



Robot Framework integration on LAVA

1	Contents	
2	Introduction	2
3	Robot framework architecture overview	3
4	Test Data	3
5	Robot Framework	4
6	Test Libraries & Test Tools	4
7	System Under Test	4
8	Robot Framework on LAVA	4
9	Integration approach	5
10	Test execution workflow	6
11	Framework operation	8
12	Impact analysis on Apertis distribution	9
13	Infrastructure	9
14	Development environment	10
15	Test development	10
16	Testing	10
17	Summary	10

18 Introduction

19 The aim of this document is to provide a suitable solution for the integration of
20 Robot Framework into the LAVA automated test infrastructure. LAVA doesn't
21 currently support triggering or executing Robot Framework test suites. Thanks
22 to this integration the coverage test can be extended to cover different test areas
23 by adding additional customized libraries and toolchains.

24 **LAVA**¹ (Linaro Automation and Validation Architecture) is a continuous inte-
25 gration system for deploying operating systems onto physical and virtual hard-
26 ware for running tests. Tests can be simple boot testing, bootloader testing and
27 system level testing, although extra hardware may be required for some sys-
28 tem tests. Results are tracked over time and data can be exported for further
29 analysis.

30 **Robot Framework**² is open source software released under the Apache License
31 2.0 and is a simple, yet powerful and easily extensible tool which utilizes the
32 keyword driven testing approach. It uses a tabular syntax which enables creating
33 test cases in a uniform way. All these features ensure that Robot Framework can

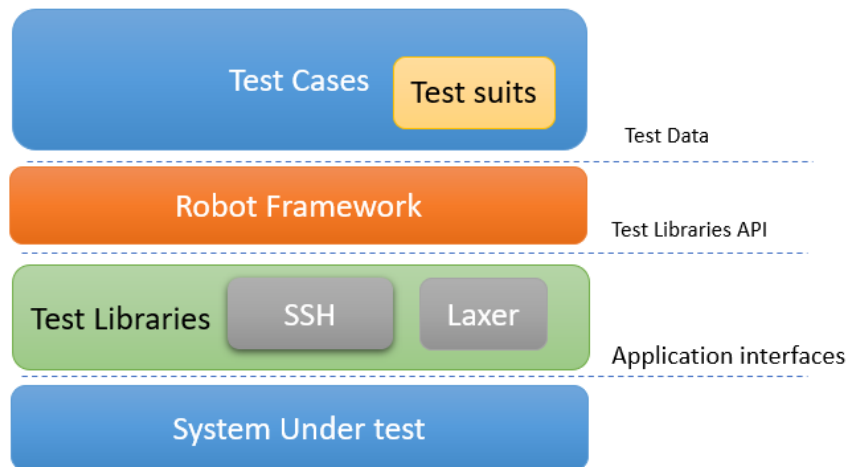
¹<https://www.lavasoftware.org/>

²<https://robotframework.org/>

34 be quickly used to automate test cases. The best benefit with Robot Framework
 35 for the users is that there is no need for using any sort of programming language
 36 for implementing and running tests.

37 Integrating Robot Framework on LAVA infrastructure adds additional benefits
 38 of Robotic Process Automation (RPA), [ATDD](https://en.wikipedia.org/wiki/Acceptance_test-driven_development)³ (Acceptance test-driven devel-
 39 opment) and also allows the use a wide range of open source libraries developed
 40 for automation testing.

41 Robot framework architecture overview



42

43 Test Data

44 The Robot framework has a layered architecture. The top layer is the simple,
 45 powerful, and extensible keyword-driven descriptive language for testing and
 46 automation. This language resembles a natural language, is quick to develop, is
 47 easy to reuse, and is easy to extend.

48 Test data, the first layer of the Robot framework is in a tabular format. Since
 49 the data is in a tabular format, maintaining the data is very easy. This test
 50 data is the input to Robot Framework, once it is received, it is processed and
 51 on execution reports and logs are generated. The report is in HTML and XML
 52 format and offers detailed information about every line that is executed as a
 53 part of the test case.

