

Linux ACPI Test Framework

Revision 0.1

Last Save Date: 5/16/2006

Last Changed By: Ling Yu

Key Contributors: Ling Yu

Revision History

Revision	Date	Author	Reason for Changes
0.1	5/16/06	Ling Yu	Initial Revision

-

Introduction

ACPI subsystem consists of several separate features, so tester maybe hope to execute and get testing results independently. But consider the case reusability, easy to understand and easy to run, we hope the doc, code, and test output have some standardized structure. At same time the test case development flexibility is also need to keep. This document tries to define a simple, flexible, standardized and easy to conformed guide to integrate the test cases to ACPI test framework and also defines requirements for test development to ensure easy setup and run tests and analyze their results.

Copyright notice is at the discretion of a test developer because of possible sensitive code. Test Framework code grants GNU GPL.

-

Test Framework Structure

Directory Structure

```

acpi-test/
|-- ABAT/
| |-- TestSpec.txt
| |-- README
| |-- Makefile
| |-- runall.sh
| `-- <test suite files or folders>
|-- function/

```

```

| |-- TestSpec.txt
| |-- README
| |-- automatic/
| | |-- Makefile
| | |-- runall.sh
| | `-- <test suite files or folders>
| `-- manual/
| | |-- README
| | |-- Makefile
| | |-- runall.sh
| | `-- <test suite files or folders>
|-- performance/
| |-- TestSpec.txt
| |-- README
| |-- Makefile
| |-- runall.sh
| `-- <test suite files or folders>
|-- stress/
| |-- TestSpec.txt
| |-- README
| |-- Makefile
| |-- runall.sh
| `-- <test suite files or folders>
|-- regression/
| |-- TestSpec.txt
| |-- README
| |-- Makefile
| |-- runall.sh
| `-- <test suite files or folders>
`-- 3rd_party/
    |-- README
    .

```

File Definitions

This test framework consists of several directories for different test types: ABAT, function, performance, stress, and regression which all have following absolutely necessary files:

TestSpec.txt - A document about test specification and test strategy. This document shall describe what test verifies (specifications, test assertions) and algorithms used for that (test strategy). This document should be plain text.

README - It can describe the feature description, test environment requirement, how to run the test cases and some known issues.

Makefile - Makefile for make, the top level Makefile of every test type should be the only entry to build all test cases, it should invoke other Makefiles in sub-directories, and doesn't need user enter sub-dir to build one by one. Basically Makefile shall follow GNU make Manual.

runall.sh - The shell script to execute testing steps, which should be the only entry to finish full test cases execution one by one.

This file has requirements listed below:

1. Copyright, License, and Authors are required on the top of the file.
2. runall.sh should output execution result information as expected result output format.
3. runall.sh can output log information, which will help user locate the defects.
4. runall.sh can have command line parameters for debugging purposes.
5. runall.sh can invoke other scripts or tools to help itself to finish test steps in the current directory.

<test suite files or folders> - test case files or directories, they can be C, C++, shell, and other data files, and also can be organized as directories.

1. The files or directories should be well organized, user can easy to know which file or dir is for which test case. Recommend to use test case id as file or directory name.
2. Test case should be automatic as possible as it can.
3. Test cases are recommended to be able to run separately.
4. Copyright, License, and Authors are required on the top of every file. Test case should have some descriptions, which can make the test case easy to read.
5. Test case should use the one or more values in TEST_RETURN_CODE string in its results output, these return value meaning should be full considered, for example, if a case need run on specific hardware/OS, it should detect these circumstance, and can return NOTSUPPORT for the mismatch.
6. Tests shall have assertions which they verify. These assertions (possibly only one for the test) shall be listed in TestSpec.txt document.

Test framework also has a directory 3rd-party to place all external tools which might be invoked by test suite. A README file describes the usage of 3rd-party directory.

Return Code Definitions

TEST_RETURN_CODE	Values	Meanings
PASS	0	Test case passes.
FAIL	1	Test case fails because the implementation doesn't comply with the feature description.
NOTSUPPORT	2	Test case can't run successfully because of the lack of some specific hardware on the current platform or current platform isn't the feature defined platform.
UNRESOLVED	3	The test case could not get the expected result because of other reason except the unsatisfied feature, such as wrong configuration, missing files, and etc.
UNTESTED	4	Used when a feature does not have a test associated with it because: <ul style="list-style-type: none"> - The test is just a stub and doesn't do anything - The test is only partially complete and can't really finish the test - The test is complete in some cases, but certain things can happen that leave the test incomplete. When these happen, it's UNTESTED.

Result Format

Following is the format of the test suite output

* Test ACPI <The feature tested by this test suite> *

<outputs from every test case in this test suite>

Test Summary:

TOTAL: <total_case_number>

PASS: <passed_case_number>

FAIL: <failed_case_number>

NOTSUPPORT: <number of non-passed case because of unsupported platform>

UNRESOLVED: <number of non-passed case number because of other reason>

UNTESTED: <number of unready test case>
