

## Wind River Framework for Automated Software Testing (FAST)

Wind River Framework for Automated Software Testing (FAST) was developed to help customers overcome the challenges of testing large software distributions for mobile devices. 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 as many open source and existing test frameworks as possible so testing is comprehensive, without sacrificing time-to-market. Many open source test frameworks contain thousands of tests and many packages that come with their own test suites. 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).

### The Challenge of Testing

Testing very large open source software distributions for mobile devices is a tremendous challenge. There are typically dozens of subsystems containing thousands of packages 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
- Multidevice
- Peripheral
- Board support package (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 of generating and reporting results, executing the tests and tracking and evaluating the test results can especially complicate the 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.

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 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 MySQL 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 collected by FAST. FAST can be thought of as a meta-test framework that can run other test frameworks and individual tests.

From the FAST database, Wind River can create a variety of test reports. 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

The Challenge of Testing.....	1	Test Execution .....	3
The FAST Solution .....	2	Wind River Test Library .....	4
System Testing Plug-In .....	2	Reporting .....	5
		Appendix .....	6

## The FAST Solution

FAST offers significant advantages 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 the testing 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 the scripts.
- **Integrates other testing frameworks:** Any existing test framework or test application can be integrated into FAST using a FAST script. FAST 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 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:** 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 meth-

ods that perform common operations that are repeated in many tests (i.e., setting up network interfaces). When a new target is introduced, Wind River 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.

- **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 FAST database so FAST can track the test effort of the entire test organization.

## System Testing Plug-In

Wind River System Testing Plug-in is an application test mechanism that can simulate user interactions with user interfaces (UIs). Along with FAST, Wind River developed this technology to test large software distributions. System Testing Plug-in enables the automated testing of UIs and the user experience. This kind of testing has traditionally been manual because by its very nature it involves human interaction with the system UI and interaction among multiple applications on the device or even multiple devices.

System Testing Plug-in is a unique, powerful, and exciting technology that makes UI elements available for interrogation and manipulation from automated test scripts. It can easily repurpose existing tests for use with a new UI. For example, a calendar supports the same features across implementations, such as entering an appointment, listing today’s appointments, setting an appointment alarm, and so on. With System Testing Plug-in, tests for these common scenarios need only be written once for one UI and can be easily ported to a different UI by using simple abstraction layer code.

System Testing Plug-in also supports parallelizing tests, such as testing across multiple targets and automating use case and scenario-based testing. System Testing Plug-in offers significant advantages for application testing. It is able to simulate almost any user interaction with the target, including clicking widgets in the UI, dragging graphical elements, reading the contents of the UI elements, typing entries on a physical or virtual keyboard, and so forth.

System Testing Plug-in scripts are integrated into FAST. FAST has a standard wrapper that can be applied to