³https://en.wikipedia.org/wiki/Acceptance_test-driven_development

54 **Robot Framework**

55 Robot Framework is a generic, application and technology independent frame-
56 work. The primary advantage of the Robot framework is that it is agnostic of
57 the device under test (DUT). The interaction with the layers below the frame-
58 work can be done using the libraries built-in or user-created that make use of
59 application interfaces.

60 **Test Libraries & Test Tools**

61 A library in a Robot Framework terminology, extends the Robot Framework
62 language with new keywords, and provides the implementation for these new
63 keywords. Each Robot Framework library acts as glue between the high level
64 language and low level details of the item being tested, or of the environment
65 in which the item to be tested is present.

66 Robot Framework has a rich set of built-in libraries e.g HTTP, FTP, SSH, and
67 XML, as well as user interface and databases.

68 **System Under Test**

69 This is the actual DUT on which the testing activity is performed. It could
70 either be a library or an app. Libraries act as an interface between the Robot
71 Framework and the system under test. Hence, there is no way through which
72 the framework can directly talk to the system under test. The Robot Framework
73 supports various file formats namely HTML, TSV (Tab Separated Values), reST
74 (Restructured Text), and Plain text. As per the official documentation of Robot
75 framework, the plain text format is recommended.

76 **Robot Framework on LAVA**

77 There are two main constraints on automated tests setup on LAVA, the asyn-
78 chronous way of updating results and user not having control over the job once
79 it is submitted. Developers and CI pipeline can both submit jobs to LAVA,
80 but they cannot interact with a job while it is running. The LAVA workflow
81 defines the process of submitting a job, waiting for the job to be selected for
82 execution, waiting for the job to complete its execution, and downloading of
83 the test results.

84 Considering the above constraints and the wide range of desired test areas, in-
85 tegrating the Robot Framework with LAVA provides more chances to automate
86 complex tests by making use of its open source libraries.

87 The Robot Framework can add value to Apertis, but adding it to Apertis will
88 involve developing and/or modifying Robot Framework libraries and developing
89 a run-time compatibility layer for LAVA. The run-time compatibility layer for
90 LAVA has two major objectives: keep testing environments as close as possible

91 to production environments, and to adapt the execution of Robot Framework
92 tests to suit the LAVA constraints.

93 Integration approach

94 A LAVA instance consists of two primary components **masters** and **workers**
95 works as a [master-slave model]([https://en.wikipedia.org/wiki/Master%E2%80%93slave_\(technology\)](https://en.wikipedia.org/wiki/Master%E2%80%93slave_(technology))), where the master controls one or more devices and
96 serves as their communication hub.
97

98 The worker is responsible for running the `lava-worker` daemon to start and mon-
99 itor test jobs running on the dispatcher. Each master has a worker installed
100 by default and additional workers can be added on separate machines, known
101 as remote workers. The admin decides how many devices are assign to each
102 worker. In large instances, it is common for all devices to be assigned to remote
103 workers to manage the load.

104 The simplest possible configuration is to run the master and worker components
105 on a single machine, but for larger instances it can also be configured to support
106 multiple workers controlling a larger number of attached devices in a [multi node](#)⁴
107 model.

108 There are three possible approaches available to integrate Robot Framework on
109 LAVA:

- 110 1. Integrating a standalone development setup inside the dispatcher.
- 111 2. Introduce a different device type to enable standalone docker with Robot
112 Framework instance
- 113 3. Introducing a `test:docker` container to run a Robot Framework instance

