Workload Automation Documentation

Release 2.7.0

WA Mailing List <workload-automation@arm.com>,Sergei Trofimo

Jul 06, 2018

Contents

Workload Automation (WA) is a framework for running workloads on real hardware devices. WA supports a number of output formats as well as additional instrumentation (such as Streamline traces). A number of workloads are included with the framework.

Contents

CHAPTER 1

What's New

1.1 What's New in Workload Automation

1.1.1 Version 2.6.0

Note: Users who are currently using the GitHub master version of WA should uninstall the existing version before upgrading to avoid potential issues.

Additions:

Workloads

- `AdobeReader`: A workload that carries out following typical productivity tasks. These include opening a file, performing various gestures and zooms on screen and searching for a predefined set of strings.
- `octaned8`: A workload to run the binary (non-browser) version of the JS benchmark Octane.
- `GooglePlayBooks`: A workload to perform standard productivity tasks with Google Play Books. This workload performs various tasks, such as searching for a book title online, browsing through a book, adding and removing notes, word searching, and querying information about the book.
- `GooglePhotos`: A workload to perform standard productivity tasks with Google Photos. Carries out various tasks, such as browsing images, performing zooms, and post-processing the image.
- `GoogleSlides`: Carries out various tasks, such as creating a new presentation, adding text, images, and shapes, as well as basic editing and playing a slideshow.
- `Youtube`: The workload plays a video, determined by the `video_source` parameter. While the video is playing, some common actions such as video seeking, pausing playback and navigating the comments section are performed.

- `Skype`: Replacement for the `skypevideo` workload. Logs into Skype and initiates a voice or video call with a contact.

Framework

- `AndroidUxPerfWorkload`: Added a new workload class to encapsulate functionality common to all uxperf workloads.
- `UxPerfUiAutomation`: Added class which contains methods specific to UX performance testing.
- `get-assets`: Added new script and command to retrieve external assets for workloads

Results Processors

- `uxperf`: Parses device logcat for *UX_PERF* markers to produce performance metrics for workload actions using specified instrumentation.

Other

- `State Detection`: Added feature to use visual state detection to verify the state of a workload after setup and run.

Fixes/Improvements:

Documentation

- `Revent`: Added file structure to the documentation.
- Clarified documentation regarding binary dependencies.
- Updated documentation with `create` and `get-assets` commands.

Instruments

- `sysfs_extractor`: Fixed error when *tar.gz* file already existed on device, now overwrites.
- `cpufreq`: Fixed error when *tar.gz* file already existed on device, now overwrites.
- **`file-poller`:**
  - Improved csv output.
  - Added error checking and reporting.
  - Changed `files` to be a mandatory parameter.
- **`fps`:**
  - Added a new parameter to fps instrument to specify the time period between calls to `dumpsys SurfaceFlinger --latency` when collecting frame data.
  - Added gfxinfo methods to obtain fps stats. Auto detects and uses appropriate method via android version of device.
  - Fixed issue with regex.

    - Now handles empty frames correctly.
- `energy_model`: Ensures that the `ui` runtime parameter is only set for ChromeOS devices.
- `ftrace`: Added support to handle traces collected by both WA and devlib.
- `Perf`: Updated 32bit binary file for little endian devices.

Resource Getters

- `http_getter`: Now used to try and find executables files from a provided `remove_assets_url`.

Result Processors

- `cpu_states`: Fixes using stand-alone script with timeline option.

Workloads

- `antutu`: Fixed setting permissions of `FINE_LOCATION` on some devices.
- `bbench` Fixed handling of missing results.
- **`camerarecord`:**
    - Added frame stats collection through dumpsys gfxinfo.
    - Added possibility to select slow_motion recording mode.
- **`Geekbench`:**
    - Fixed output file listing causing pull failure.
    - Added support for Geekbench 4.
- **`recentfling`:**
    - Fixed issue when binaries were not uninstalled correctly.
    - Scripts are now deployed via `install()` to ensure they are executable.
    - Fixed handling of when a PID file is deleted before reaching processing results stage.
    - Added parameter to not start any apps before flinging.
- `rt-app`: Added camera recorder simulation.
- `sysbench`: Added arm64 binary.
- `Vellamo`: Fixed capitalization in part of UIAutomation to prevent potential issues.
- `Spec2000`: Now uses WA deployed version of busybox.
- `NetStat`: Updated to support new default logcat format in Android 6.
- `Dex2oat`: Now uses root if available.

Framework

- **adb_shell:**
  - Fixed issue when using single quoted command with `adb_shell`.
  - Correctly forward stderror to the caller for newer version of adb.
- **revent**
  - Added `-S` argument to "record" command to automatically record a screen capture after a recording is completed.
  - Fixed issue with multiple iterations of a revent workload.
  - Added `-s` option to executable to allow waiting on stdin.
  - Removed timeout in command as `-s` is specified.
  - Revent recordings can now be parsed and used within WA.
  - Fixed issue when some recordings wouldn't be retrieved correctly.
  - Timeout is now based on recording duration.
  - Added *magic* and file version to revent files. Revent files should now start with `REVENT` followed by the file format version.
  - Added support for gamepad recording. This type of recording contains only the events from a gamepad device (which is automatically identified).
  - A `mode` field has been added to the recording format to help distinguish between the normal and gamepad recording types.
  - Added `-g` option to `record` command to expose the gamepad recording mode.
  - The structure of revent code has undergone a major overhaul to improve maintainability and robustness.
  - More detailed `info` command output.
  - Updated Makefile to support debug/production builds.
- `Android API`: Upgraded Android API level from 17 to 18.
- `uiautomator`: The window hierarchy is now dumped to a file when WA fails on android devices.
- **AndroidDevice:**
  - Added support for downgrading when installing an APK.
  - Added a `broadcast_media_mounted` method to force a re-index of the mediaserver cache for a specified directory.
  - Now correctly handles `None` output for `get_pids_of()` when there are no running processes with the specified name.
  - Renamed the capture method from `capture_view_hierachy` to `capture_ui_hierarchy`.
  - Changed the file extension of the capture file to `.uix`
  - Added `-rf` to delete_files to be consistent with `LinuxDevice`.
- `LinuxDevice`: Now ensures output from both stdout and etderr is propagated in the event of a DeviceError.
- **APKWorkload:**
  - Now ensure APKs are replaced properly when reinstalling.

  - Now checks APK version and ABI when installing.
  - Fixed error on some devices when trying to grant permissions that were already granted.
  - Fixed some permissions not being granted.
  - Now allows disabling the main activity launch in setup (required for some apps).
  - Added parameter to clear data on reset (default behaviour unchanged).
  - Ignores exception for non-fatal permission grant failure.
  - Fixed issue of multiple versions of the same workload failing to find their APK.
  - Added method to ensure a valid apk version is used within a workload.
  - Updated how APK resolution is performed to maximise likelihood of a workload running.
  - When `check_apk` is `True` will prefer host APK and if no suitable APK is found, will use target APK if the correct version is present. When `False` will prefer target apk if it is a valid version otherwise will fallback to host APK.
- `RunConfiguration`: Fixed disabling of instruments in workload specs.
- **`Devices`:**
  - Added network connectivity check for devices.
  - Subclasses can now set `requires_network` to `True` and network connectivity check will be performed during `setup()`.
- **`Workloads`:**
  - Added network check methods.
  - Fixed versions to be backwards compatible.
  - Updated workload versions to match APK files.
  - Fixed issues with calling super.
- `Assets`: Added script to retrieve external assets for workloads.
- `Execution`: Added a `clean_up` global config option to delete WA files from devices.
- `Runner`: No longer takes a screenshot or dump of UI hierarchy for some errors when unnecessary, e.g. host errors.
- `core`: Constraints and allowed values are now checked when set instead of when validating.
- **`FpsProcessor`:**
  - Added requirement on `filtered_vsyncs_to_compose` for `total_vsync metric`.
  - Removed misleading comment in class description.
- `BaseUiAutomation`: Added new Marker API so workloads generate start and end markers with a string name.
- `AndroidUiAutoBenchmark`: Automatically checks for known package versions that don't work well with AndroidUiAutoBenchmark workloads.

Other

- Updated setup.py url to be a valid URI.
- Fixed workload name in big.Little sample agenda.

Incompatible changes

Framework

- `check_abi`: Now renamed to `exact_abi`, is used to ensure that if enabled, only an apk containing no native code or code designed for the devices primary abi is use.
- `AndroidDevice`: Renamed `supported_eabis` property to `supported_abis` to be consistent with linux devices.

Workloads

- `skypevideo`: Workload removed and replaced with `skype` workload.

1.1.2 Version 2.5.0

Additions:

Instruments

- `servo_power`: Added support for chromebook servo boards.
- `file_poller`: polls files and outputs a CSV of their values over time.
- `systrace`: The Systrace tool helps analyze the performance of your application by capturing and displaying execution times of your applications processes and other Android system processes.

Workloads

- `blogbench`: Blogbench is a portable filesystem benchmark that tries to reproduce the load of a real-world busy file server.
- `stress-ng`: Designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces.
- `hwuitest`: Uses hwuitest from AOSP to test rendering latency on Android devices.
- `recentfling`: Tests UI jank on android devices.
- `apklaunch`: installs and runs an arbitrary apk file.
- `googlemap`: Launches Google Maps and replays previously recorded interactions.

Framework

- `wlauto.utils.misc`: Added `memoised` function decorator that allows caching of previous function/method call results.
- **Added new `Device` APIs:**
  - `lsmod`: lists kernel modules
  - `insmod`: inserts a kernel module from a `.ko` file on the host.

  - `get_binary_path`: Checks `binary_directory` for the wanted binary, if it is not found there it will try to use `which`
  - `install_if_needed`: Will only install a binary if it is not already on the target.
  - `get_device_model`: Gets the model of the device.
- **`wlauto.core.execution.ExecutionContext`:**
  - `add_classfiers`: Allows adding a classfier to all metrics for the current result.

Other

- **Commands:**
  - `record`: Simplifies recording revent files.
  - `replay`: Plays back revent files.

Fixes/Improvements:

Devices

- **`juno`:**
  - Fixed `bootargs` parameter not being passed to `_boot_via_uboot`.
  - Removed default `bootargs`
- **`gem5_linux`:**
  - Added `login_prompt` and `login_password_prompt` parameters.
- `generic_linux`: ABI is now read from the target device.

Instruments

- **`trace-cmd`:**
  - Added the ability to report the binary trace on the target device, removing the need for `trace-cmd` binary to be present on the host.
  - Updated to handle messages that the trace for a CPU is empty.
  - Made timeout for pulling trace 1 minute at minimum.
- **`perf`: per-cpu statistics now get added as metrics to the results (with a** classifier used to identify the cpu).
- **`daq`:**
  - Fixed bug where an exception would be raised if `merge_channels=False`
  - No longer allows duplicate channel labels
- **`juno_energy`:**
  - Summary metrics are now calculated from the contents of `energy.csv` and added to the overall results.
  - Added a `strict` parameter. When this is set to `False` the device check during validation is omitted.
- `sysfs_extractor`: tar and gzip are now performed separately to solve permission issues.

- **fps:**
  - Now only checks for crashed content if `crash_check` is `True`.
  - Can now process multiple `view` attributes.
- `hwmon`: Sensor naming fixed, they are also now added as result classifiers

Resource Getters

- `extension_asset`: Now picks up the path to the mounted filer from the `remote_assets_path` global setting.

Result Processors

- **cpustates:**
  - Added the ability to configure how a missing `START` marker in the trace is handled.
  - Now raises a warning when there is a `START` marker in the trace but no `STOP` marker.
  - Exceptions in PowerStateProcessor no longer stop the processing of the rest of the trace.
  - Now ensures a known initial state by nudging each CPU to bring it out of idle and writing starting CPU frequencies to the trace.
  - Added the ability to create a CPU utilisation timeline.
  - Fixed issues with getting frequencies of hotplugged CPUs
- `csv`: Zero-value classifieres are no longer converted to an empty entry.
- `ipynb_exporter`: Default template no longer shows a blank plot for workloads without `summary_metrics`

Workloads

- **vellamo:**
  - Added support for v3.2.4.
  - Fixed getting values from logcat.
- `cameracapture`: Updated to work with Android M+.
- `camerarecord`: Updated to work with Android M+.
- **lmbench:**
  - Added the output file as an artifact.
  - Added taskset support
- `antutu` - Added support for v6.0.1
- `ebizzy`: Fixed use of `os.path` to `self.device.path`.
- `bbench`: Fixed browser crashes & permissions issues on android M+.
- **geekbench:**
  - Added check whether device is rooted.

- `manual`: Now only uses logcat on Android devices.
- **`applaunch`:**
  - Fixed `cleanup` not getting forwarded to script.
  - Added the ability to stress IO during app launch.
- `dhrystone`: Now uses WA's resource resolution to find it's binary so it uses the correct ABI.
- `glbench`: Updated for new logcat formatting.