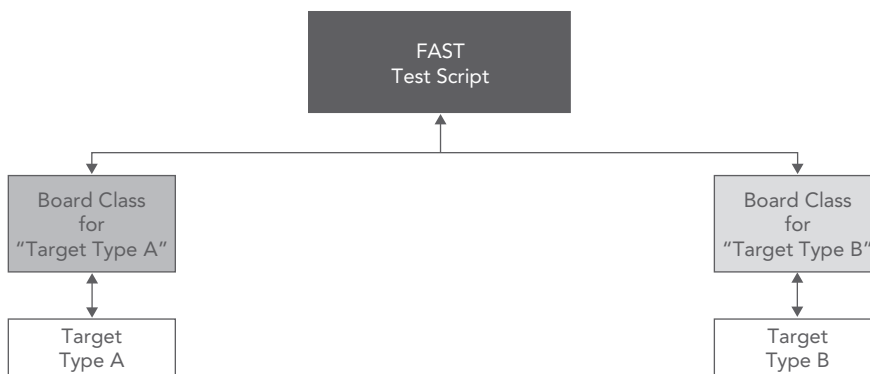


Figure 1: FAST test script configuration for multiple targets using board class

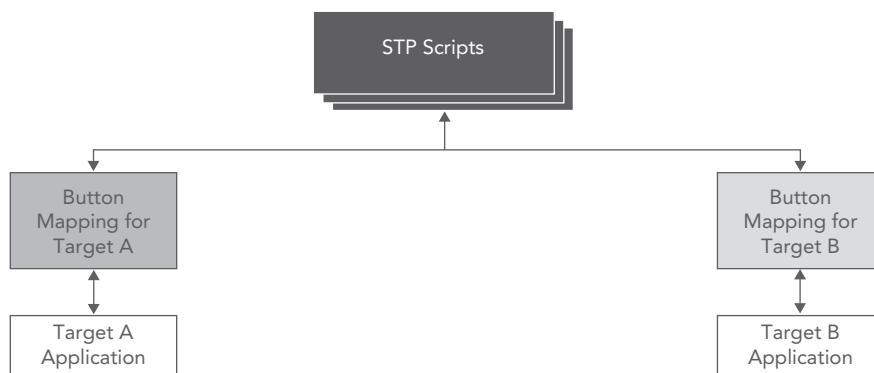


Figure 2: System Testing Plug-in script GUI mapping

any System Testing Plug-in test script. This allows these scripts to be executed, and their results can be tracked in the FAST database and reports. System Testing Plug-in scripts do not rely on the position of UI elements in an application. Instead, System Testing Plug-in uses the accessibility features of widget frameworks to know the names of the widgets to interact with. The widget creates a map of the buttons so the test scripts can refer to the buttons by name rather than screen location. Most GUI testing applications specify actions such as "Click at position 10,10." A statement by System Testing Plug-in would be more like "Click the button mapped as 'OK.'"

This means that no matter how the application changes its layout, the System Testing Plug-in script can still execute the test. The same System Testing Plug-in script works on multiple implementations of the same programs (i.e., calculators, calendars, dialers, browsers) with different interfaces. However, the same script can be executed on all of them, with only minor changes to the button name maps.

System Testing Plug-in scripts can be combined into one. This enables the testing of multiple applications running at the same time, for example, when a user is listening to music and reading email and a call comes in. Interleaving existing test scripts allows Wind River to easily create new test scenarios.

## Test Execution

Configuring a target for testing can be easy and straightforward. FAST uses a board class to specify communication and control for the target. With a 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.