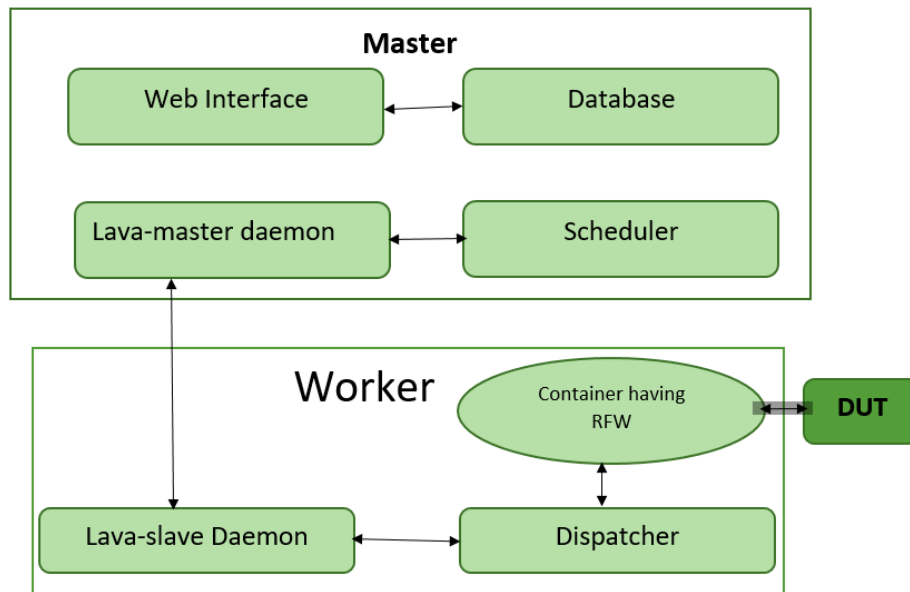
114 The first approach consists of creating a QEMU emulator with the Apertis SDK
115 image and installing Robot Framework. In this approach, a user can run all
116 automated tests related to the system and toolchain. Mainly this approach
117 is to test the headless functionality which are part of development activities.
118 However, running DUT related tests such as Fixed Function or HMI images is
119 not feasible, therefore this approach is not meeting all the use cases of production
120 readiness.

121 The second approach consists of creating a separate device type on the LAVA
122 instance which contains a test Docker container where robot framework runs
123 under the worker context. This setup provides the benefits of isolation and
124 security, but it includes the additional effort of maintaining a different device
125 type on LAVA. Test suites would need to specifically mention the device-type
126 along with the architecture to run the tests on this instance. An additional
127 advantage is that each test suite execution will be run on an independent Docker
128 container making parallel execution possible for different jobs, this approach

⁴<https://docs.lavasoftware.org/lava/multinode.html>

129 increases the isolation of running the test suites and handling the report, but
 130 increases memory overhead if too many devices are attached and simultaneously
 131 running.

132 The third approach consists of introducing a `test:docker` login mechanism on
 133 the LAVA instance. This approach is completely developed and open sourced
 134 by Apertis team. Here, the job description should define the docker part by
 135 providing valid credential to pull the docker to run on dispatcher instance and
 136 execute the test steps mentioned on the test suits.



137

138 After evaluating the above three approaches, the third approach is the best
 139 fit for integrating Robot Framework on LAVA as it provides relatively easy
 140 maintenance and feature customization.

141 Test execution workflow

142 Test cases and test suites can be developed using the developers editor of choice
 143 and these tests can be run manually on the Apertis SDK, or configured to be
 144 run on LAVA.

145 Following workflow provide the steps to integrate Robot Framework tests and
 146 to be run on LAVA.

147 Create a common group for all the Robot Framework tests running on LAVA
 148 under `apertis-test-cases`⁵/lava called `group-robot-tpl.yaml` as follows:

