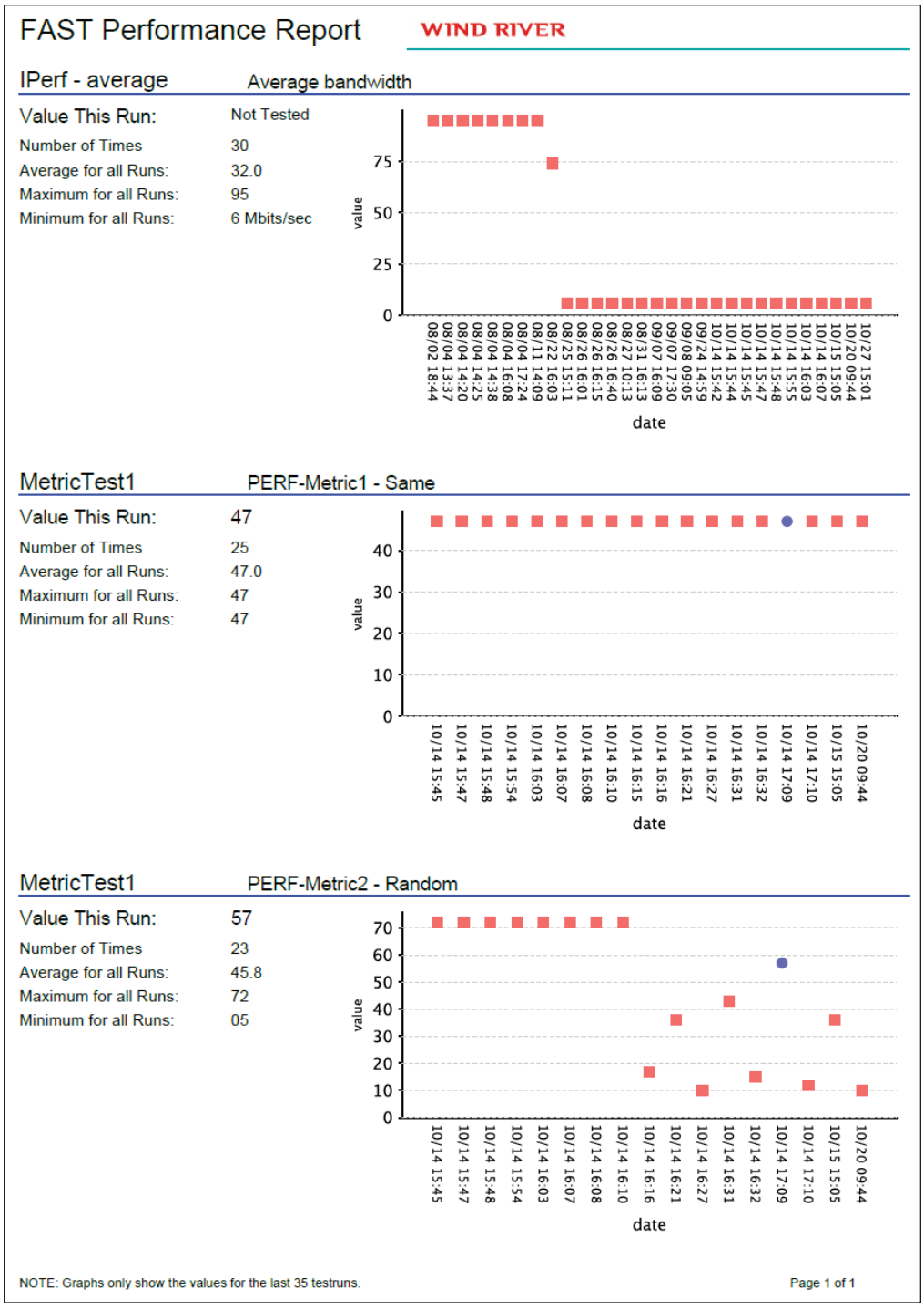
Group: Mobile Office	Script: Email SMTP SIM	Test Case: Email SMTP SIM	Result: passed
Group: Mobile Office	Script: Email IMAP	Test Case: Email IMAP	Result: passed
Group: Packages	Script: strace	Test Case: strace	Result: passed
Group: Packages	Script: sdl_image	Test Case: sdl_image	Result: passed