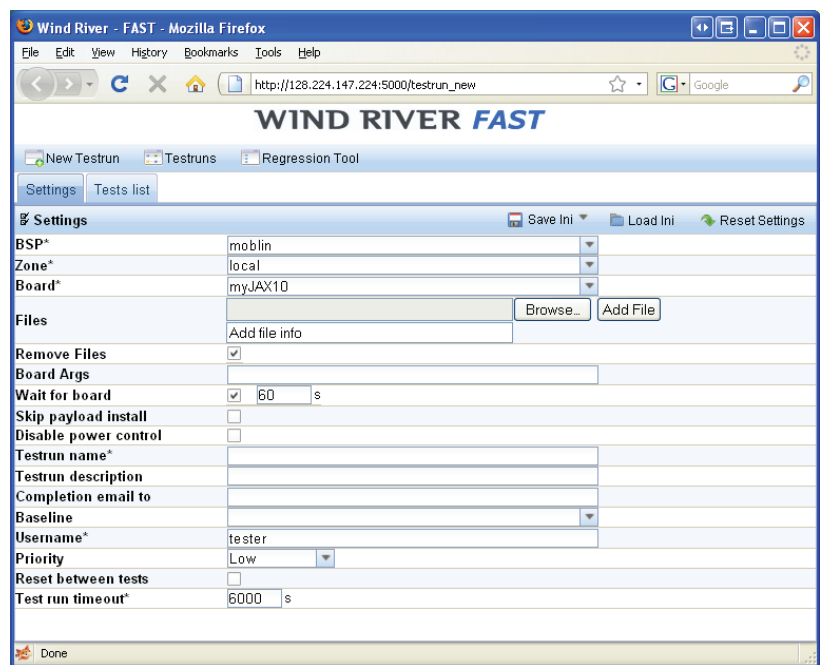


Figure 3: FAST user interface

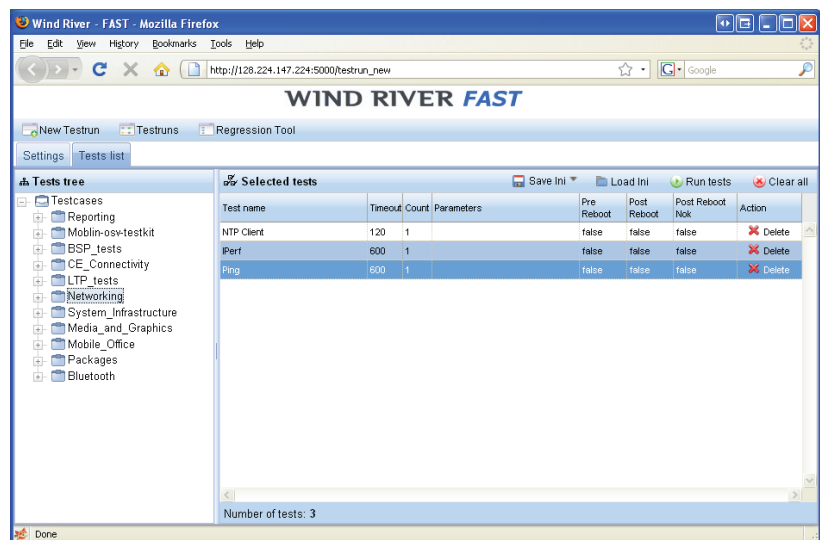


Figure 4: FAST test lists

- 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 right-clicking them. Each test may then be configured, including default time-out, count, parameters, and reboot options (reboot before, after, or after when the test does not pass). Once the test runs are configured, each test's progress is displayed and a test's details are available, including the start and end dates, any test cases it ran, their status (passed/failed), and any artifacts, such as log files, or key values (performance metrics) that are associated with the test case.

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's 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, design 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 and, instead of running them, 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.

### Wind River Test Library

Wind River has a large suite of tests created for our 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:** FAST includes the latest version of the compliance suites for Android Compatibility Test Suite (CTS), Moblin (OSV, ISV, and Application Checker), and GENIVI.
- **System framework:** There is a complete set of system framework tests specifically for Wind River Platform for Moblin.
- **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:** Linux Test Project (LTP), IOzone, iPerf, and Moblin Compliance Tool Suite 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 Moblin.
- **BSPs:** Tests include IOzone for board support packages (BSPs).
- **Stress/stability/soak:** 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.
- **UI:** Automated test suites for Wind River Platform for Moblin UIs are created using the System Testing Plug-in framework. System Testing Plug-in tests are run by FAST in the same manner as any other test framework.