Framework

- **`ReventWorkload`:**
  - Now kills all revent instances on teardown.
  - Device model name is now used when searching for revent files, falling back to WA device name.
- **`BaseLinuxDevice`:**
  - `killall` will now run as root by default if the device is rooted.
  - `list_file_systems` now handles blank lines.
  - All binaries are now installed into `binaries_directory` this allows..
  - Busybox is now deployed on non-root devices.
  - gzipped property files are no zcat'ed
- **`LinuxDevice`:**
  - `kick_off` no longer requires root.
  - `kick_off` will now run as root by default if the device is rooted.
  - No longer raises an exception if a connection was dropped during a reboot.
  - Added a delay before polling for a connection to avoid re-connecting to a device that is still in the process of rebooting.
- `wlauto.utils.types`: `list_or_string` now ensures that elements of a list are strings.
- **`AndroidDevice`:**
  - `kick_off` no longer requires root.
  - Build props are now gathered via `getprop` rather than trying to parse build.prop directly.
  - WA now pushes its own `sqlite3` binary.
  - Now uses `content` instead of `settings` to get `ANDROID_ID`
  - **`swipe_to_unlock` parameter is now actually used. It has been changed to** take a direction to accomodate various devices.
  - `ensure_screen_is_on` will now also unlock the screen if swipe_to_unlock is set.
  - Fixed use of variables in as_root=True commands.
  - `get_pids_of` now used `busybox grep` since as of Android M+ ps cannot filter by process name anymore.
  - Fixed installing APK files with whitespace in their path/name.
- **`adb_shell`:**

    - Fixed handling of line breaks at the end of command output.
    - Newline separator is now detected from the target.
    - As of ADB v1.0.35, ADB returns the return code of the command run. WA now handles this correctly.
- **ApkWorkload:**
    - Now attempts to grant all runtime permissions for devices on Android M+.
    - Can now launch packages that don't have a launch activity defined.
    - Package version is now added to results as a classifier.
    - Now clears app data if an uninstall failed to ensure it starts from a known state.
- `wlauto.utils.ipython`: Updated to work with ipython v5.
- **Gem5Device:**
    - Added support for deploying the `m5` binary.
    - No longer waits for the boot animation to finish if it has been disabled.
    - Fixed runtime error caused by lack of kwargs.
    - No longer depends on `busybox`.
    - Split out commands to resize shell to `resize_shell`.
    - Now tries to connect to the shell up to 10 times.
    - No longer renames gzipped files.
- Agendas: - Now errors when an agenda key is empty.
- `wlauto.core.execution.RunInfo`: `run_name` will now default to `{output_folder}_{date}_{time}`.
- **Extensions:**
    - Two different parameters can now have the same global alias as long as they their types match.
    - You can no longer `override` parameters that are defined at the same level.
- `wlauto.core.entry_point`: Now gives a better error when a config file doesn't exist.
- `wlauto.utils.misc`: Added `aarch64` to list for arm64 ABI.
- `wlauto.core.resolver`: Now shows what version was being search for when a resource is not found.
- Will no longer start instruments ect. if a run has no workload specs.
- `wlauto.utils.uboot`: Now detects uboot version to use correct line endings.
- `wlauto.utils.trace_cmd`: Added a parser for sched_switch events.

Other

- Updated to pylint v1.5.1
- Rebuilt `busybox` binaries to prefer built-in applets over system binaries.
- `BaseUiAutomation`: Added functions for checking version strings.

Incompatible changes

Instruments

- `apk_version`: Removed, use result classifiers instead.

Framework

- `BaseLinuxDevice`: Removed `is_installed` use `install_if_needed` and `get_binary_path` instead.
- `LinuxDevice`: Removed `has_root` method, use `is_rooted` instead.
- `AndroidDevice`: `swipe_to_unlock` method replaced with `perform_unlock_swipe`.

1.1.3 Version 2.4.0

Additions:

Devices

- `gem5_linux` and `gem5_android`: Interfaces for Gem5 simulation environment running Linux and Android respectively.
- `XE503C1211`: Interface for Samsung XE503C12 Chromebooks.
- `chromeos_test_image`: Chrome OS test image device. An off the shelf device will not work with this device interface.

Instruments

- `freq_sweep`: Allows "sweeping" workloads across multiple CPU frequencies.
- `screenon`: Ensures screen is on, before each iteration, or periodically on Android devices.
- `energy_model`: This instrument can be used to generate an energy model for a device based on collected power and performance measurments.
- `netstats`: Allows monitoring data sent/received by applications on an Android device.

Modules

- `cgroups`: Allows query and manipulation of cgroups controllers on a Linux device. Currently, only cpusets controller is implemented.
- `cpuidle`: Implements cpuidle state discovery, query and manipulation for a Linux device. This replaces the more primitive get_cpuidle_states method of LinuxDevice.
- `cpufreq` has now been split out into a device module

Reasource Getters

- `http_assets`: Downloads resources from a web server.

Results Processors

- `ipynb_exporter`: Generates an IPython notebook from a template with the results and runs it.
- `notify`: Displays a desktop notification when a run finishes (Linux only).
- `cpustates`: Processes power ftrace to produce CPU state and parallelism stats. There is also a script to invoke this outside of WA.

Workloads

- `telemetry`: Executes Google's Telemetery benchmarking framework
- `hackbench`: Hackbench runs tests on the Linux scheduler
- `ebizzy`: This workload resembles common web server application workloads.
- `power_loadtest`: Continuously cycles through a set of browser-based activities and monitors battery drain on a device (part of ChromeOS autotest suite).
- `rt-app`: Simulates configurable real-time periodic load.
- `linpack-cli`: Command line version of linpack benchmark.
- `lmbench`: A suite of portable ANSI/C microbenchmarks for UNIX/POSIX.
- `stream`: Measures memory bandwidth.
- `iozone`: Runs a series of disk I/O performance tests.
- `androbench`: Measures the storage performance of device.
- `autotest`: Executes tests from ChromeOS autotest suite.

Framework

- **`wlauto.utils`:**
  - Added `trace_cmd`, a generic trace-cmd paraser.
  - Added `UbootMenu`, allows navigating Das U-boot menu over serial.
- **`wlauto.utils.types`:**
  - `caseless_string`: Behaves exactly like a string, except this ignores case in comparisons. It does, however, preserve case.
  - `list_of`: allows dynamic generation of type-safe list types based on an existing type.
  - `arguments`: represents arguments that are passed on a command line to an application.
  - `list-or`: allows dynamic generation of types that accept either a base type or a list of base type. Using this `list_or_integer`, `list_or_number` and `list_or_bool` were also added.
- **`wlauto.core.configuration.WorkloadRunSpec`:**
  - `copy`: Allows making duplicates of `WorkloadRunSpec`'s
- **`wlatuo.utils.misc`:**
  - `list_to_ranges` and `ranges_to_list`: convert between lists of integers and corresponding range strings, e.g. between [0,1,2,4] and '0-2,4'

  - `list_to_mask` and `mask_to_list`: convert between lists of integers and corresponding integer masks, e.g. between [0,1,2,4] and 0x17

- **`wlauto.instrumentation`:**
  - `instrument_is_enabled`: Returns whether or not an instrument is enabled for the current job.
- **`wlauto.core.result`:**
  - Added "classifiers" field to Metric objects. This is a dict mapping classifier names (arbitrary strings) to corresponding values for that specific metrics. This is to allow extensions to add extension-specific annotations to metric that could be handled in a generic way (e.g. by result processors). They can also be set in agendas.
- Failed jobs will now be automatically retired
- Implemented dynamic device modules that may be loaded automatically on device initialization if the device supports them.
- Added support for YAML configs.
- Added `initialze` and `finalize` methods to workloads.
- **`wlauto.core.ExecutionContext`:**
  - Added `job_status` property that returns the status of the currently running job.

Fixes/Improvements

Devices

- `tc2`: Workaround for buffer overrun when loading large initrd blob.
- **`juno`:**
  - UEFI config can now be specified as a parameter.
  - Adding support for U-Boot booting.
  - No longer auto-disconnects ADB at the end of a run.
  - Added `actually_disconnect` to restore old disconnect behaviour
  - Now passes `video` command line to Juno kernel to work around a known issue where HDMI loses sync with monitors.
  - Fixed flashing.

Instruments

- **`trace_cmd`:**
  - Fixed `buffer_size_file` for non-Android devices
  - Reduce starting priority.
  - Now handles trace headers and thread names with spaces
- `energy_probe`: Added `device_entry` parameter.
- **`hwmon`:**
  - Sensor discovery is now done only at the start of a run.

  - Now prints both before/after and mean temperatures.

- **daq:**
  - Now reports energy
  - Fixed file descriptor leak
  - `daq_power.csv` now matches the order of labels (if specified).
  - Added `gpio_sync`. When enabled, this wil cause the instrument to insert a marker into ftrace, while at the same time setting a GPIO pin high.
  - Added `negative_values` parameter. which can be used to specify how negative values in the samples should be handled.
  - Added `merge_channels` parameter. When set DAQ channel will be summed together.
  - Workload labels, rather than names, are now used in the "workload" column.
- **cpufreq:**
  - Fixes missing directories problem.
  - Refined the availability check not to rely on the top-level cpu/cpufreq directory
  - Now handles non-integer output in `get_available_frequencies`.
- **sysfs_extractor:**
  - No longer raises an error when both device and host paths are empty.
  - Fixed pulled files verification.
- **perf:**
  - Updated binaries.
  - Added option to force install.
  - `killall` is now run as root on rooted Android devices.
- **fps:**
  - now generates detailed FPS traces as well as report average FPS.
  - Updated jank calcluation to only count "large" janks.
  - Now filters out bogus `actual-present` times and ignore janks above `PAUSE_LATENCY`
- **delay:**
  - Added `fixed_before_start` parameter.
  - Changed existing `*_between_specs` and `*_between_iterations` callbacks to be `very_slow`
- **streamline:**
  - Added Linux support
  - `gatord` is now only started once at the start of the run.

modules

- **flashing:**
  - Fixed vexpress flashing

  - Added an option to keep UEFI entry

Result Processors

- **cpustate:**
  - Now generates a timeline csv as well as stats.
  - Adding ID to overall cpustate reports.
- `csv`: (partial) `results.csv` will now be written after each iteration rather than at the end of the run.

Workloads

- `glb_corporate`: clears logcat to prevent getting results from previous run.
- **sysbench:**
  - Updated sysbench binary to a statically linked verison
  - Added `file_test_mode parameter` - this is a mandatory argumet if `test` is `"fileio"`.
  - Added `cmd_params` parameter to pass options directly to sysbench invocation.
  - Removed Android browser launch and shutdown from workload (now runs on both Linux and Android).
  - Now works with unrooted devices.
  - Added the ability to run based on time.
  - Added a parameter to taskset to specific core(s).
  - Added `threads` parameter to be consistent with dhrystone.
  - Fixed case where default `timeout` < `max_time`.
- **Dhrystone:**
  - added `taskset_mask` parameter to allow pinning to specific cores.
  - Now kills any running instances during setup (also handles CTRL-C).
- `sysfs_extractor`: Added parameter to explicitly enable/disable tempfs caching.
- **antutu:**
  - Fixed multi-`times` playback for v5.
  - Updated result parsing to handle Android M logcat output.
- `geekbench`: Increased timout to cater for slower devices.
- `idle`: Now works on Linux devices.
- `manhattan`: Added `run_timemout` parameter.
- `bbench`: Now works when binaries_directory is not in path.
- `nemamark`: Made duration configurable.

Framework

- **BaseLinuxDevice:**
  - Now checks that at least one core is enabled on another cluster before attempting to set number of cores on a cluster to `0`.
  - No longer uses `sudo` if already logged in as `root`.
  - Now saves `dumpsys window` output to the `__meta` directory.
  - Now takes `password_prompt` as a parameter for devices with a non standard `sudo` password prompt.
  - No longer raises an error if `keyfile` or `password` are not provided when they are not necessary.
  - **Added new cpufreq APIs:**
    - `core` APIs take a core name as the parameter (e.g. "a15")
    - `cluster` APIs take a numeric cluster ID (eg. 0)
    - `cpu` APIs take a cpufreq cpu ID as a parameter.
  - `set_cpu_frequency` now has a `exact` parameter. When true (the default) it will produce an error when the specified frequency is not supported by the cpu, otherwise cpufreq will decide what to do.
  - Added `{core}_frequency` runtime parameter to set cluster frequency.
  - Added `abi` property.
  - `get_properties` moved from `LinuxDevice`, meaning `AndroidDevice` will try to pull the same files. Added more paths to pull by default too.
  - fixed `list_file_systems` for Android M and Linux devices.
  - Now sets `core_clusters` from `core_names` if not explicitly specified.
  - Added `invoke` method that allows invoking an executable on the device under controlled contions (e.g. within a particular directory, or taskset to specific CPUs).
  - No longer attempts to `get_sysfile_value()` as root on unrooted devices.
- **LinuxDevice:**
  - Now creates `binaries_directory` path if it doesn't exist.
  - Fixed device reset
  - Fixed `file_exists`
  - implemented `get_pid_of()` and `ps()`. Existing implementation relied on Android version of ps.
  - `listdir` will now return an empty list for an empty directory instead of a list containing a single empty string.
- **AndroidDevice:**
  - Executable (un)installation now works on unrooted devices.
  - Now takes into account `binar_directory` when setting up busybox path.
  - update `android_prompt` so that it works even if is not `"/"`
  - `adb_connect`: do not assume port 5555 anymore.
  - Now always deploys busybox on rooted devices.

    - Added `swipe_to_unlock` method.
- Fixed initialization of `~/.workload_automation.`.
- Fixed replaying events using revent on 64 bit platforms.
- Improved error repoting when loading extensions.
- `result` objects now track their output directories.
- `context.result` will not result in `context.run_result` when not executing a job.
- **`wlauto.utils.ssh`:**
    - Fixed key-based authentication.
    - Fixed carriage return stripping in ssh.
    - Now takes `password_prompt` as a parameter for non standard `sudo` password prompts.
    - Now with 100% more thread safety!
    - If a timeout condition is hit, ^C is now sent to kill the current foreground process and make the shell available for subsequent commands.
    - More robust `exit_code` handling for ssh interface
    - Now attempts to deal with dropped connections
    - Fixed error reporting on failed exit code extraction.
    - Now handles backspaces in serial output
    - Added `port` argument for telnet connections.
    - Now allows telnet connections without a password.
- Fixed config processing for extensions with non-identifier names.
- Fixed `get_meansd` for numbers < 1
- **`wlatuo.utils.ipython`:**
    - Now supports old versions of IPython
    - Updated version check to only initialize ipython utils if version is < 4.0.0. Version 4.0.0 changes API and breaks WA's usage of it.
- Added `ignore` parameter to `check_output`
- **Agendas:**
    - Now raise an error if an agenda contains duplicate keys
    - Now raise an error if config section in an agenda is not dict-like
    - Now properly handles `core_names` and `core_clusters`
    - When merging list parameters from different sources, duplicates are no longer removed.
- The `INITIAL_BOOT` signal is now sent went performing a hard reset during intial boot
- updated `ExecutionContext` to keep a reference to the `runner`. This will enable Extenstions to do things like modify the job queue.
- Parameter now automatically convert int and boot kinds to integer and boolean respectively, this behavior can be supressed by specifying `convert_types``=``False` when defining the parameter.
- Fixed resource resolution when dependency location does not exist.