⁵<https://gitlab.apertis.org/tests/apertis-test-cases/>

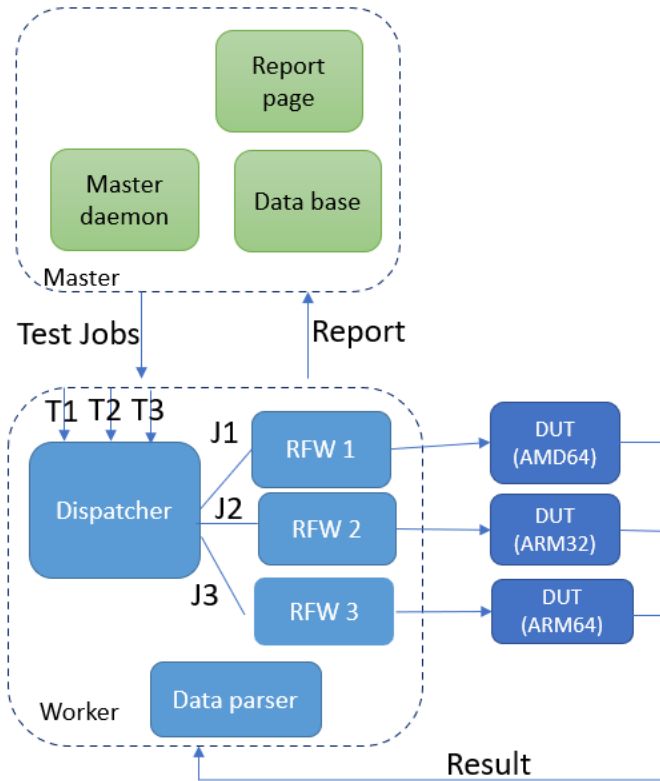
```

149 - test:
150     timeout:
151         minutes: 180
152     namespace: rfw-test
153     name: {{group}}-tests
154     docker:
155         image: "docker://registry.gitlab.apertis.org/infrastructure/apertis-
156 docker-images/{{release_version}}-rfw-docker:latest"
157     login:
158         registry: "registry.gitlab.apertis.org"
159         user: "gitlab-ci-token"
160         password: "{{ ' '{{job.CI_JOB_TOKEN}}' }}"
161     definitions:
162     - repository: https://gitlab-ci-token:{{ ' '{{job.CI_JOB_TOKEN}}' }}@gitlab.apertis.org/tests/apertis-
163 test-cases.git
164         branch: 'apertis/v2023'
165         history: False
166         from: git
167         name: robot-connman-tests
168         path: test-cases/robot-connman.yaml
169         parameters:
170             DEVICE_IP: "${lava-target-ip}"
171             ROBOT_FRAMEWORK_CONNMAN_URL: |-
172             https://gitlab-ci-token:{{ ' '{{job.CI_JOB_TOKEN}}' }}@gitlab.apertis.org/tests/robotframework.git

```

173 This template provides the basic information and credentials for fetching and
174 running the Robot Framework with Docker in LAVA. Tests can be added util-
175 ising this template.

176 Framework operation



177

178 The above diagram shows the basic workflow of LAVA jobs using Robot Frame-
 179 work. A job will be created on the master daemon which specifies a suite of tests
 180 (T1 to T3), which DUT that the tests will be run on and the Apertis release
 181 plus image type which they should be run against.

182 When it is time for it to run, the master daemon passes the job to the dispatcher
 183 with the required DUT. The dispatcher will launch a Robot Framework docker
 184 instance (J1 to J3) which will connect to the required DUT using the SCP
 185 and SSH protocols to copy required files to and from the DUT and execute
 186 commands on it, rather than copying the entire test suite and Robot Framework
 187 to the DUT and executing it from there. This has the advantage that minimal
 188 alterations will be made to the image that is being tested. The required test
 189 suite will be executed from within its docker environment, with each job running
 190 in its own fresh isolated docker environment, ensuring that it is not affected by
 191 content left from previous jobs.

192 Once the test execution is completed, Robot Framework will generate a test
193 report and a number of logs which will be copied from the docker instance and
194 shared with the LAVA server. Once this is done the docker instance will be
195 cleaned up. A summary of the testing results and the test reports/logs will be
196 made available via the [dashboard](#)⁶.

197 It is likely that Robot Framework tests will have dependencies which are required
198 for the tests to run correctly. Where these dependencies form part of the test
199 harness in the docker instance (for example, libraries to drive peripherals such
200 as a touch simulator to simulate touch events for HMI tests), these should form
201 part of the docker definition and installed from the Apertis repositories when the
202 docker instance is created. Where these dependencies need to be available on the
203 DUT, they either need to be preinstalled as part of the image or are required to
204 be added to the image during testing (such as by applying an overlay on OSTree
205 based images).

206 When run, the Robot Framework generates three files in its output directory:

- 207 • `output.xml`: An XML formatted record of the test execution, including
208 data such as test names, statuses, messages, and tags.
- 209 • `log.html`: A detailed HTML formatted log of your test execution, which
210 includes timestamps, keywords, arguments, screenshots, and console out-
211 put.
- 212 • `report.html`: An HTML formatted summary of your test execution, which
213 shows the overall statistics, test cases run and errors raised.