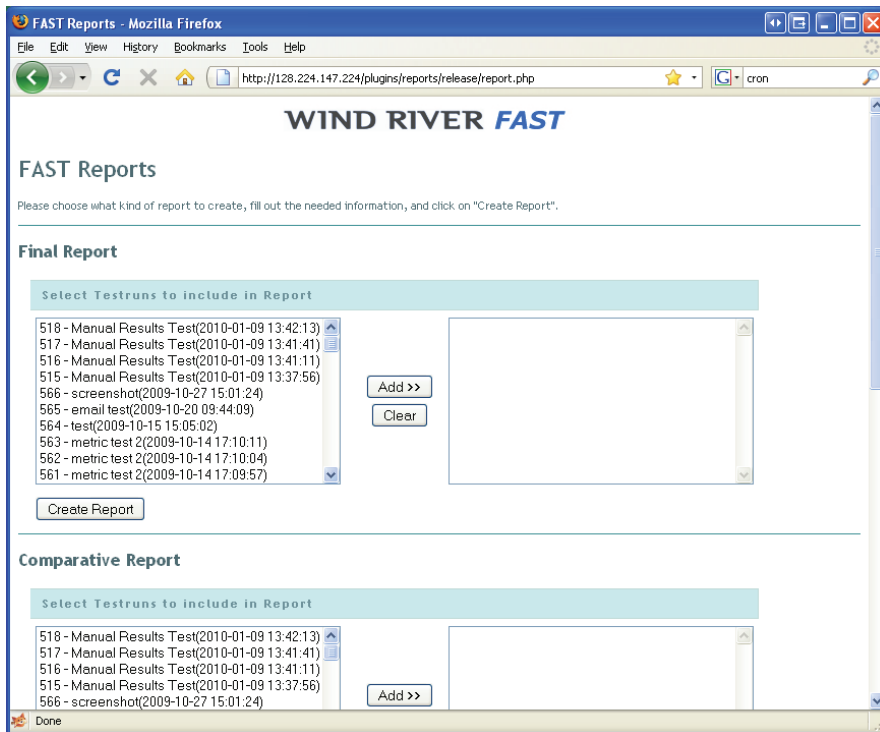


Figure 5: FAST test report window

## Reporting

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

There are four 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.
- **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

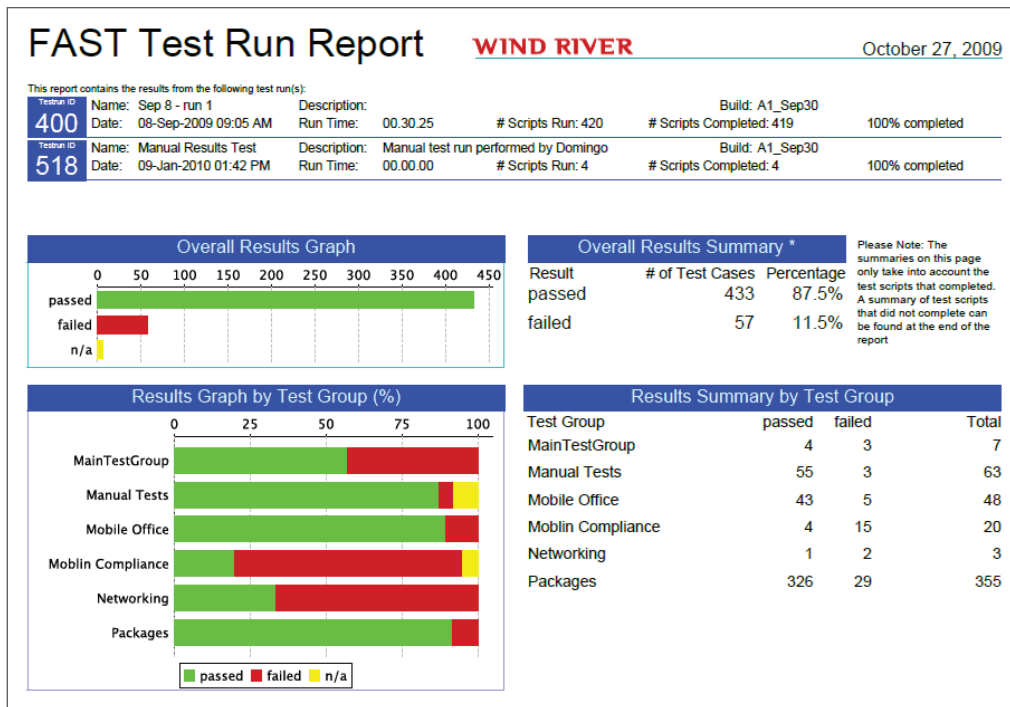
(e.g., showing the last 30 test runs and highlighting the current test run). This allows developers and managers to see whether a change in the software on a certain date caused a degradation (or improvement) of a particular performance metric. See Appendix Figure 5 for a sample report.

- **Requirements tracking:** The requirements tracking report shows all the requirements in the FAST database, the last time they were tested, and the last test result. This is primarily to help QA managers decide whether to ship a release because they can quickly see the status of all the requirements testing. See Appendix Figure 6 for a sample report.