- All device `push` and `pull` commands now raise `DeviceError` if they didn't succeed.
- Fixed showing Parameter default of `False` for boolean values.
- Updated csv result processor with the option to use classifiers to add columns to `results.csv`.
- `wlauto.utils.formatter`: Fix terminal size discovery.
- The extension loader will now follow symlinks.
- Added arm64-v8a to ABI map
- WA now reports syntax errors in a more informative way.
- Resource resolver: now prints the path of the found resource to the log.
- Resource getter: look for executable in the bin/ directory under resource owner's dependencies directory as well as general dependencies bin.
- **`GamingWorkload`:**
  - Added an option to prevent clearing of package data before execution.
  - Added the ability to override the timeout of deploying the assets tarball.
- `ApkWorkload`: Added an option to skip host-side APK check entirely.
- `utils.misc.normalize`: only normalize string keys.
- Better error reporting for subprocess.CalledProcessError
- `boolean` now interprets `'off'` as `False`
- `wlauto.utils.uefi`: Added support for debug builds.
- `wlauto.utils.serial_port`: Now supports fdexpect versions > 4.0.0
- Semanatics for `initialize`/`finalize` for *all* Extensions are changed so that now they will always run at most once per run. They will not be executed twice even if invoked via instances of different subclasses (if those subclasses defined their own verions, then their versions will be invoked once each, but the base version will only get invoked once).
- Pulling entries from procfs does not work on some platforms. WA now tries to cat the contents of a property_file and write it to a output file on the host.

Documentation

- **`installation`:**
  - Added `post install` section which lists workloads that require additional external dependencies.
  - Added the `uninstall` and `upgrade` commands for users to remove or upgrade Workload Automation.
  - Added documentation explaining how to use `remote_assets_path` setting.
  - Added warning about potential permission issues with pip.
- `quickstart`: Added steps for setting up WA to run on Linux devices.
- `device_setup`: fixed `generic_linux` `device_config` example.
- `contributing`: Clarified style guidelines
- `daq_device_setup`: Added an illustration for DAQ wiring.
- `writing_extensions`: Documented the Workload initialize and finalize methods.

- Added descriptions to extension that didn't have one.

Other

- **daq_server:**
  - Fixed showing available devices.
  - Now works with earlier versions of the DAQmx driver.thus you can now run the server on Linux systems.
  - DAQ error messages are now properly propaged to the client.
  - Server will now periodically clean up uncollected files.
  - fixed not being able to resolve IP address for hostname (report "localhost" in that case).
  - Works with latest version of twisted.
- `setup.py`: Fixed paths to work with Mac OS X.
- `summary_csv` is no longer enabled by default.
- `status` result processor is now enabled by default.
- **Commands:**
  - **show:**
    - Now shows what platform extensions support.
    - Will no longer try to use a pager if `PAGER=''` in the environment.
  - **list:**
    - Added `"-p"` option to filter results by supported platforms.
    - Added `"--packaged-only"` option to only list extensions packaged with WA.
  - `run`: Added `"--disable"` option to diable instruments.
  - **create:**
    - Added `agenda` sub-command to generate agendas for a set of extensions.
    - `create workload` now gives more informative errors if Android SDK installed but no platform has been downloaded.

Incompatible changes

Framework

- **BaseLinuxDevice:**
  - Renamed `active_cpus` to `online_cpus`
  - Renamed `get_cluster_cpu` to `get_cluster_active_cpu`
  - Renamed `get_core_cpu` to `get_core_online_cpu`
- All extension's `initialize` function now takes one (and only one) parameter, `context`.
- `wlauto.core.device`: Removed `init` function. Replaced with `initialize`

1.1.4 Version 2.3.0

- First publicly-released version.

CHAPTER 2

Usage

This section lists general usage documentation. If you're new to WA2, it is recommended you start with the *Quickstart* page. This section also contains installation and configuration guides.

2.1 Quickstart

This guide will show you how to quickly start running workloads using Workload Automation 2.

2.1.1 Install

Note: This is a quick summary. For more detailed instructions, please see the *Installation* section.

Make sure you have Python 2.7 and a recent Android SDK with API level 18 or above installed on your system. A complete install of the Android SDK is required, as WA uses a number of its utilities, not just adb. For the SDK, make sure that either `ANDROID_HOME` environment variable is set, or that `adb` is in your `PATH`.

Note: If you plan to run Workload Automation on Linux devices only, SSH is required, and Android SDK is optional if you wish to run WA on Android devices at a later time.

However, you would be starting off with a limited number of workloads that will run on Linux devices.

In addition to the base Python 2.7 install, you will also need to have `pip` (Python's package manager) installed as well. This is usually a separate package.

Once you have those, you can install WA with:

```
sudo -H pip install wlauto
```

This will install Workload Automation on your system, along with its mandatory dependencies.

(Optional) Verify installation

Once the tarball has been installed, try executing

```
wa -h
```

You should see a help message outlining available subcommands.

(Optional) APK files

A large number of WA workloads are installed as APK files. These cannot be distributed with WA and so you will need to obtain those separately.

For more details, please see the *Installation* section.

2.1.2 Configure Your Device

Locate the device configuration file, config.py, under the ~/.workload_automation directory. Then adjust the device configuration settings accordingly to the device you are using.

Android

By default, the device is set to 'generic_android'. WA is configured to work with a generic Android device through `adb`. If you only have one device listed when you execute `adb devices`, and your device has a standard Android configuration, then no extra configuration is required.

However, if your device is connected via network, you will have to manually execute `adb connect <device ip>` so that it appears in the device listing.

If you have multiple devices connected, you will need to tell WA which one you want it to use. You can do that by setting `adb_name` in device_config section.

```
# ...

device_config = dict(
        adb_name = 'abcdef0123456789',
        # ...
)

# ...
```

Linux

First, set the device to 'generic_linux'

```
# ...
  device = 'generic_linux'
# ...
```

Find the device_config section and add these parameters

```
# ...

device_config = dict(
        host = '192.168.0.100',
        username = 'root',
        password = 'password'
        # ...
)

# ...
```

Parameters:

- Host is the IP of your target Linux device
- Username is the user for the device
- Password is the password for the device

Enabling and Disabling Instrumentation

Some instrumentation tools are enabled after your initial install of WA.

Note: Some Linux devices may not be able to run certain instruments provided by WA (e.g. cpufreq is disabled or unsupported by the device).

As a start, keep the 'execution_time' instrument enabled while commenting out the rest to disable them.

```
# ...

Instrumentation = [
        # Records the time it took to run the workload
        'execution_time',

        # Collects /proc/interrupts before and after execution and does a diff.
        # 'interrupts',

        # Collects the contents of/sys/devices/system/cpu before and after execution␣
↪and does a diff.
        # 'cpufreq',

        # ...
)
```

This should give you basic functionality. If you are working with a development board or you need some advanced functionality (e.g. big.LITTLE tuning parameters), additional configuration may be required. Please see the *Setting Up A Device* section for more details.

2.1.3 Running Your First Workload

The simplest way to run a workload is to specify it as a parameter to WA `run` sub-command:

```
wa run dhrystone
```

You will see INFO output from WA as it executes each stage of the run. A completed run output should look something like this:

```
INFO     Initializing
INFO     Running workloads
INFO     Connecting to device
INFO     Initializing device
INFO     Running workload 1 dhrystone (iteration 1)
INFO             Setting up
INFO             Executing
INFO             Processing result
INFO             Tearing down
INFO     Processing overall results
INFO     Status available in wa_output/status.txt
INFO     Done.
INFO     Ran a total of 1 iterations: 1 OK
INFO     Results can be found in wa_output
```

Once the run has completed, you will find a directory called `wa_output` in the location where you have invoked `wa run`. Within this directory, you will find a "results.csv" file which will contain results obtained for dhrystone, as well as a "run.log" file containing detailed log output for the run. You will also find a sub-directory called 'drystone_1_1' that contains the results for that iteration. Finally, you will find a copy of the agenda file in the `wa_output/__meta` subdirectory. The contents of iteration-specific subdirectories will vary from workload to workload, and, along with the contents of the main output directory, will depend on the instrumentation and result processors that were enabled for that run.

The `run` sub-command takes a number of options that control its behavior, you can view those by executing `wa run -h`. Please see the *Commands* section for details.

2.1.4 Create an Agenda

Simply running a single workload is normally of little use. Typically, you would want to specify several workloads, setup the device state and, possibly, enable additional instrumentation. To do this, you would need to create an "agenda" for the run that outlines everything you want WA to do.

Agendas are written using YAML markup language. A simple agenda might look like this:

```
config:
        instrumentation: [~execution_time]
        result_processors: [json]
global:
        iterations: 2
workloads:
        - memcpy
        - name: dhrystone
          params:
                mloops: 5
                threads: 1
```

This agenda

- Specifies two workloads: memcpy and dhrystone.
- Specifies that dhrystone should run in one thread and execute five million loops.
- Specifies that each of the two workloads should be run twice.
- Enables json result processor, in addition to the result processors enabled in the config.py.

- Disables execution_time instrument, if it is enabled in the config.py

An agenda can be created in a text editor and saved as a YAML file. Please make note of where you have saved the agenda.

Please see *Agenda* section for more options.

2.1.5 Examples

These examples show some useful options with the `wa run` command.

To run your own agenda:

```
wa run <path/to/agenda> (e.g. wa run ~/myagenda.yaml)
```

To redirect the output to a different directory other than wa_output:

```
wa run dhrystone -d my_output_directory
```

To use a different config.py file:

```
wa run -c myconfig.py dhrystone
```

To use the same output directory but override existing contents to store new dhrystone results:

```
wa run -f dhrystone
```

To display verbose output while running memcpy:

```
wa run --verbose memcpy
```

2.1.6 Uninstall

If you have installed Workload Automation via `pip`, then run this command to uninstall it:

```
sudo pip uninstall wlauto
```

Note: It will *not* remove any user configuration (e.g. the ~/.workload_automation directory).

2.1.7 Upgrade

To upgrade Workload Automation to the latest version via `pip`, run:

```
sudo pip install --upgrade --no-deps wlauto
```

2.2 Installation

This page describes how to install Workload Automation 2.

2.2.1 Prerequisites

Operating System

WA runs on a native Linux install. It was tested with Ubuntu 12.04, but any recent Linux distribution should work. It should run on either 32-bit or 64-bit OS, provided the correct version of Android (see below) was installed. Officially, **other environments are not supported**. WA has been known to run on Linux Virtual machines and in Cygwin environments, though additional configuration may be required in both cases (known issues include makings sure USB/serial connections are passed to the VM, and wrong python/pip binaries being picked up in Cygwin). WA *should* work on other Unix-based systems such as BSD or Mac OS X, but it has not been tested in those environments. WA *does not* run on Windows (though it should be possible to get limited functionality with minimal porting effort).

Note: If you plan to run Workload Automation on Linux devices only, SSH is required, and Android SDK is optional if you wish to run WA on Android devices at a later time. Then follow the steps to install the necessary python packages to set up WA.

However, you would be starting off with a limited number of workloads that will run on Linux devices.

Android SDK

You need to have the Android SDK with at least one platform installed. To install it, download the ADT Bundle from here. Extract it and add `<path_to_android_sdk>/sdk/platform-tools` and `<path_to_android_sdk>/sdk/tools` to your `PATH`. To test that you've installed it properly, run `adb version`. The output should be similar to this:

```
adb version
Android Debug Bridge version 1.0.31
```

Once that is working, run

```
android update sdk
```

This will open up a dialog box listing available android platforms and corresponding API levels, e.g. `Android 4.3 (API 18)`. For WA, you will need at least API level 18 (i.e. Android 4.3), though installing the latest is usually the best bet.

Optionally (but recommended), you should also set `ANDROID_HOME` to point to the install location of the SDK (i.e. `<path_to_android_sdk>/sdk`).

Note: You may need to install 32-bit compatibility libararies for the SDK to work properly. On Ubuntu you need to run:

```
sudo apt-get install lib32stdc++6 lib32z1
```

Python

Workload Automation 2 requires Python 2.7 (Python 3 is not supported at the moment).

pip

pip is the recommended package manager for Python. It is not part of standard Python distribution and would need to be installed separately. On Ubuntu and similar distributions, this may be done with APT:

```
sudo apt-get install python-pip
```

Note: Some versions of pip (in particluar v1.5.4 which comes with Ubuntu 14.04) are know to set the wrong permissions when installing packages, resulting in WA failing to import them. To avoid this it is recommended that you update pip and setuptools before proceeding with installation:

```
sudo -H pip install --upgrade pip
sudo -H pip install --upgrade setuptools
```

If you do run into this issue after already installing some packages, you can resolve it by running

```
sudo chmod -R a+r /usr/local/lib/python2.7/dist-packagessudo
find /usr/local/lib/python2.7/dist-packages -type d -exec chmod a+x {} \;
```

(The paths above will work for Ubuntu; they may need to be adjusted for other distros).

Python Packages

Note: pip should automatically download and install missing dependencies, so if you're using pip, you can skip this section.

Workload Automation 2 depends on the following additional libraries:

- pexpect
- docutils
- pySerial
- pyYAML
- python-dateutil

You can install these with pip:

```
sudo -H pip install pexpect
sudo -H pip install pyserial
sudo -H pip install pyyaml
sudo -H pip install docutils
sudo -H pip install python-dateutil
```

Some of these may also be available in your distro's repositories, e.g.

```
sudo apt-get install python-serial
```

Distro package versions tend to be older, so pip installation is recommended. However, pip will always download and try to build the source, so in some situations distro binaries may provide an easier fall back. Please also note that distro package names may differ from pip packages.

Optional Python Packages

Note: unlike the mandatory dependencies in the previous section, pip will *not* install these automatically, so you will have to explicitly install them if/when you need them.

In addition to the mandatory packages listed in the previous sections, some WA functionality (e.g. certain extensions) may have additional dependencies. Since they are not necessary to be able to use most of WA, they are not made mandatory to simplify initial WA installation. If you try to use an extension that has additional, unmet dependencies, WA will tell you before starting the run, and you can install it then. They are listed here for those that would rather install them upfront (e.g. if you're planning to use WA to an environment that may not always have Internet access).

- nose
- pandas
- PyDAQmx
- pymongo
- jinja2

Note: Some packages have C extensions and will require Python development headers to install. You can get those by installing `python-dev` package in apt on Ubuntu (or the equivalent for your distribution).

2.2.2 Installing

Installing the latest released version from PyPI (Python Package Index):

```
sudo -H pip install wlauto
```

This will install WA along with its mandatory dependencies. If you would like to install all optional dependencies at the same time, do the following instead:

```
sudo -H pip install wlauto[all]
```

Alternatively, you can also install the latest development version from GitHub (you will need git installed for this to work):

```
git clone git@github.com:ARM-software/workload-automation.git workload-automation
sudo -H pip install ./workload-automation
```

If the above succeeds, try

```
wa --version
```

Hopefully, this should output something along the lines of "Workload Automation version $version".