214 Currently the LAVA server is not processing any of these Robot Framework test
215 reports, it only tracks the test status. We plan to add a data parser and provide
216 the parsed data to LAVA. The Robot Framework reports will also be stored and
217 a link provided to them from the LAVA report.

218 The Robot Framework only generates the status report at the end of test execu-
219 tion. To allow for more real time tracking of the testing, the Robot Framework
220 provides a listener mechanism which can be used to provide fine grain moni-
221 toring of each individual tests execution. A listener script should be written
222 to interface between the Robot Framework and LAVA and made available as
223 part of the main test scripts. This integration will provide greater integration
224 between the Robot Framework and the existing LAVA infrastructure and will
225 be very beneficial when debugging failing tests.

226 Impact analysis on Apertis distribution

227 Infrastructure

228 Integrating Robot Framework on existing Apertis infrastructure will requires
229 the following changes :

⁶<https://qa.apertis.org/>

- 230 • Improvement of LAVA workers to enable them to run docker instances.
- 231 • Configure pipelines to ensure the capture of all the Robot Framework
- 232 results.
- 233 • Extend the [Apertis test report site](https://qa.apertis.org)⁷ to show the Robot Framework results

234 Development environment

235 The current development environment integrates Robot Framework with all its
236 standard libraries, along with the `SSH` library as part of the SDK distribution.
237 Using the Apertis SDK a developer can write Robot Framework test cases to
238 run on the SDK and DUTs running Fixed Function or HMI images.

239 Test development

- 240 • Impact on Apertis development is that we have start developing new test
- 241 suites for robot framework.
- 242 • Start developing new yaml files which helps in executing the robot test
- 243 suites from containers
- 244 • Apertis tests needs to rewrite the existing LAVA test job to execute the
- 245 robot test suites

246 Testing

- 247 • With approaches mentioned above we can keep the existing scripts as they
- 248 are and start executing tests defined with the new Robot Framework test
- 249 suites which will help to improve the test coverage.

250 Summary

251 The integration of the Robot Framework into the Apertis, enables tests to be
252 written using this simple, yet powerful and easily extensible testing framework
253 for Apertis whilst also taking advantage to the many features provided by the
254 Apertis test framework:

- 255 • End to End workflow
 - 256 – LAVA pipelines can take care of all test stages: control of board
 - 257 power; flashing & booting images; loading tests; running the tests;
 - 258 and reporting test results.
 - 259 – Tests can be run in parallel on different targets, reducing test cycle
 - 260 time, when compared with manual tests run by a limited test team.
 - 261 – Devices can be reserved for specific tests or specific users.
- 262 • Internet facing Web Service

⁷<https://qa.apertis.org>

- 263 – One centrally hosted and maintained front end service which can be
- 264 utilised by multiple teams, each providing worker systems connected
- 265 to their specific DUTs.
- 266 – Internet connectivity enables collaboration with external partners.
- 267 – Remote management of devices. Many maintenance tasks can be
- 268 completed without physical access to DUTs.
- 269 – Remote access to users for running tests, viewing logs & reports.
- 270 – Role based access permissions allowing granular control over access
- 271 to functionality and specific DUTs.
- 272 – Access to mail notifications and alerts.
- 273 • Sharing of physical assets between multiple software projects
- 274 – DUTs can be shared between multiple projects, such as teams fo-
- 275 cusing on different operating systems or teams focusing on different
- 276 software stacks within a larger operating system can schedule jobs to
- 277 be run on shared hardware, reducing the number of physical devices
- 278 needed for testing across an organisation
- 279 • Continuous testing
- 280 – Periodic triggering of test runs against DUTs as part of continuous
- 281 testing to ensure acceptable operation as system evolves.
- 282 – Reuse of common tests between integration and continuous testing
- 283 regimes avoiding duplication of effort.
- 284 • Handles inconsistency.
- 285 – Retry mechanisms mitigate against test failures due to temporary
- 286 failure of ancillary operations, such as transient download failures.
- 287 • Inbuilt reporting dashboard
- 288 – Insight full metrics available in the inbuilt dashboard.
- 289 – Access to full test reports and test definitions.
- 290 – Access to test logs including timing metrics.