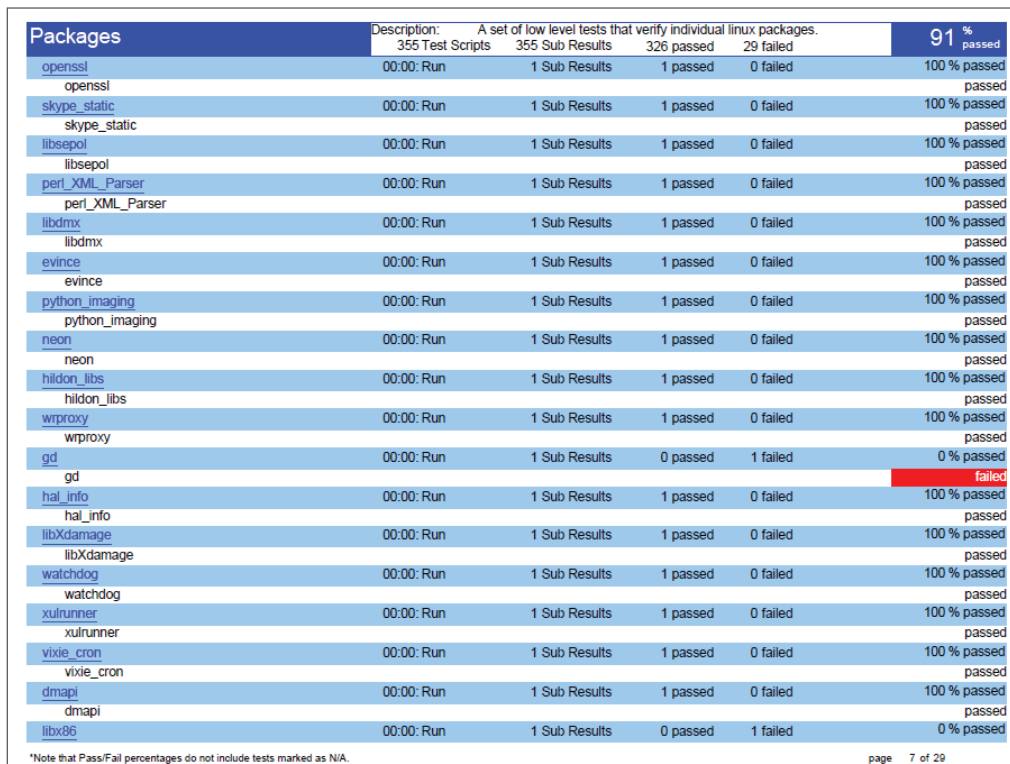
Adding new reports is relatively easy. Since all the information is stored in a MySQL database, you can use your preferred reporting engine to generate a report or engage Wind River to create new reports that meet your needs. FAST uses the open source JasperReports to create and generate the reports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

## Appendix

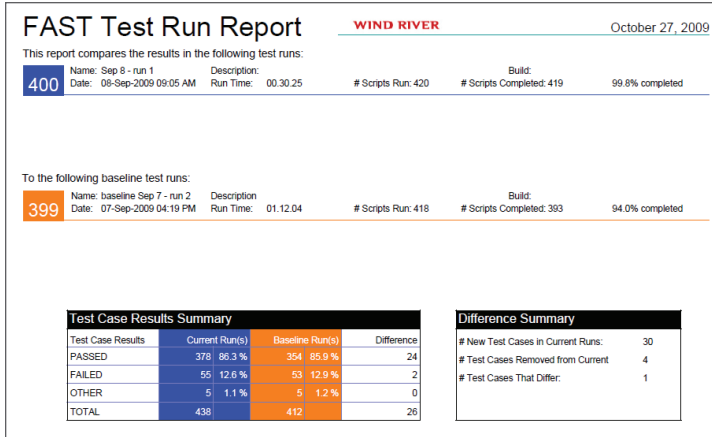
The following images are samples of the types of reports that can be generated with the FAST system. In addition to these reports, others can be created.