2.2.3 (Optional) Post Installation

Some WA extensions have additional dependencies that need to be statisfied before they can be used. Not all of these can be provided with WA and so will need to be supplied by the user. They should be placed into `~/.workload_automation/dependencies/<extenion name>` so that WA can find them (you may need to

create the directory if it doesn't already exist). You only need to provide the dependencies for workloads you want to use.

Binary Files

Some workloads require native binaries to work. Different binaries will be required for different ABIs. WA may not include the required binary for a workload due to licensing/distribution issues, or may not have a binary compiled for your device's ABI. In such cases, you will have to supply the missing binaries.

Executable binaries for a workload should be placed inside `~/.workload_automation/dependencies/<extension name>/bin/<ABI>` directory. This directory may not already exist, in which case you would have to create it.

Binaries placed in that location will take precidence over any already inclueded with WA. For example, if you have your own `drystone` binary compiled for `arm64`, and you want WA to pick it up, you can do the following on WA host machine

```
mkdir -p ~/.workload_automation/dependencies/dhrystone/bin/arm64/
cp /path/to/your/dhrystone ~/.workload_automation/dependencies/dhrystone/bin/arm64/
```

APK Files

APKs are applicaton packages used by Android. These are necessary to install an application onto devices that do not have Google Play (e.g. devboards running AOSP). The following is a list of workloads that will need one, including the version(s) for which UI automation has been tested. Automation may also work with other versions (especially if it's only a minor or revision difference – major version differens are more likely to contain incompatible UI changes) but this has not been tested.

workload	package	name	version code	version name
andebench	com.eembc.coremark	AndEBench	v1383a	1383
angrybirds	com.rovio.angrybirds	Angry Birds	2.1.1	2110
angrybirds_rio	com.rovio.angrybirdsrio	Angry Birds	1.3.2	1320
anomaly2	com.elevenbitstudios.anomaly2Benchmark	A2 Benchmark	1.1	50
antutu	com.antutu.ABenchMark	AnTuTu Benchmark	5.3	5030000
antutu	com.antutu.ABenchMark	AnTuTu Benchmark	3.3.2	3322
antutu	com.antutu.ABenchMark	AnTuTu Benchmark	4.0.3	4000300
benchmarkpi	gr.androiddev.BenchmarkPi	BenchmarkPi	1.11	5
caffeinemark	com.flexycore.caffeinemark	CaffeineMark	1.2.4	9
castlebuilder	com.ettinentertainment.castlebuilder	Castle Builder	1.0	1
castlemaster	com.alphacloud.castlemaster	Castle Master	1.09	109
cfbench	eu.chainfire.cfbench	CF-Bench	1.2	7
citadel	com.epicgames.EpicCitadel	Epic Citadel	1.07	901107
dungeondefenders	com.trendy.ddapp	Dungeon Defenders	5.34	34
facebook	com.facebook.katana	Facebook	3.4	258880
geekbench	ca.primatelabs.geekbench2	Geekbench 2	2.2.7	202007
geekbench	com.primatelabs.geekbench3	Geekbench 3	3.0.0	135
glb_corporate	net.kishonti.gfxbench	GFXBench	3.0.0	1
glbenchmark	com.glbenchmark.glbenchmark25	GLBenchmark 2.5	2.5	4
glbenchmark	com.glbenchmark.glbenchmark27	GLBenchmark 2.7	2.7	1
gunbros2	com.glu.gunbros2	GunBros2	1.2.2	122
ironman	com.gameloft.android.ANMP.GloftIMHM	Iron Man 3	1.3.1	1310

Continued on next page

Table 2.1 – continued from previous page

workload	package	name	version code	version name
krazykart	com.polarbit.sg2.krazyracers	Krazy Kart Racing	1.2.7	127
linpack	com.greenecomputing.linpackpro	Linpack Pro for Android	1.2.9	31
nenamark	se.nena.nenamark2	NenaMark2	2.4	5
peacekeeper	com.android.chrome	Chrome	18.0.1025469	1025469
peacekeeper	org.mozilla.firefox	Firefox	23.0	2013073011
quadrant	com.aurorasoftworks.quadrant.ui.professional	Quadrant Professional	2.0	2000000
realracing3	com.ea.games.r3_row	Real Racing 3	1.3.5	1305
smartbench	com.smartbench.twelve	Smartbench 2012	1.0.0	5
sqlite	com.redlicense.benchmark.sqlite	RL Benchmark	1.3	5
templerun	com.imangi.templerun	Temple Run	1.0.8	11
thechase	com.unity3d.TheChase	The Chase	1.0	1
truckerparking3d	com.tapinator.truck.parking.bus3d	Truck Parking 3D	2.5	7
vellamo	com.quicinc.vellamo	Vellamo	3.0	3001
vellamo	com.quicinc.vellamo	Vellamo	2.0.3	2003
videostreaming	tw.com.freedi.youtube.player	FREEdi YT Player	2.1.13	79

Gaming Workloads

Some workloads (games, demos, etc) cannot be automated using Android's UIAutomator framework because they render the entire UI inside a single OpenGL surface. For these, an interaction session needs to be recorded so that it can be played back by WA. These recordings are device-specific, so they would need to be done for each device you're planning to use. The tool for doing is `revent` and it is packaged with WA. You can find instructions on how to use it *here*.

This is the list of workloads that rely on such recordings:

angrybirds
angrybirds_rio
anomaly2
castlebuilder
castlemastera
citadel
dungeondefenders
gunbros2
ironman
krazykart
realracing3
templerun
truckerparking3d

Maintaining Centralized Assets Repository

If there are multiple users within an organization that may need to deploy assets for WA extensions, that organization may wish to maintain a centralized repository of assets that individual WA installs will be able to automatically retrieve asset files from as they are needed. This repository can be any directory on a network filer that mirrors the structure of `~/.workload_automation/dependencies`, i.e. has a subdirectories named after the extensions which assets they contain. Individual WA installs can then set `remote_assets_path` setting in their config to point to the local mount of that location.

2.2.4 (Optional) Uninstalling

If you have installed Workload Automation via `pip` and wish to remove it, run this command to uninstall it:

```
sudo -H pip uninstall wlauto
```

Note: This will *not* remove any user configuration (e.g. the ~/.workload_automation directory)

2.2.5 (Optional) Upgrading

To upgrade Workload Automation to the latest version via `pip`, run:

```
sudo -H pip install --upgrade --no-deps wlauto
```

2.3 Setting Up A Device

WA should work with most Android devices out-of-the box, as long as the device is discoverable by `adb` (i.e. gets listed when you run `adb devices`). For USB-attached devices, that should be the case; for network devices, `adb connect` would need to be invoked with the IP address of the device. If there is only one device connected to the host running WA, then no further configuration should be necessary (though you may want to *tweak some Android settings*).

If you have multiple devices connected, have a non-standard Android build (e.g. on a development board), or want to use of the more advanced WA functionality, further configuration will be required.

2.3.1 Android

General Device Setup

You can specify the device interface by setting `device` setting in `~/.workload_automation/config.py`. Available interfaces can be viewed by running `wa list devices` command. If you don't see your specific device listed (which is likely unless you're using one of the ARM-supplied platforms), then you should use `generic_android` interface (this is set in the config by default).

```
device = 'generic_android'
```

The device interface may be configured through `device_config` setting, who's value is a `dict` mapping setting names to their values. You can find the full list of available parameter by looking up your device interface in the *Devices* section of the documentation. Some of the most common parameters you might want to change are outlined below.

`adb_name`
: If you have multiple Android devices connected to the host machine, you will need to set this to indicate to WA which device you want it to use.

`working_directory`
: WA needs a "working" directory on the device which it will use for collecting traces, caching assets it pushes to the device, etc. By default, it will create one under `/sdcard` which should be mapped and writable on standard Android builds. If this is not the case for your device, you will need to specify an alternative working directory (e.g. under `/data/local`).

scheduler
: This specifies the scheduling mechanism (from the perspective of core layout) utilized by the device). For recent big.LITTLE devices, this should generally be "hmp" (ARM Hetrogeneous Mutli-Processing); some legacy development platforms might have Linaro IKS kernels, in which case it should be "iks". For homogeneous (single-cluster) devices, it should be "smp". Please see `scheduler` parameter in the `generic_android` device documentation for more details.

core_names
: This and `core_clusters` need to be set if you want to utilize some more advanced WA functionality (like setting of core-related runtime parameters such as governors, frequencies, etc). `core_names` should be a list of core names matching the order in which they are exposed in sysfs. For example, ARM TC2 SoC is a 2x3 big.LITTLE system; its core_names would be `['a7', 'a7', 'a7', 'a15', 'a15']`, indicating that cpu0-cpu2 in cpufreq sysfs structure are A7's and cpu3 and cpu4 are A15's.

core_clusters
: If `core_names` is defined, this must also be defined. This is a list of integer values indicating the cluster the corresponding core in `cores_names` belongs to. For example, for TC2, this would be `[0, 0, 0, 1, 1]`, indicating that A7's are on cluster 0 and A15's are on cluster 1.

A typical `device_config` inside `config.py` may look something like

```
device_config = dict(
        'adb_name'='0123456789ABCDEF',
        'working_direcory'='/sdcard/wa-working',
        'core_names'=['a7', 'a7', 'a7', 'a15', 'a15'],
        'core_clusters'=[0, 0, 0, 1, 1],
        # ...
)
```

Configuring Android

There are a few additional tasks you may need to perform once you have a device booted into Android (especially if this is an initial boot of a fresh OS deployment):

- You have gone through FTU (first time usage) on the home screen and in the apps menu.
- You have disabled the screen lock.
- You have set sleep timeout to the highest possible value (30 mins on most devices).
- You have disabled brightness auto-adjust and have set the brightness to a fixed level.
- You have set the locale language to "English" (this is important for some workloads in which UI automation looks for specific text in UI elements).

TC2 Setup

This section outlines how to setup ARM TC2 development platform to work with WA.

Pre-requisites

You can obtain the full set of images for TC2 from Linaro:

https://releases.linaro.org/latest/android/vexpress-lsk.

For the easiest setup, follow the instructions on the "Firmware" and "Binary Image Installation" tabs on that page.

Note: The default `reboot_policy` in `config.py` is to not reboot. With this WA will assume that the device is already booted into Android prior to WA being invoked. If you want to WA to do the initial boot of the TC2, you will have to change reboot policy to at least `initial`.

Setting Up Images

Note: Make sure that both DIP switches near the black reset button on TC2 are up (this is counter to the Linaro guide that instructs to lower one of the switches).

Note: The TC2 must have an Ethernet connection.

If you have followed the setup instructions on the Linaro page, you should have a USB stick or an SD card with the file system, and internal microSD on the board (VEMSD) with the firmware images. The default Linaro configuration is to boot from the image on the boot partition in the file system you have just created. This is not supported by WA, which expects the image to be in NOR flash on the board. This requires you to copy the images from the boot partition onto the internal microSD card.

Assuming the boot partition of the Linaro file system is mounted on `/media/boot` and the internal microSD is mounted on `/media/VEMSD`, copy the following images:

```
cp /media/boot/zImage /media/VEMSD/SOFTWARE/kern_mp.bin
cp /media/boot/initrd /media/VEMSD/SOFTWARE/init_mp.bin
cp /media/boot/v2p-ca15-tc2.dtb /media/VEMSD/SOFTWARE/mp_a7bc.dtb
```

Optionally

The default device tree configuration the TC2 is to boot on the A7 cluster. It is also possible to configure the device tree to boot on the A15 cluster, or to boot with one of the clusters disabled (turning TC2 into an A7-only or A15-only device). Please refer to the "Firmware" tab on the Linaro paged linked above for instructions on how to compile the appropriate device tree configurations.

WA allows selecting between these configurations using `os_mode` boot parameter of the TC2 device interface. In order for this to work correctly, device tree files for the A15-bootcluster, A7-only and A15-only configurations should be copied into `/media/VEMSD/SOFTWARE/` as `mp_a15bc.dtb`, `mp_a7.dtb` and `mp_a15.dtb` respectively.

This is entirely optional. If you're not planning on switching boot cluster configuration, those files do not need to be present in VEMSD.

config.txt

Also, make sure that `USB_REMOTE` setting in `/media/VEMSD/config.txt` is set to `TRUE` (this will allow rebooting the device by writing reboot.txt to VEMSD).

```
USB_REMOTE: TRUE                ;Selects remote command via USB
```

TC2-specific device_config settings

There are a few settings that may need to be set in `device_config` inside your `config.py` which are specific to TC2:

Note: TC2 *does not* accept most "standard" android `device_config` settings.

adb_name If you're running WA with reboots disabled (which is the default reboot policy), you will need to manually run `adb connect` with TC2's IP address and set this.

root_mount WA expects TC2's internal microSD to be mounted on the host under `/media/VEMSD`. If this location is different, it needs to be specified using this setting.

boot_firmware WA defaults to try booting using UEFI, which will require some additional firmware from ARM that may not be provided with Linaro releases (see the UEFI and PSCI section below). If you do not have those images, you will need to set `boot_firmware` to `bootmon`.

fs_medium TC2's file system can reside either on an SD card or on a USB stick. Boot configuration is different depending on this. By default, WA expects it to be on `usb`; if you are using and SD card, you should set this to `sd`.

bm_image Bootmon image that comes as part of TC2 firmware periodically gets updated. At the time of the release, `bm_v519r.axf` was used by ARM. If you are using a more recent image, you will need to set this indicating the image name (just the name of the actual file, *not* the path). Note: this setting only applies if using `bootmon` boot firmware.

serial_device WA will assume TC2 is connected on `/dev/ttyS0` by default. If the serial port is different, you will need to set this.

UEFI and PSCI

UEFI is a boot firmware alternative to bootmon. Currently UEFI is coupled with PSCI (Power State Coordination Interface). That means that in order to use PSCI, UEFI has to be the boot firmware. Currently the reverse dependency is true as well (for TC2). Therefore using UEFI requires enabling PSCI.

