PLUMA

Automated testing framework
to deliver and maintain industrial-grade devices

Full-stack testing | *Automated testing* | *Non-intrusive*

One-stop-shop for embedded testing

Test management

- Library of pre-built test actions ✓
- Library of pre-built resources ✓
- Extensible with Python code ✓

Continuous testing

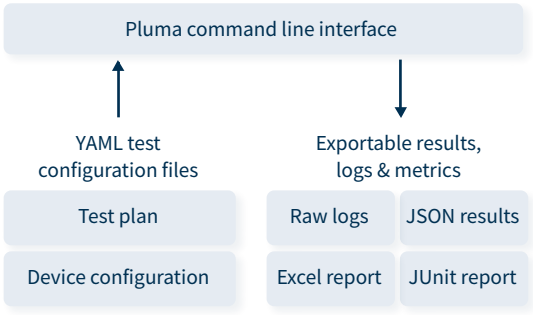
- CI integration (Gitlab, Jenkins, Github...) ✓
- Deployment to the target system (Linux NFS, JTAG, file transfer, etc.) ✓

Reporting & metrics

- Raw logs ✓
- JSON results ✓
- Excel, JUnit & XRay integrations ✓
- Performance metrics ✓

Pluma command line interface help build & automate all your test scenarios

- Endurance testing & technical stress
- Performance measurement
- Data connectivity
- Application features
- Protocol testing (BLE, CAN, Modbus)...



Compare Pluma and Robot Framework

Compare

The Embedded Kit

contact@theembeddedkit.io

Try Pluma for free

Built-in actions

Console commands

login: Attempt to login on the active console

shutdown: Send the shutdown command specified from shutdown_command in system's configuration

wait_for_pattern: Wait for a specific pattern on the currently active console

set_console: Set the default device console to use

close_console: Attempt to close the specified or currently active console

send: Send data or commands on a console (can be a serial interface)

run_on_device: Run one or more commands on the device; Commands are expected to run from a POSIX shell

run_on_host: Run one or more commands on the host runner

Variables & expressions

set_variables: (or setvar:) Set one or more variables at runtime

check: Check that the expression/value is True, or matches the expected attribute, if provided

match: Check that the text matches with the regular expression provided

match_any_line: Check that any line matches a regex in a text/output

Measurements & logging

log: Log a message in the standard output, and global log file

metrics: Generate one or multiple metrics, from list(s) of numeral values and provide statistical information

take_picture: Use host connected camera to take a picture

take_screenshot: Take a screenshot of the target display

Deployment & flashing

deploy: Deploy one or more files on the target device

pull: Pull one or more files from the target device

switch_sdwire_to_host/_to_board: Use SDWire device to emulate a physical SDCard

ocd_start/_end/_write/_reset: Deploy firmware and reset target with a JTAG probe

nfs/tftpd: New Linux firmware deployment

ocd_command: Run any OpenOCD command

Flow control

wait: Wait for a specific duration; duration can be a number in seconds (10 or 1.5) or a string like 1h 2m 3s

wait_for_pattern: Wait for a specific pattern on the console

break_sequence: Break the iteration of the parent group immediately

External control

power_on/_off/_cycle: Use the power controller defined to control the board power state

gpio_write/_read/_write (Raspi): Interact with GPIOs to perform an action or check state on the host or DUT

Host interface control

ble_scan/_connect/_disconnect/_gatt_await_notification/_gatt_read/_gatt_write: Test a BLE device acting as a GATT server

can_open/_close/_scan_ids/_scan_messages/_read_messages/_write_messages: Test device's nodes connectivity and behavior on a CAN bus

serial_write: Write to a serial port

serial_read: Check & save the value read

UI control

mouse_move/_click: Control the mouse movement and button on the target board

keyboard: Simulate keyboard keystrokes on the target board

qt_mouse_click/_mouse_begin_drag/_mouse_end_drag/_mouse_drop_urls/_input_text/_enter_key/_get_property/_set_property/_qt_get_bounding_box/_qt_exists_and_visible/_qt_take_screenshot/_qt_quit: Full interaction with Qt Qml graphical interface (click, get color, get text, etc.)

User interaction

manual_action: Print a message and waits for the user to press ENTER

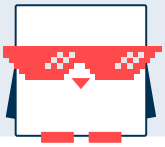
manual_test: Print a message, expected behavior, and wait for user's feedback

Web

http_request: Send an HTTP request

postman_run: Test web APIs by running a postman collection

Linux test suite



Developed with years of experience in Linux kernel and driver development, system image creation and customization, this test suite offers a vast set of tests specialized for your Linux projects - ensuring your Linux system is issue free.

Boot tests

Boot: boot a board and execute a command on the device
Power failure: test the resilience of the board when the power is cut during a stressful command
Reboot: reboot a board and execute a command on the device

I2C tests

I2C detect: detect the I^2C buses and verify that required devices are present

System tests

LTP: the Linux Test Project integration
Ptests: run ptests suite on the board
Physical RAM regions: verify that userspecified RAM regions are found in /proc/iomem
Random Number Generator Test: rng performance test
System services: system service presence check

PCI tests

PCI enumeration: check required devices on the device

Network tests

Ping: ping between host and device
Iperf: iperf between host and device
Port scanning: port scanning using Nmap
SSH: cryptographic algorithms | no password authentication | SSH password authentication

WiFi tests

AP Connect: connect to an Access Point on a set of frequency/channel width/standard
AP Scan: scan an Access Point on a set of frequency/channel width/standard
Authentication: configure a specific authentication on an access point and verify that the target under test can (or cannot) connect to it
Latency: test the ping latency of the Wi-Fi link
Performance: test the performance (bandwidth, link speed) of the Wi-Fi link
Robustness AP on/off: test the robustness of the target when the access point is disabled and re-enabled

Linux Test Project (LTP) integration

The Linux Test Project (LTP) is a large suite of Linux tests that covers different categories of the Linux system.

Yocto Ptest integration

Pluma wraps 'ptest-runner' to allow for:

- Making the appropriate checks to verify that user-specified ptests (if any) are installed
- Running 'ptest-runner' with or without a ptest list
- Parsing the output of 'ptest-runner' and verifying that user-specified ptests (if any) were actually run
- Generating success rate metrics for each ptests.