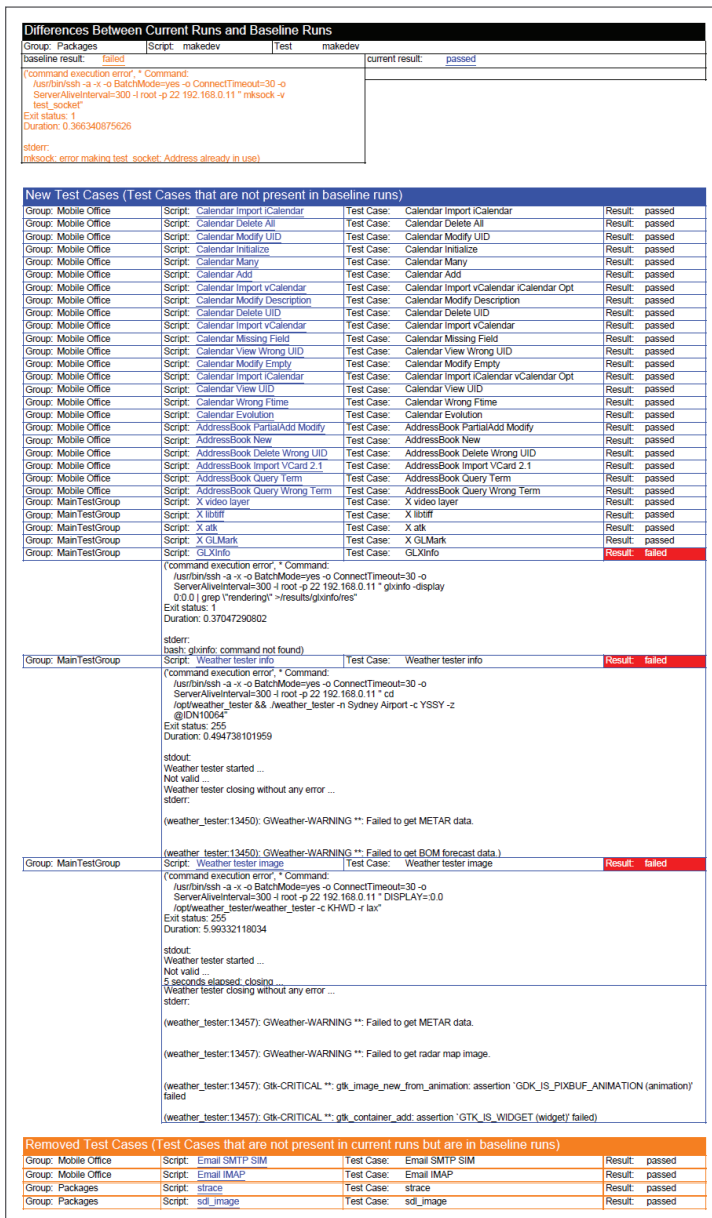
Appendix Figure 1: Sample final report—summary