In case you intend to use uefi/psci mode instead of bootmon, you will need two additional files: tc2_sec.bin and tc2_uefi.bin. after obtaining those files, place them inside /media/VEMSD/SOFTWARE/ directory as such:

```
cp tc2_sec.bin /media/VEMSD/SOFTWARE/
cp tc2_uefi.bin /media/VEMSD/SOFTWARE/
```

Juno Setup

Note: At the time of writing, the Android software stack on Juno was still very immature. Some workloads may not run, and there maybe stability issues with the device.

The full software stack can be obtained from Linaro:

https://releases.linaro.org/14.08/members/arm/android/images/armv8-android-juno-lsk

Please follow the instructions on the "Binary Image Installation" tab on that page. More up-to-date firmware and kernel may also be obtained by registered members from ARM Connected Community: http://www.arm.com/community/ (though this is not guaranteed to work with the Linaro file system).

UEFI

Juno uses UEFI to boot the kernel image. UEFI supports multiple boot configurations, and presents a menu on boot to select (in default configuration it will automatically boot the first entry in the menu if not interrupted before a timeout). WA will look for a specific entry in the UEFI menu (`'WA'` by default, but that may be changed by setting `uefi_entry` in the `device_config`). When following the UEFI instructions on the above Linaro page, please make sure to name the entry appropriately (or to correctly set the `uefi_entry`).

There are two supported way for Juno to discover kernel images through UEFI. It can either load them from NOR flash on the board, or form boot partition on the file system. The setup described on the Linaro page uses the boot partition method.

If WA does not find the UEFI entry it expects, it will create one. However, it will assume that the kernel image resides in NOR flash, which means it will not work with Linaro file system. So if you're replicating the Linaro setup exactly, you will need to create the entry manually, as outline on the above-linked page.

Rebooting

At the time of writing, normal Android reboot did not work properly on Juno Android, causing the device to crash into an irrecoverable state. Therefore, WA will perform a hard reset to reboot the device. It will attempt to do this by toggling the DTR line on the serial connection to the device. In order for this to work, you need to make sure that SW1 configuration switch on the back panel of the board (the right-most DIP switch) is toggled *down*.

2.3.2 Linux

General Device Setup

You can specify the device interface by setting `device` setting in `~/.workload_automation/config.py`. Available interfaces can be viewed by running `wa list devices` command. If you don't see your specific device listed (which is likely unless you're using one of the ARM-supplied platforms), then you should use `generic_linux` interface (this is set in the config by default).

```
device = 'generic_linux'
```

The device interface may be configured through `device_config` setting, who's value is a `dict` mapping setting names to their values. You can find the full list of available parameter by looking up your device interface in the *Devices* section of the documentation. Some of the most common parameters you might want to change are outlined below.

Currently, the only only supported method for talking to a Linux device is over SSH. Device configuration must specify the parameters need to establish the connection.

`host`
: This should be either the the DNS name or IP address of the device.

`username`
: The login name of the user on the device that WA will use. This user should have a home directory (unless an alternative working directory is specified using `working_directory` config – see below), and, for full functionality, the user should have sudo rights (WA will be able to use sudo-less acounts but some instruments or workload may not work).

`password`
: Password for the account on the device. Either this of a `keyfile` (see below) must be specified.

keyfile
: If key-based authentication is used, this may be used to specify the SSH identity file instead of the password.

property_files
: This is a list of paths that will be pulled for each WA run into the __meta subdirectory in the results. The intention is to collect meta-data about the device that may aid in reporducing the results later. The paths specified do not have to exist on the device (they will be ignored if they do not). The default list is `['/proc/version', '/etc/debian_version', '/etc/lsb-release', '/etc/arch-release']`

In addition, `working_directory`, `scheduler`, `core_names`, and `core_clusters` can also be specified and have the same meaning as for Android devices (see above).

A typical `device_config` inside `config.py` may look something like

```
device_config = dict(
        host='192.168.0.7',
        username='guest',
        password='guest',
        core_names=['a7', 'a7', 'a7', 'a15', 'a15'],
        core_clusters=[0, 0, 0, 1, 1],
        # ...
)
```

2.3.3 Related Settings

Reboot Policy

This indicates when during WA execution the device will be rebooted. By default this is set to `never`, indicating that WA will not reboot the device. Please see `reboot_policy` documentation in *Configuration* for

more details.

Execution Order

`execution_order` defines the order in which WA will execute workloads. `by_iteration` (set by default) will execute the first iteration of each spec first, followed by the second iteration of each spec (that defines more than one iteration) and so forth. The alternative will loop through all iterations for the first first spec first, then move on to second spec, etc. Again, please see *Configuration* for more details.

2.3.4 Adding a new device interface

If you are working with a particularly unusual device (e.g. a early stage development board) or need to be able to handle some quirk of your Android build, configuration available in `generic_android` interface may not be enough for you. In that case, you may need to write a custom interface for your device. A device interface is an `Extension` (a plug-in) type in WA and is implemented similar to other extensions (such as workloads or instruments). Pleaser refer to *Adding a Device* section for information on how this may be done.

2.4 Commands

Installing the wlauto package will add `wa` command to your system, which you can run from anywhere. This has a number of sub-commands, which can be viewed by executing

```
wa -h
```

Individual sub-commands are discussed in detail below.

2.4.1 run

The most common sub-command you will use is `run`. This will run specified workload(s) and process resulting output. This takes a single mandatory argument that specifies what you want WA to run. This could be either a workload name, or a path to an "agenda" file that allows to specify multiple workloads as well as a lot additional configuration (see *Agenda* section for details). Executing

```
wa run -h
```

Will display help for this subcommand that will look something like this:

```
usage: run [-d DIR] [-f] AGENDA

Execute automated workloads on a remote device and process the resulting
output.

positional arguments:
  AGENDA                Agenda for this workload automation run. This defines
                        which workloads will be executed, how many times, with
                        which tunables, etc. See /usr/local/lib/python2.7
                        /dist-packages/wlauto/agenda-example.csv for an
                        example of how this file should be structured.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        specify an additional config.py
  -v, --verbose         The scripts will produce verbose output.
  --version             Output the version of Workload Automation and exit.
  --debug               Enable debug mode. Note: this implies --verbose.
  -d DIR, --output-directory DIR
                        Specify a directory where the output will be
                        generated. If the directory already exists, the script
                        will abort unless -f option (see below) is used,in
                        which case the contents of the directory will be
                        overwritten. If this option is not specified, then
                        wa_output will be used instead.
  -f, --force           Overwrite output directory if it exists. By default,
                        the script will abort in this situation to prevent
                        accidental data loss.
  -i ID, --id ID        Specify a workload spec ID from an agenda to run. If
                        this is specified, only that particular spec will be
                        run, and other workloads in the agenda will be
                        ignored. This option may be used to specify multiple
                        IDs.
```

Output Directory

The exact contents on the output directory will depend on configuration options used, instrumentation and output processors enabled, etc. Typically, the output directory will contain a results file at the top level that lists all measure-

ments that were collected (currently, csv and json formats are supported), along with a subdirectory for each iteration executed with output for that specific iteration.

At the top level, there will also be a run.log file containing the complete log output for the execution. The contents of this file is equivalent to what you would get in the console when using –verbose option.

Finally, there will be a __meta subdirectory. This will contain a copy of the agenda file used to run the workloads along with any other device-specific configuration files used during execution.

2.4.2 create

This can be used to create various WA-related objects, currently workloads, packages and agendas. The full set of options for this command are:

```
usage: wa create [-h] [-c CONFIG] [-v] [--debug] [--version]
                 {workload,package,agenda} ...

positional arguments:
  {workload,package,agenda}
    workload            Create a new workload. By default, a basic workload
                        template will be used but you can use options to
                        specify a different template.
    package             Create a new empty Python package for WA extensions.
                        On installation, this package will "advertise" itself
                        to WA so that Extensions with in it will be loaded by
                        WA when it runs.
    agenda              Create an agenda whit the specified extensions
                        enabled. And parameters set to their default values.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        specify an additional config.py
  -v, --verbose         The scripts will produce verbose output.
  --debug               Enable debug mode. Note: this implies --verbose.
  --version             show program's version number and exit
```

Use “wa create <object> -h” to see all the object-specific arguments. For example:

```
wa create agenda -h
```

will display the relevant options that can be used to create an agenda.

2.4.3 get-assets

This command can download external extension dependencies used by Workload Automation. It can be used to download assets for all available extensions or those specificity listed. The full set of options for this command are:

```
usage: wa get-assets [-h] [-c CONFIG] [-v] [--debug] [--version] [-f]
                     [--url URL] (-a | -e EXT [EXT ...])

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        specify an additional config.py
  -v, --verbose         The scripts will produce verbose output.
```

```
  --debug               Enable debug mode. Note: this implies --verbose.
  --version             show program's version number and exit
  -f, --force           Always fetch the assets, even if matching versions
                        exist in local cache.
  --url URL             The location from which to download the files. If not
                        provided, config setting ``remote_assets_url`` will be
                        used if available, else uses the default
                        REMOTE_ASSETS_URL parameter in the script.
  -a, --all             Download assets for all extensions found in the index.
                        Cannot be used with -e.
  -e EXT [EXT ...]      One or more extensions whose assets to download.
                        Cannot be used with --all.
```

2.4.4 list

This lists all extensions of a particular type. For example:

```
wa list workloads
```

will list all workloads currently included in WA. The list will consist of extension names and short descriptions of the functionality they offer.

2.4.5 show

This will show detailed information about an extension, including more in-depth description and any parameters/configuration that are available. For example executing:

```
wa show andebench
```

will produce something like:

```
andebench

AndEBench is an industry standard Android benchmark provided by The Embedded␣
↪Microprocessor Benchmark Consortium
(EEMBC).

parameters:

number_of_threads
Number of threads that will be spawned by AndEBench.
        type: int

single_threaded
If ``true``, AndEBench will run with a single thread. Note: this must not be␣
↪specified if ``number_of_threads``
has been specified.
        type: bool

http://www.eembc.org/andebench/about.php

From the website:

- Initial focus on CPU and Dalvik interpreter performance
- Internal algorithms concentrate on integer operations
```

```
- Compares the difference between native and Java performance
- Implements flexible multicore performance analysis
- Results displayed in Iterations per second
- Detailed log file for comprehensive engineering analysis
```

2.4.6 record

This command simplifies the process of recording an revent file. It will automatically deploy revent and even has the option of automatically opening apps. WA uses two parts to the names of revent recordings in the format, {device_name}.{suffix}.revent. - device_name can either be specified manually with the `-d` argument or it can be automatically determined. On Android device it will be obtained from `build.prop`, on Linux devices it is obtained from `/proc/device-tree/model`. - suffix is used by WA to determine which part of the app execution the recording is for, currently these are either `setup` or `run`. This should be specified with the `-s` argument. The full set of options for this command are:

```
usage: wa record [-h] [-c CONFIG] [-v] [--debug] [--version] [-d DEVICE]
             [-s SUFFIX] [-o OUTPUT] [-p PACKAGE] [-g] [-C]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        specify an additional config.py
  -v, --verbose         The scripts will produce verbose output.
  --debug               Enable debug mode. Note: this implies --verbose.
  --version             show program's version number and exit
  -d DEVICE, --device DEVICE
                        The name of the device
  -s SUFFIX, --suffix SUFFIX
                        The suffix of the revent file, e.g. ``setup``
  -o OUTPUT, --output OUTPUT
                        Directory to save the recording in
  -p PACKAGE, --package PACKAGE
                        Package to launch before recording
  -g, --gamepad         Record from a gamepad rather than all devices.
  -C, --clear           Clear app cache before launching it
```

2.4.7 replay

Along side `record` wa also has a command to playback recorded revent files. It behaves very similar to the `record` command taking many of the same options:

```
usage: wa replay [-h] [-c CONFIG] [-v] [--debug] [--version] [-p PACKAGE] [-C]
             revent

positional arguments:
  revent                The name of the file to replay

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        specify an additional config.py
  -v, --verbose         The scripts will produce verbose output.
  --debug               Enable debug mode. Note: this implies --verbose.
  --version             show program's version number and exit
```

```
  -p PACKAGE, --package PACKAGE
                        Package to launch before recording
  -C, --clear           Clear app cache before launching it
```

2.5 Agenda

An agenda specifies what is to be done during a Workload Automation run, including which workloads will be run, with what configuration, which instruments and result processors will be enabled, etc. Agenda syntax is designed to be both succinct and expressive.

Agendas are specified using YAML notation. It is recommended that you familiarize yourself with the linked page.

Note: Earlier versions of WA have supported CSV-style agendas. These were there to facilitate transition from WA1 scripts. The format was more awkward and supported only a limited subset of the features. Support for it has now been removed.

2.5.1 Specifying which workloads to run

The central purpose of an agenda is to specify what workloads to run. A minimalist agenda contains a single entry at the top level called "workloads" that maps onto a list of workload names to run:

```
workloads:
        - dhrystone
        - memcpy
        - cyclictest
```

This specifies a WA run consisting of `dhrystone` followed by `memcpy`, followed by `cyclictest` workloads, and using instruments and result processors specified in config.py (see *Configuration* section).

Note: If you're familiar with YAML, you will recognize the above as a single-key associative array mapping onto a list. YAML has two notations for both associative arrays and lists: block notation (seen above) and also in-line notation. This means that the above agenda can also be written in a single line as

```
workloads: [dhrystone, memcpy, cyclictest]
```

(with the list in-lined), or

```
{workloads: [dhrystone, memcpy, cyclictest]}
```

(with both the list and the associative array in-line). WA doesn't care which of the notations is used as they all get parsed into the same structure by the YAML parser. You can use whatever format you find easier/clearer.

Multiple iterations

There will normally be some variability in workload execution when running on a real device. In order to quantify it, multiple iterations of the same workload are usually performed. You can specify the number of iterations for each workload by adding `iterations` field to the workload specifications (or "specs"):

```
workloads:
        - name: dhrystone
          iterations: 5
        - name: memcpy
          iterations: 5
        - name: cyclictest
          iterations: 5
```

Now that we're specifying both the workload name and the number of iterations in each spec, we have to explicitly name each field of the spec.