← *New Test Cases*

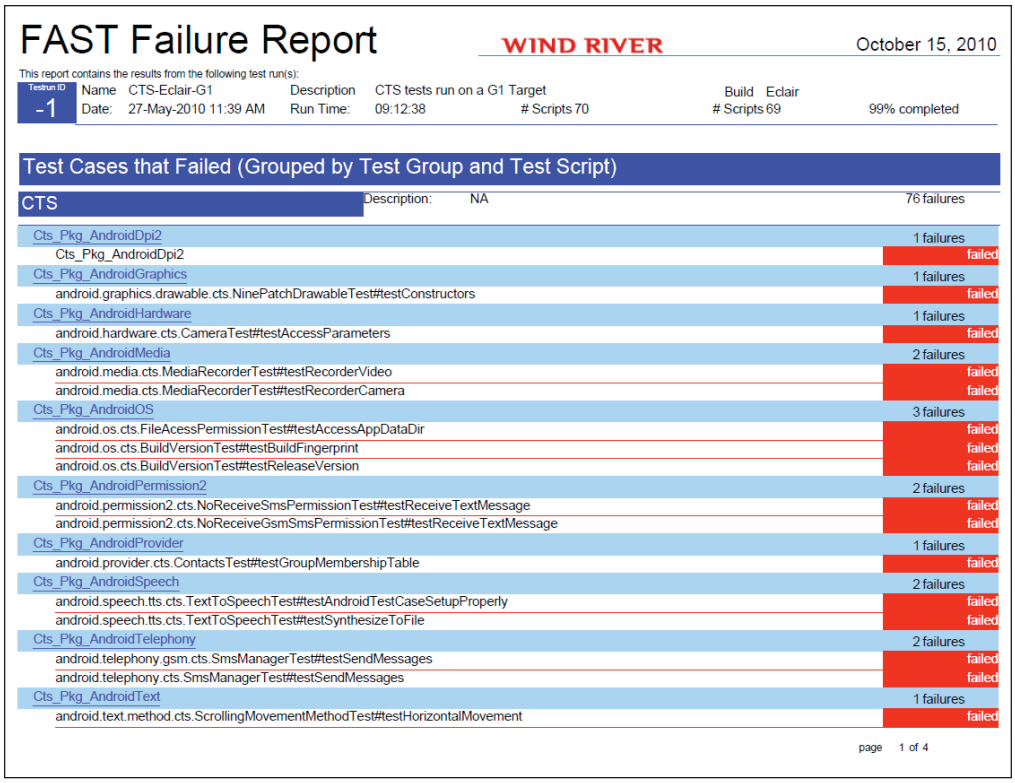
← *Pass/Fail Detail*

← *Removed Test Cases*

Appendix Figure 4: Sample comparative report—new test cases, pass/fail, and removed test cases



Appendix Figure 5: Wind River FAST performance report



FAST Failure Report

WIND RIVER

October 15, 2010

This report contains the results from the following test run(s):

Testrun ID	Name	Description	Build
-1	CTS-Eclair-G1	CTS tests run on a G1 Target	Eclair

Date: 27-May-2010 11:39 AM Run Time: 09:12:38 # Scripts 70 # Scripts 69 99% completed

Test Cases that Failed (Grouped by Test Group and Test Script)

CTS Description: NA 76 failures

Cts_Pkg_AndroidDpi2	1 failures
Cts_Pkg_AndroidDpi2	failed
Cts_Pkg_AndroidGraphics	1 failures
android.graphics.drawable.cts.NinePatchDrawableTest#testConstructors	failed
Cts_Pkg_AndroidHardware	1 failures
android.hardware.cts.CameraTest#testAccessParameters	failed
Cts_Pkg_AndroidMedia	2 failures
android.media.cts.MediaRecorderTest#testRecorderVideo	failed
android.media.cts.MediaRecorderTest#testRecorderCamera	failed
Cts_Pkg_AndroidOS	3 failures
android.os.cts.FileAcessPermissionTest#testAccessAppDataDir	failed
android.os.cts.BuildVersionTest#testBuildFingerprint	failed
android.os.cts.BuildVersionTest#testReleaseVersion	failed
Cts_Pkg_AndroidPermission2	2 failures
android.permission2.cts.NoReceiveSmsPermissionTest#testReceiveTextMessage	failed
android.permission2.cts.NoReceiveGsmSmsPermissionTest#testReceiveTextMessage	failed
Cts_Pkg_AndroidProvider	1 failures
android.provider.cts.ContactsTest#testGroupMembershipTable	failed
Cts_Pkg_AndroidSpeech	2 failures
android.speech.tts.cts.TextToSpeechTest#testAndroidTestCaseSetupProperly	failed
android.speech.tts.cts.TextToSpeechTest#testSynthesizeToFile	failed
Cts_Pkg_AndroidTelephony	2 failures
android.telephony.gsm.cts.SmsManagerTest#testSendMessages	failed
android.telephony.cts.SmsManagerTest#testSendMessages	failed
Cts_Pkg_AndroidText	1 failures
android.text.method.cts.ScrollingMovementMethodTest#testHorizontalMovement	failed

page 1 of 4

Appendix Figure 6: Wind River FAST failure report