Appendix Figure 2: Sample final report—detail



Appendix Figure 3: Sample comparative report—summary



← New Test Cases

← Pass/Fail Detail

← Removed Test Cases

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

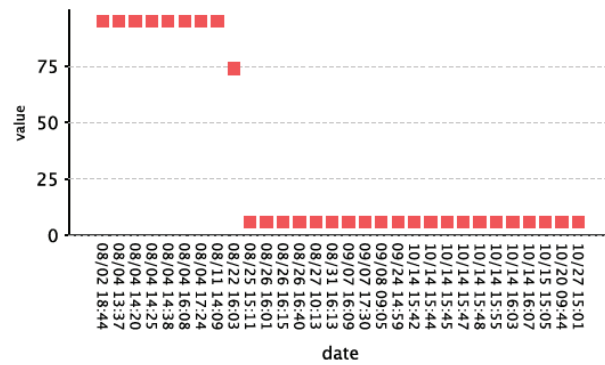
# FAST Performance Report

WIND RIVER

## IPerf - average

### Average bandwidth

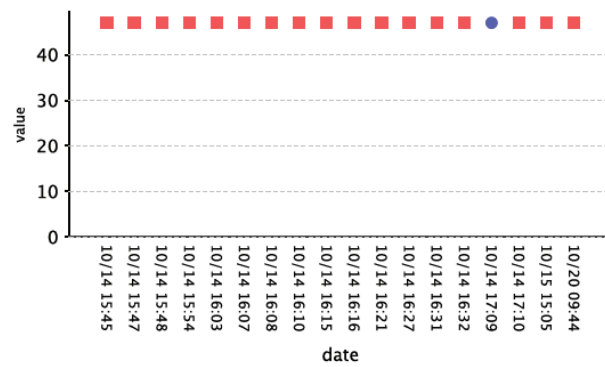
Value This Run: Not Tested  
 Number of Times: 30  
 Average for all Runs: 32.0  
 Maximum for all Runs: 95  
 Minimum for all Runs: 6 Mbits/sec



## MetricTest1

### PERF-Metric1 - Same

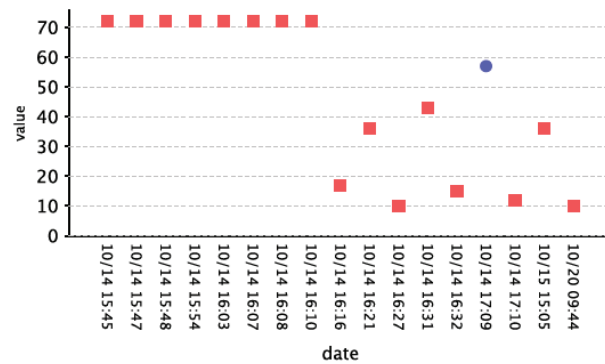
Value This Run: 47  
 Number of Times: 25  
 Average for all Runs: 47.0  
 Maximum for all Runs: 47  
 Minimum for all Runs: 47



## MetricTest1

### PERF-Metric2 - Random

Value This Run: 57  
 Number of Times: 23  
 Average for all Runs: 45.8  
 Maximum for all Runs: 72  
 Minimum for all Runs: 05



NOTE: Graphs only show the values for the last 35 testruns.

Page 1 of 1

Appendix Figure 5: FAST performance report

# FAST Requirement Tracking Report

This report contains test results for requirement testing that was performed during the past 60 days. Elements highlighted in bright colors have been tested in the past 10 days. Elements in lighter colors have been tested in the past 60 days (but not the past 10 days)

### Summary of Results

Total Number of Requirements:	815		
Number of Requirements Tested:	5	0.61 %	
Number of Requirements that Passed:	3	0.37 %	
Number of Requirements that Failed:	2	0.25 %	

Req.	Description	Last Tested On	Test by Test Script/Test Case	Test Run	Tested on Build	Last Result
1.1.1.1.1	System service maintenance	20-Oct	IPerf - average bandwidth	565		failed
1.1.1.1.2	Application watchdog	20-Oct	Ping	565		passed
1.1.1.1.3	Application termination - User	08-Sep	xinit	400		passed
1.1.1.1.4	Application termination - User initiated	25-Sep	X atk	490		passed
1.1.1.1.5	Application termination - System	16-Sep	Performance-Power-Test - [ Checked requirement:	421		passed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		passed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		passed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		passed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		failed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		failed
		16-Sep	Performance-Power-Test - [ Checked requirement:	421		failed
1.1.1.1.6	Application event notification					not tested
1.1.1.1.7	Application Navigation					not tested
1.1.1.1.8	Application launching other					not tested
1.1.1.1.9	Application Interruptibility					not tested
1.1.1.1.10	Non-responsive Application					not tested
1.1.1.1.11	Application request: TERMINATE					not tested
1.1.1.1.12	Application request: SLEEP					not tested
1.1.1.1.13	Application state restore					not tested
1.1.1.2.1	Inter application communication					not tested

Appendix Figure 6: Sample requirements tracking report