It is often the case that, as in in the example above, you will want to run all workloads for the same number of iterations. Rather than having to specify it for each and every spec, you can do with a single entry by adding a `global` section to your agenda:

```
global:
        iterations: 5
workloads:
        - dhrystone
        - memcpy
        - cyclictest
```

The global section can contain the same fields as a workload spec. The fields in the global section will get added to each spec. If the same field is defined both in global section and in a spec, then the value in the spec will overwrite the global value. For example, suppose we wanted to run all our workloads for five iterations, except cyclictest which we want to run for ten (e.g. because we know it to be particularly unstable). This can be specified like this:

```
global:
        iterations: 5
workloads:
        - dhrystone
        - memcpy
        - name: cyclictest
          iterations: 10
```

Again, because we are now specifying two fields for cyclictest spec, we have to explicitly name them.

Configuring workloads

Some workloads accept configuration parameters that modify their behavior. These parameters are specific to a particular workload and can alter the workload in any number of ways, e.g. set the duration for which to run, or specify a media file to be used, etc. The vast majority of workload parameters will have some default value, so it is only necessary to specify the name of the workload in order for WA to run it. However, sometimes you want more control over how a workload runs.

For example, by default, dhrystone will execute 10 million loops across four threads. Suppose you device has six cores available and you want the workload to load them all. You also want to increase the total number of loops accordingly to 15 million. You can specify this using dhrystone's parameters:

```
global:
        iterations: 5
workloads:
        - name: dhrystone
          params:
                threads: 6
                mloops: 15
```

```
        - memcpy
        - name: cyclictest
          iterations: 10
```

Note: You can find out what parameters a workload accepts by looking it up in the *Workloads* section. You can also look it up using WA itself with "show" command:

```
wa show dhrystone
```

see the *Invocation* section for details.

In addition to configuring the workload itself, we can also specify configuration for the underlying device. This can be done by setting runtime parameters in the workload spec. For example, suppose we want to ensure the maximum score for our benchmarks, at the expense of power consumption, by setting the cpufreq governor to "performance" on cpu0 (assuming all our cores are in the same DVFS domain and so setting the governor for cpu0 will affect all cores). This can be done like this:

```
global:
        iterations: 5
workloads:
        - name: dhrystone
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
        - memcpy
        - name: cyclictest
          iterations: 10
```

Here, we're specifying `sysfile_values` runtime parameter for the device. The value for this parameter is a mapping (an associative array, in YAML) of file paths onto values that should be written into those files. `sysfile_values` is the only runtime parameter that is available for any (Linux) device. Other runtime parameters will depend on the specifics of the device used (e.g. its CPU cores configuration). I've renamed `params` to `workload_params` for clarity, but that wasn't strictly necessary as `params` is interpreted as `workload_params` inside a workload spec.

Note: `params` field is interpreted differently depending on whether it's in a workload spec or the global section. In a workload spec, it translates to `workload_params`, in the global section it translates to `runtime_params`.

Runtime parameters do not automatically reset at the end of workload spec execution, so all subsequent iterations will also be affected unless they explicitly change the parameter (in the example above, performance governor will also be used for `memcpy` and `cyclictest`. There are two ways around this: either set `reboot_policy` WA setting (see *Configuration* section) such that the device gets rebooted between spec executions, thus being returned to its initial state, or set the default runtime parameter values in the `global` section of the agenda so that they get set for every spec that doesn't explicitly override them.

Note: "In addition to `runtime_params` there are also `boot_params` that work in a similar way, but they get passed to the device when it reboots. At the moment `TC2` is the only device that defines a boot parameter, which is

explained in TC2 documentation, so boot parameters will not be mentioned further.

IDs and Labels

It is possible to list multiple specs with the same workload in an agenda. You may wish to this if you want to run a workload with different parameter values or under different runtime configurations of the device. The workload name therefore does not uniquely identify a spec. To be able to distinguish between different specs (e.g. in reported results), each spec has an ID which is unique to all specs within an agenda (and therefore with a single WA run). If an ID isn't explicitly specified using `id` field (note that the field name is in lower case), one will be automatically assigned to the spec at the beginning of the WA run based on the position of the spec within the list. The first spec *without an explicit ID* will be assigned ID `1`, the second spec *without an explicit ID* will be assigned ID `2`, and so forth.

Numerical IDs aren't particularly easy to deal with, which is why it is recommended that, for non-trivial agendas, you manually set the ids to something more meaningful (or use labels – see below). An ID can be pretty much anything that will pass through the YAML parser. The only requirement is that it is unique to the agenda. However, is usually better to keep them reasonably short (they don't need to be *globally* unique), and to stick with alpha-numeric characters and underscores/dashes. While WA can handle other characters as well, getting too adventurous with your IDs may cause issues further down the line when processing WA results (e.g. when uploading them to a database that may have its own restrictions).

In addition to IDs, you can also specify labels for your workload specs. These are similar to IDs but do not have the uniqueness restriction. If specified, labels will be used by some result processes instead of (or in addition to) the workload name. For example, the `csv` result processor will put the label in the "workload" column of the CSV file.

It is up to you how you chose to use IDs and labels. WA itself doesn't expect any particular format (apart from uniqueness for IDs). Below is the earlier example updated to specify explicit IDs and label dhrystone spec to reflect parameters used.

```
global:
        iterations: 5
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

2.5.2 Result Processors and Instrumentation

Result Processors

Result processors, as the name suggests, handle the processing of results generated form running workload specs. By default, WA enables a couple of basic result processors (e.g. one generates a csv file with all scores reported by workloads), which you can see in `~/.workload_automation/config.py`. However, WA has a number of

other, more specialized, result processors (e.g. for uploading to databases). You can list available result processors with `wa list result_processors` command. If you want to permanently enable a result processor, you can add it to your `config.py`. You can also enable a result processor for a particular run by specifying it in the `config` section in the agenda. As the name suggests, `config` section mirrors the structure of `config.py`(although using YAML rather than Python), and anything that can be specified in the latter, can also be specified in the former.

As with workloads, result processors may have parameters that define their behavior. Parameters of result processors are specified a little differently, however. Result processor parameter values are listed in the config section, namespaced under the name of the result processor.

For example, suppose we want to be able to easily query the results generated by the workload specs we've defined so far. We can use `sqlite` result processor to have WA create an sqlite database file with the results. By default, this file will be generated in WA's output directory (at the same level as results.csv); but suppose we want to store the results in the same file for every run of the agenda we do. This can be done by specifying an alternative database file with `database` parameter of the result processor:

```
config:
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

A couple of things to observe here:

- There is no need to repeat the result processors listed in `config.py`. The processors listed in `result_processors` entry in the agenda will be used *in addition to* those defined in the `config.py`.
- The database file is specified under "sqlite" entry in the config section. Note, however, that this entry alone is not enough to enable the result processor, it must be listed in `result_processors`, otherwise the "sqilte" config entry will be ignored.
- The database file must be specified as an absolute path, however it may use the user home specifier '~' and/or environment variables.

Instrumentation

WA can enable various "instruments" to be used during workload execution. Instruments can be quite diverse in their functionality, but the majority of instruments available in WA today are there to collect additional data (such as trace) from the device during workload execution. You can view the list of available instruments by using `wa list instruments` command. As with result processors, a few are enabled by default in the `config.py` and additional ones may be added in the same place, or specified in the agenda using `instrumentation` entry.

For example, we can collect core utilisation statistics (for what proportion of workload execution N cores were utilized above a specified threshold) using `coreutil` instrument.

```
config:
        instrumentation: [coreutil]
        coreutil:
                threshold: 80
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

Instrumentation isn't "free" and it is advisable not to have too many instruments enabled at once as that might skew results. For example, you don't want to have power measurement enabled at the same time as event tracing, as the latter may prevent cores from going into idle states and thus affecting the reading collected by the former.

Unlike result processors, instrumentation may be enabled (and disabled – see below) on per-spec basis. For example, suppose we want to collect /proc/meminfo from the device when we run `memcpy` workload, but not for the other two. We can do that using `sysfs_extractor` instrument, and we will only enable it for `memcpy`:

```
config:
        instrumentation: [coreutil]
        coreutil:
                threshold: 80
        sysfs_extractor:
                paths: [/proc/meminfo]
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
```

```
        - id: 02_memc
          name: memcpy
          instrumentation: [sysfs_extractor]
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

As with `config` sections, `instrumentation` entry in the spec needs only to list additional instruments and does not need to repeat instruments specified elsewhere.

Note: At present, it is only possible to enable/disable instrumentation on per-spec base. It is *not* possible to provide configuration on per-spec basis in the current version of WA (e.g. in our example, it is not possible to specify different `sysfs_extractor` paths for different workloads). This restriction may be lifted in future versions of WA.

Disabling result processors and instrumentation

As seen above, extensions specified with `instrumentation` and `result_processor` clauses get added to those already specified previously. Just because an instrument specified in `config.py` is not listed in the `config` section of the agenda, does not mean it will be disabled. If you do want to disable an instrument, you can always remove/comment it out from `config.py`. However that will be introducing a permanent configuration change to your environment (one that can be easily reverted, but may be just as easily forgotten). If you want to temporarily disable a result processor or an instrument for a particular run, you can do that in your agenda by prepending a tilde (~) to its name.

For example, let's say we want to disable `cpufreq` instrument enabled in our `config.py` (suppose we're going to send results via email and so want to reduce to total size of the output directory):

```
config:
        instrumentation: [coreutil, ~cpufreq]
        coreutil:
                threshold: 80
        sysfs_extractor:
                paths: [/proc/meminfo]
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
          instrumentation: [sysfs_extractor]
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

2.5.3 Sections

It is a common requirement to be able to run the same set of workloads under different device configurations. E.g. you may want to investigate impact of changing a particular setting to different values on the benchmark scores, or to quantify the impact of enabling a particular feature in the kernel. WA allows this by defining "sections" of configuration with an agenda.

For example, suppose what we really want, is to measure the impact of using interactive cpufreq governor vs the performance governor on the three benchmarks. We could create another three workload spec entries similar to the ones we already have and change the sysfile value being set to "interactive". However, this introduces a lot of duplication; and what if we want to change spec configuration? We would have to change it in multiple places, running the risk of forgetting one.

A better way is to keep the three workload specs and define a section for each governor:

```
config:
        instrumentation: [coreutil, ~cpufreq]
        coreutil:
                threshold: 80
        sysfs_extractor:
                paths: [/proc/meminfo]
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
sections:
        - id: perf
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
        - id: inter
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪interactive
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
          instrumentation: [sysfs_extractor]
        - id: 03_cycl
          name: cyclictest
          iterations: 10
```

A section, just like an workload spec, needs to have a unique ID. Apart from that, a "section" is similar to the `global` section we've already seen – everything that goes into a section will be applied to each workload spec. Workload specs defined under top-level `workloads` entry will be executed for each of the sections listed under `sections`.

Note: It is also possible to have a `workloads` entry within a section, in which case, those workloads will only be executed for that specific section.

In order to maintain the uniqueness requirement of workload spec IDs, they will be namespaced under each section by prepending the section ID to the spec ID with an under score. So in the agenda above, we no longer have a workload spec with ID `01_dhry`, instead there are two specs with IDs `perf_01_dhry` and `inter_01_dhry`.

Note that the `global` section still applies to every spec in the agenda. So the precedence order is – spec settings override section settings, which in turn override global settings.

2.5.4 Other Configuration

As mentioned previously, `config` section in an agenda can contain anything that can be defined in `config.py` (with Python syntax translated to the equivalent YAML). Certain configuration (e.g. `run_name`) makes more sense to define in an agenda than a config file. Refer to the *Configuration* section for details.

```
config:
        project: governor_comparison
        run_name: performance_vs_interactive

        device: generic_android
        reboot_policy: never

        instrumentation: [coreutil, ~cpufreq]
        coreutil:
                threshold: 80
        sysfs_extractor:
                paths: [/proc/meminfo]
        result_processors: [sqlite]
        sqlite:
                database: ~/my_wa_results.sqlite
global:
        iterations: 5
sections:
        - id: perf
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪performance
        - id: inter
          runtime_params:
                sysfile_values:
                        /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor:␣
↪interactive
workloads:
        - id: 01_dhry
          name: dhrystone
          label: dhrystone_15over6
          workload_params:
                threads: 6
                mloops: 15
        - id: 02_memc
          name: memcpy
          instrumentation: [sysfs_extractor]
        - id: 03_cycl
```

```
        name: cyclictest
        iterations: 10
```

2.6 Configuration

In addition to specifying run execution parameters through an agenda, the behavior of WA can be modified through configuration file(s). The default configuration file is `~/.workload_automation/config.py` (the location can be changed by setting `WA_USER_DIRECTORY` environment variable, see *Environment Variables* section below). This file will be created when you first run WA if it does not already exist. This file must always exist and will always be loaded. You can add to or override the contents of that file on invocation of Workload Automation by specifying an additional configuration file using `--config` option.

The config file is just a Python source file, so it can contain any valid Python code (though execution of arbitrary code through the config file is discouraged). Variables with specific names will be picked up by the framework and used to modify the behavior of Workload automation.

Note: As of version 2.1.3 is also possible to specify the following configuration in the agenda. See *configuration in an agenda*.

2.6.1 Available Settings

Note: Extensions such as workloads, instrumentation or result processors may also pick up certain settings from this file, so the list below is not exhaustive. Please refer to the documentation for the specific extensions to see what settings they accept.

device
: This setting defines what specific Device subclass will be used to interact the connected device. Obviously, this must match your setup.

device_config
: This must be a Python dict containing setting-value mapping for the configured `device`. What settings and values are valid is specific to each device. Please refer to the documentation for your device.

reboot_policy
: This defines when during execution of a run the Device will be rebooted. The possible values are:

  "never" The device will never be rebooted.

  "initial" The device will be rebooted when the execution first starts, just before executing the first workload spec.

  "each_spec" The device will be rebooted before running a new workload spec. Note: this acts the same as each_iteration when execution order is set to by_iteration

  "each_iteration" The device will be rebooted before each new iteration.

  See also:

  Framework Overview

execution_order
: Defines the order in which the agenda spec will be executed. At the moment, the following execution orders are supported:

"by_iteration" The first iteration of each workload spec is executed one after the other, so all workloads are executed before proceeding on to the second iteration. E.g. A1 B1 C1 A2 C2 A3. This is the default if no order is explicitly specified.

In case of multiple sections, this will spread them out, such that specs from the same section are further part. E.g. given sections X and Y, global specs A and B, and two iterations, this will run

```
X.A1, Y.A1, X.B1, Y.B1, X.A2, Y.A2, X.B2, Y.B2
```

"by_section" Same as `"by_iteration"`, however this will group specs from the same section together, so given sections X and Y, global specs A and B, and two iterations, this will run

```
X.A1, X.B1, Y.A1, Y.B1, X.A2, X.B2, Y.A2, Y.B2
```

"by_spec" All iterations of the first spec are executed before moving on to the next spec. E.g. A1 A2 A3 B1 C1 C2 This may also be specified as `"classic"`, as this was the way workloads were executed in earlier versions of WA.

"random" Execution order is entirely random.

Added in version 2.1.5.

retry_on_status
This is list of statuses on which a job will be cosidered to have failed and will be automatically retried up to `max_retries` times. This defaults to `["FAILED", "PARTIAL"]` if not set. Possible values are:

`"OK"` This iteration has completed and no errors have been detected

`"PARTIAL"` One or more instruments have failed (the iteration may still be running).

`"FAILED"` The workload itself has failed.

`"ABORTED"` The user interupted the workload

max_retries
The maximum number of times failed jobs will be retried before giving up. If not set, this will default to `3`.

Note: this number does not include the original attempt

---

instrumentation
This should be a list of instruments to be enabled during run execution. Values must be names of available instruments. Instruments are used to collect additional data, such as energy measurements or execution time, during runs.

See also:

wlauto.instrumentation package

result_processors
This should be a list of result processors to be enabled during run execution. Values must be names of available result processors. Result processor define how data is output from WA.

See also:

wlauto.result_processors package

logging
A dict that contains logging setting. At the moment only three settings are supported:

"file format" Controls how logging output appears in the run.log file in the output directory.

"verbose format" Controls how logging output appear on the console when `--verbose` flag was used.

"regular format" Controls how logging output appear on the console when `--verbose` flag was not used.

All three values should be Python old-style format strings specifying which log record attributes should be displayed.

remote_assets_path
Path to the local mount of a network assets repository. See *Maintaining Centralized Assets Repository*.

There are also a couple of settings are used to provide additional metadata for a run. These may get picked up by instruments or result processors to attach context to results.

project
A string naming the project for which data is being collected. This may be useful, e.g. when uploading data to a shared database that is populated from multiple projects.

project_stage
A dict or a string that allows adding additional identifier. This is may be useful for long-running projects.

run_name
A string that labels the WA run that is bing performed. This would typically be set in the `config` section of an agenda (see *configuration in an agenda*) rather than in the config file.

2.6.2 Environment Variables

In addition to standard configuration described above, WA behaviour can be altered through environment variables. These can determine where WA looks for various assets when it starts.

WA_USER_DIRECTORY
This is the location WA will look for config.py, inustrumentation , and it will also be used for local caches, etc. If this variable is not set, the default location is `~/.workload_automation` (this is created when WA is installed).

Note: This location **must** be writable by the user who runs WA.

WA_EXTENSION_PATHS
By default, WA will look for extensions in its own package and in subdirectories under `WA_USER_DIRECTORY`. This environment variable can be used specify a colon-separated list of additional locations WA should use to look for extensions.

CHAPTER 3

Extensions

This section lists extensions that currently come with WA2. Each package below represents a particular type of extension (e.g. a workload); each sub-package of that package is a particular instance of that extension (e.g. the Andebench workload). Clicking on a link will show what the individual extension does, what configuration parameters it takes, etc.

For how to implement you own extensions, please refer to the guides in the *In-depth* section.

3.1 Workloads

3.1.1 adobereader

The Adobe Reader workflow carries out the following typical productivity tasks.

Test description:

1. Open a local file on the device
2. **Gestures test:** 2.1. Swipe down across the central 50% of the screen in 200 x 5ms steps 2.2. Swipe up across the central 50% of the screen in 200 x 5ms steps 2.3. Swipe right from the edge of the screen in 50 x 5ms steps 2.4. Swipe left from the edge of the screen in 50 x 5ms steps 2.5. Pinch out 50% in 100 x 5ms steps 2.6. Pinch In 50% in 100 x 5ms steps
3. **Search test:** Search `document_name` for each string in the `search_string_list`
4. Close the document

Known working APK version: 16.1

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

document_name [str] The document name to use for the Gesture and Search test.

default: `'uxperf_test_doc.pdf'`

search_string_list [list_of_strs] For each string in the list, a document search is performed using the string as the search term. At least one must be provided.

constraint: `len(value) > 0`

default: `['The quick brown fox jumps over the lazy dog', 'TEST_SEARCH_STRING']`

3.1.2 andebench

AndEBench is an industry standard Android benchmark provided by The Embedded Microprocessor Benchmark Consortium (EEMBC).

http://www.eembc.org/andebench/about.php

From the website:

- Initial focus on CPU and Dalvik interpreter performance
- Internal algorithms concentrate on integer operations
- Compares the difference between native and Java performance
- Implements flexible multicore performance analysis
- Results displayed in Iterations per second
- Detailed log file for comprehensive engineering analysis

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

number_of_threads [integer] Number of threads that will be spawned by AndEBench.

single_threaded [boolean] If `true`, AndEBench will run with a single thread. Note: this must not be specified if `number_of_threads` has been specified.

native_only [boolean] If `true`, AndEBench will execute only the native portion of the benchmark.

3.1.3 androbench

Measures the storage performance of an Android device.

Website: http://www.androbench.org/wiki/AndroBench

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.4 angrybirds

Angry Birds game.

A very popular Android 2D game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.5 angrybirds_rio

Angry Birds Rio game.

The sequel to the very popular Android 2D game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.6 anomaly2

Anomaly 2 game demo and benchmark.

Plays three scenes from the game, benchmarking each one. Scores reported are intended to represent overall perceived quality of the game, based not only on raw FPS but also factors like smoothness.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.7 antutu

AnTuTu Benchmark is an benchmarking tool for Android Mobile Phone/Pad. It can run a full test of a key project, through the "Memory Performance","CPU Integer Performance","CPU Floating point Performance","2D 3D Graphics Performance","SD card reading/writing speed","Database IO" performance testing, and gives accurate analysis for Andriod smart phones.

http://www.antutulabs.com/AnTuTu-Benchmark

From the website:

AnTuTu Benchmark can support the latest quad-core cpu. In reaching the overall and individual scores of the hardware, AnTuTu Benchmark could judge your phone by the scores of the performance of the hardware. By uploading the scores, Benchmark can view your device in the world rankings, allowing points to let you know the level of hardware performance equipment.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

version [str] Specify the version of AnTuTu to be run. If not specified, the latest available version will be used.

allowed values: `'3.3.2'`, `'4.0.3'`, `'5.3.0'`, `'6.0.1'`

default: `'6.0.1'`

times [integer] The number of times the benchmark will be executed in a row (i.e. without going through the full setup/teardown process). Note: this does not work with versions prior to 4.0.3.

default: `1`

enable_sd_tests [boolean] If `True` enables SD card tests in pre version 4 AnTuTu. These tests were know to cause problems on platforms without an SD card. This parameter will be ignored on AnTuTu version 4 and higher.

3.1.8 apklaunch

Installs and runs a .apk file, waits wait_time_seconds, and tests if the app has started successfully.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

apk_file [str (mandatory)] Name to the .apk to run

uninstall_required [boolean] Set to true if the package should be uninstalled

wait_time_seconds [integer] Seconds to wait before testing if the app is still alive

3.1.9 applaunch

This workload launches and measures the launch time of applications for supporting workloads.

Currently supported workloads are the ones that implement `ApplaunchInterface`. For any workload to support this workload, it should implement the `ApplaunchInterface`. The corresponding java file of the workload associated with the application being measured is executed during the run. The application that needs to be measured is passed as a parametre `workload_name`. The parameters required for that workload have to be passed as a dictionary which is captured by the parametre `workload_params`. This information can be obtained by inspecting the workload details of the specific workload.

The workload allows to run multiple iterations of an application launch in two modes:

1. Launch from background
2. Launch from long-idle

These modes are captured as a parameter applaunch_type.

`launch_from_background` Launches an application after the application is sent to background by pressing Home button.

`launch_from_long-idle` Launches an application after killing an application process and clearing all the caches.

Test Description:

- During the initialization and setup, the application being launched is launched for the first time. The jar file of the workload of the application is moved to device at the location `workdir` which further implements the methods needed to measure the application launch time.
- **Run phase calls the UiAutomator of the applaunch which runs in two subphases.**
    1. **Applaunch Setup Run:** During this phase, welcome screens and dialogues during the first launch of the instrumented application are cleared.

2. **Applaunch Metric Run:** During this phase, the application is launched multiple times determined by the iteration number specified by the parametre `applaunch_iterations`. Each of these iterations are instrumented to capture the launch time taken and the values are recorded as UX-PERF marker values in logfile.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

workload_name [str] Name of the uxperf workload to launch

default: `'gmail'`

workload_params [dict] parameters of the uxperf workload whose application launch time is measured

applaunch_type [str] Choose launch_from_long-idle for measuring launch time from long-idle. These two types are described in the class description.

allowed values: `'launch_from_background'`, `'launch_from_long-idle'`

default: `'launch_from_background'`

applaunch_iterations [integer] Number of iterations of the application launch

default: `1`

report_results [boolean] Choose to report results of the application launch time.

default: `True`

3.1.10 appshare

Workload to test how responsive a device is when context switching between application tasks. It combines workflows from googlephotos, gmail and skype.

** Setup ** Credentials for the user account used to log into the Skype app have to be provided in the agenda, as well as the display name of the contact to call.

For reliable testing, this workload requires a good and stable internet connection, preferably on Wi-Fi.

Although this workload attempts to be network independent it requires a network connection (ideally, wifi) to run. This is because the welcome screen UI is dependent on an existing connection.

Test description:

1. **GooglePhotos is started in offline access mode** 1.1. The welcome screen is dismissed 1.2. Any promotion popup is dismissed 1.3. The provided `test_image` is selected and displayed
2. **The image is then shared across apps to Gmail** 2.1. The first run dialogue is dismissed 2.2. Enter recipient details in the To field 2.3. Enter text in the Subject field 2.4. Enter text in the Body field 2.5. Click the Send mail button
3. Return to Googlephotos and login to Skype via share action
4. **Return to Googlephotos and share the `test_image` with Skype** 4.1. Search for the `skype_contact_name` from the Contacts list 4.2. Dismiss any update popup that appears 4.3. The image is posted in the Chat

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

test_image [str] An image to be copied onto the device that will be shared across multiple apps

default: `'uxperf_1600x1200.jpg'`

email_recipient [str] The email address of the recipient to recieve the shared image

default: `'wa-devnull@mailinator.com'`

skype_login_name [str (mandatory)] Account to use when logging into skype from which to share the image

skype_login_pass [str (mandatory)] Password associated with the skype account

skype_contact_name [str] This is the contact display name as it appears in the people list

default: `'Echo / Sound Test Service'`

3.1.11 audio

Audio workload plays an MP3 file using the built-in music player. By default, it plays Canon_in_D_Pieano.mp3 for 30 seconds.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

duration [integer] The duration the music will play for in seconds.

default: `30`

audio_file [str] The (on-host) path to the audio file to be played.

Note: If the default file is not present locally, it will be downloaded.

---

default: `'~/.workload_automation/dependencies/Canon_in_D_Piano.mp3'`

perform_cleanup [boolean] If `True`, workload files on the device will be deleted after execution.

clear_file_cache [boolean] Clear the the file cache on the target device prior to running the workload.

default: `True`

3.1.12 autotest

Executes tests from ChromeOS autotest suite

Note: This workload *must* be run inside a CromeOS SDK chroot.

See: https://www.chromium.org/chromium-os/testing/power-testing

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

test [str (mandatory)] The test to be run

test_that_args [arguments] Extra arguments to be passed to test_that_invocation.

run_timeout [integer] Timeout, in seconds, for the test execution.

default: `1800`

3.1.13 bbench

BBench workload opens the built-in browser and navigates to, and scrolls through, some preloaded web pages and ends the workload by trying to connect to a local server it runs after it starts. It can also play the workload while it plays an audio file in the background.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

with_audio [boolean] Specifies whether an MP3 should be played in the background during workload execution.

server_timeout [integer] Specifies the timeout (in seconds) before the server is stopped.

default: `300`

force_dependency_push [boolean] Specifies whether to push dependency files to the device to the device if they are already on it.

audio_file [str] The (on-host) path to the audio file to be played. This is only used if `with_audio` is `True`.

default: `'~/.workload_automation/dependencies/Canon_in_D_Piano.mp3'`

perform_cleanup [boolean] If `True`, workload files on the device will be deleted after execution.

clear_file_cache [boolean] Clear the the file cache on the target device prior to running the workload.

default: `True`

browser_package [str] Specifies the package name of the device's browser app.

default: `'com.android.browser'`

browser_activity [str] Specifies the startup activity name of the device's browser app.

default: `'.BrowserActivity'`

3.1.14 benchmarkpi

Measures the time the target device takes to run and complete the Pi calculation algorithm.

http://androidbenchmark.com/howitworks.php

from the website:

The whole idea behind this application is to use the same Pi calculation algorithm on every Android Device and check how fast that proccess is. Better calculation times, conclude to faster Android devices. This way you can also check how lightweight your custom made Android build is. Or not.

As Pi is an irrational number, Benchmark Pi does not calculate the actual Pi number, but an approximation near the first digits of Pi over the same calculation circles the algorithms needs.

So, the number you are getting in miliseconds is the time your mobile device takes to run and complete the Pi calculation algorithm resulting in a approximation of the first Pi digits.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.15 blogbench

Blogbench is a portable filesystem benchmark that tries to reproduce the load of a real-world busy file server.

Blogbench stresses the filesystem with multiple threads performing random reads, writes and rewrites in order to get a realistic idea of the scalability and the concurrency a system can handle.

Source code are available from: https://download.pureftpd.org/pub/blogbench/

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

iterations [integer] The number of iterations to run

default: `30`

3.1.16 caffeinemark

CaffeineMark is a series of tests that measure the speed of Java programs running in various hardware and software configurations.

http://www.benchmarkhq.ru/cm30/info.html

From the website:

CaffeineMark scores roughly correlate with the number of Java instructions executed per second, and do not depend significantly on the the amount of memory in the system or on the speed of a computers disk drives or internet connection.

The following is a brief description of what each test does:

- Sieve: The classic sieve of eratosthenes finds prime numbers.
- **Loop: The loop test uses sorting and sequence generation as to measure** compiler optimization of loops.
- **Logic: Tests the speed with which the virtual machine executes** decision-making instructions.
- **Method: The Method test executes recursive function calls to see how** well the VM handles method calls.
- Float: Simulates a 3D rotation of objects around a point.
- Graphics: Draws random rectangles and lines.
- Image: Draws a sequence of three graphics repeatedly.
- Dialog: Writes a set of values into labels and editboxes on a form.

The overall CaffeineMark score is the geometric mean of the individual scores, i.e., it is the 9th root of the product of all the scores.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.17 cameracapture

Uses in-built Android camera app to take photos.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

no_of_captures [integer] Number of photos to be taken.

default: `5`

time_between_captures [integer] Time, in seconds, between two consecutive camera clicks.

default: `5`

3.1.18 camerarecord

Uses in-built Android camera app to record the video for given interval of time.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

recording_time [integer] The video recording time in seconds.

default: `60`

recording_mode [str] The video recording mode.

allowed values: `'normal'`, `'slow_motion'`

default: `'normal'`

3.1.19 castlebuilder

Castle Builder game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.20 castlemaster

Castle Master v1.09 game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.21 cfbench

CF-Bench is (mainly) CPU and memory benchmark tool specifically designed to be able to handle multi-core devices, produce a fairly stable score, and test both native as well managed code performance.

https://play.google.com/store/apps/details?id=eu.chainfire.cfbench&hl=en

From the website:

It tests specific device properties you do not regularly see tested by other benchmarks, and runs in a set timeframe.

It does produce some "final" scores, but as with every benchmark, you should take those with a grain of salt. It is simply not theoretically possible to produce a single number that accurately describes a device's performance.

Note: This workload relies on the device being rooted

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.22 citadel

Epic Citadel demo showcasing Unreal Engine 3.

The game has very rich graphics details. The workload only moves around its environment for the specified time.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

duration [integer] Duration, in seconds, of the run (may need to be adjusted for different devices.

default: `60`

3.1.23 cyclictest

Measures the amount of time that passes between when a timer expires and when the thread which set the timer actually runs.

Cyclic test works by taking a time snapshot just prior to waiting for a specific time interval (t1), then taking another time snapshot after the timer finishes (t2), then comparing the theoretical wakeup time with the actual wakeup time (t2 -(t1 + sleep_time)). This value is the latency for that timers wakeup.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

clock [str] specify the clock to be used during the test.

allowed values: `'monotonic'`, `'realtime'`

default: `'realtime'`

duration [integer] Specify the length for the test to run in seconds.

default: `30`

quiet [boolean] Run the tests quiet and print only a summary on exit.

default: `True`

thread [integer] Set the number of test threads

default: `8`

latency [integer] Write the value to /dev/cpu_dma_latency

default: `1000000`

extra_parameters [str] Any additional command line parameters to append to the existing parameters above. A list can be found at https://rt.wiki.kernel.org/index.php/Cyclictest or in the help page `cyclictest -h`

clear_file_cache [boolean] Clear file caches before starting test

default: `True`

screen_off [boolean] If true it will turn the screen off so that onscreen graphics do not effect the score. This is predominantly for devices without a GPU

default: `True`

3.1.24 dex2oat

Benchmarks the execution time of dex2oat (a key part of APK installation process).

ART is a new Android runtime in KitKat, which replaces Dalvik VM. ART uses Ahead-Of-Time compilation. It pre-compiles ODEX files used by Dalvik using dex2oat tool as part of APK installation process.

This workload benchmarks the time it take to compile an APK using dex2oat, which has a significant impact on the total APK installation time, and therefore user experience.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

instruction_set [str] Specifies the instruction set to compile for. Only options supported by the target device can be used.

allowed values: `'arm'`, `'arm64'`, `'x86'`, `'x86_64'`, `'mips'`

default: `'arm64'`

3.1.25 dhrystone

Runs the Dhrystone benchmark.

Original source from:

```
http://classes.soe.ucsc.edu/cmpe202/benchmarks/standard/dhrystone.c
```

This version has been modified to configure duration and the number of threads used.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

duration [integer] The duration, in seconds, for which dhrystone will be executed. Either this or `mloops` should be specified but not both.

mloops [integer] Millions of loops to run. Either this or `duration` should be specified, but not both. If neither is specified, this will default to `100`

threads [integer] The number of separate dhrystone "threads" that will be forked.

default: `4`

delay [integer] The delay, in seconds, between kicking off of dhrystone threads (if `threads` > 1).

taskset_mask [integer] The processes spawned by the workload will be pinned to cores as specified by this parameter

3.1.26 dungeondefenders

Dungeon Defenders game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.27 ebizzy

ebizzy is designed to generate a workload resembling common web application server workloads. It is highly threaded, has a large in-memory working set with low locality, and allocates and deallocates memory frequently. When running most efficiently, it will max out the CPU.

ebizzy description taken from the source code at https://github.com/linux-test-project/ltp/tree/master/utils/benchmark/ebizzy-0.3

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

threads [integer] Number of threads to execute.

default: `2`

seconds [integer] Number of seconds.

default: `10`

chunks [integer] Number of memory chunks to allocate.

default: `10`

extra_params [str] Extra parameters to pass in (e.g. -M to disable mmap). See ebizzy -? for full list of options.

3.1.28 facebook

Uses com.facebook.patana apk for facebook workload. This workload does the following activities in facebook

> Login to facebook account. Send a message. Check latest notification. Search particular user account and visit his/her facebook account. Find friends. Update the facebook status

---

Note: This workload starts disableUpdate workload as a part of setup to disable online updates, which helps to tackle problem of uncertain behavier during facebook workload run.]

---

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

> default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

> default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

> default: `True`

3.1.29 geekbench

Geekbench provides a comprehensive set of benchmarks engineered to quickly and accurately measure processor and memory performance.

http://www.primatelabs.com/geekbench/

From the website:

Designed to make benchmarks easy to run and easy to understand, Geekbench takes the guesswork out of producing robust and reliable benchmark results.

Geekbench scores are calibrated against a baseline score of 1,000 (which is the score of a single-processor Power Mac G5 @ 1.6GHz). Higher scores are better, with double the score indicating double the performance.

The benchmarks fall into one of four categories:

- integer performance.
- floating point performance.
- memory performance.

- stream performance.

Geekbench benchmarks: http://www.primatelabs.com/geekbench/doc/benchmarks.html

Geekbench scoring methedology: http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-scores

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

version [str] Specifies which version of the workload should be run.

allowed values: `'2'`, `'3.0.0'`, `'3.4.1'`, `'4.0.1'`

default: `'4.0.1'`

times [integer] Specfies the number of times the benchmark will be run in a "tight loop", i.e. without performaing setup/teardown inbetween.

default: `1`

timeout [integer] Timeout for a single iteration of the benchmark. This value is multiplied by `times` to calculate the overall run timeout.

default: `900`

disable_update_result [boolean] If `True` the results file will not be pulled from the devices /data/data/com.primatelabs.geekbench folder. This allows the workload to be run on unrooted devices and the results extracted manually later.

3.1.30 geekbench-corporate

Geekbench provides a comprehensive set of benchmarks engineered to quickly and accurately measure processor and memory performance.

http://www.primatelabs.com/geekbench/

From the website:

Designed to make benchmarks easy to run and easy to understand, Geekbench takes the guesswork out of producing robust and reliable benchmark results.

Geekbench scores are calibrated against a baseline score of 1,000 (which is the score of a single-processor Power Mac G5 @ 1.6GHz). Higher scores are better, with double the score indicating double the performance.

The benchmarks fall into one of four categories:

- integer performance.
- floating point performance.
- memory performance.
- stream performance.

Geekbench benchmarks: http://www.primatelabs.com/geekbench/doc/benchmarks.html

Geekbench scoring methedology: http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-scores

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

version [str] Specifies which version of the workload should be run.

allowed values: `'4.1.0'`

default: `'4.1.0'`

times [integer] Specfies the number of times the benchmark will be run in a "tight loop", i.e. without performaing setup/teardown inbetween.

default: `1`

timeout [integer] Timeout for a single iteration of the benchmark. This value is multiplied by `times` to calculate the overall run timeout.

default: `900`

disable_update_result [boolean] If `True` the results file will not be pulled from the devices /data/data/com.primatelabs.geekbench folder. This allows the workload to be run on unrooted devices and the results extracted manually later.

3.1.31 glb_corporate

GFXBench GL (a.k.a. GLBench) v3.0 Corporate version.

This is a version of GLBench available through a corporate license (distinct from the version available in Google Play store).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

times [integer] Specifies the number of times the benchmark will be run in a "tight loop", i.e. without performaing setup/teardown inbetween.

constraint: `value > 0`

default: `1`

resolution [str] Explicitly specifies the resultion under which the benchmark will be run. If not specfied, device's native resoution will used.

allowed values: `'720p'`, `'1080p'`, `'720'`, `'1080'`

test_id [str] ID of the GFXBench test to be run.

allowed values: `'gl_alu'`, `'gl_alu_off'`, `'gl_blending'`, `'gl_blending_off'`, `'gl_driver'`, `'gl_driver_off'`, `'gl_fill'`, `'gl_fill_off'`, `'gl_manhattan'`, `'gl_manhattan_off'`, `'gl_trex'`, `'gl_trex_battery'`, `'gl_trex_off'`, `'gl_trex_qmatch'`, `'gl_trex_qmatch_highp'`

default: `'gl_manhattan_off'`

run_timeout [integer] Time out for workload execution. The workload will be killed if it hasn't completed withint this period.

default: `600`

3.1.32 glbenchmark

Measures the graphics performance of Android devices by testing the underlying OpenGL (ES) implementation.

http://gfxbench.com/about-gfxbench.jsp

From the website:

> The benchmark includes console-quality high-level 3D animations (T-Rex HD and Egypt HD) and low-level graphics measurements.
>
> With high vertex count and complex effects such as motion blur, parallax mapping and particle systems, the engine of GFXBench stresses GPUs in order provide users a realistic feedback on their device.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

version [str] Specifies which version of the benchmark to run (different versions support different use cases).

allowed values: `'2.7'`, `'2.5'`

default: `'2.7'`

use_case [str] Specifies which usecase to run, as listed in the benchmark menu; e.g. `'GLBenchmark 2.5 Egypt HD'`. For convenience, two aliases are provided for the most common use cases: `'egypt'` and `'t-rex'`. These could be use instead of the full use case title. For version `'2.7'` it defaults to `'t-rex'`, for version `'2.5'` it defaults to `'egypt-classic'`.

variant [str] Specifies which variant of the use case to run, as listed in the benchmarks menu (small text underneath the use case name); e.g. `'C24Z16 Onscreen Auto'`. For convenience, two aliases are provided for the most common variants: `'onscreen'` and `'offscreen'`. These may be used instead of full variant names.

default: `'onscreen'`

times [integer] Specfies the number of times the benchmark will be run in a "tight loop", i.e. without performaing setup/teardown inbetween.

default: `1`

timeout [integer] Specifies how long, in seconds, UI automation will wait for results screen to appear before assuming something went wrong.

default: `200`

3.1.33 gmail

A workload to perform standard productivity tasks within Gmail. The workload carries out various tasks, such as creating new emails, attaching images and sending them.

Test description: 1. Open Gmail application 2. Click to create New mail 3. Attach an image from the local images folder to the email 4. Enter recipient details in the To field 5. Enter text in the Subject field 6. Enter text in the Compose field 7. Click the Send mail button

Known working APK version: 7.6.18.160170480

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

recipient [str] The email address of the recipient. Setting a void address will stop any mesage failures clogging up your device inbox

default: `'wa-devnull@mailinator.com'`

test_image [str] An image to be copied onto the device that will be attached to the email

default: `'uxperf_1600x1200.jpg'`

3.1.34 googlemap

Navigation app.

Stock map provided by Google Inc. Based on revent, we can use this workload to do multiple tasks such as navigation usecases, swipe & pinch etc.

Provided revent is for Odriod XU3 for navigation use case. For running on other devices, we need to build revent.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.35 googlephotos

A workload to perform standard productivity tasks with Google Photos. The workload carries out various tasks, such as browsing images, performing zooms, and post-processing the image.

Test description:

1. Four images are copied to the device
2. The application is started in offline access mode
3. Gestures are performed to pinch zoom in and out of the selected image
4. The colour of a selected image is edited by selecting the colour menu, incrementing the colour, resetting the colour and decrementing the colour using the seek bar.
5. A crop test is performed on a selected image. UiAutomator does not allow the selection of the crop markers so the image is tilted positively, reset and then tilted negatively to get a similar cropping effect.

6. A rotate test is performed on a selected image, rotating anticlockwise 90 degrees, 180 degrees and 270 degrees.

Known working APK version: 1.21.0.123444480

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

test_images [list_of_strs] A list of four JPEG and/or PNG files to be pushed to the device. Absolute file paths may be used but tilde expansion must be escaped.

constraint: `len(unique(value)) == 4`

default: `['uxperf_1200x1600.png', 'uxperf_1600x1200.jpg', 'uxperf_2448x3264.png', 'uxperf_3264x2448.jpg']`

3.1.36 googleplaybooks

A workload to perform standard productivity tasks with googleplaybooks. This workload performs various tasks, such as searching for a book title online, browsing through a book, adding and removing notes, word searching, and querying information about the book.

Test description: 1. Open Google Play Books application 2. Dismisses sync operation (if applicable) 3. Searches for a book title 4. Adds books to library if not already present 5. Opens 'My Library' contents 6. Opens selected book 7. Gestures are performed to swipe between pages and pinch zoom in and out of a page 8. Selects a specified chapter based on page number from the navigation view 9. Selects a word in the centre of screen and adds a test note to the page 10. Removes the test note from the page (clean up) 11. Searches for the number of occurrences of a common word throughout the book 12. Switches page styles from 'Day' to 'Night' to 'Sepia' and back to 'Day' 13. Uses the 'About this book' facility on the currently selected book

NOTE: This workload requires a network connection (ideally, wifi) to run, a Google account to be setup on the device, and payment details for the account. Free books require payment details to have been setup otherwise it fails. Tip: Install the 'Google Opinion Rewards' app to bypass the need to enter valid card/bank detail.

Known working APK version: 3.13.17

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

search_book_title [str] The book title to search for within Google Play Books archive. The book must either be already in the account's library, or free to purchase.

default: `'Nikola Tesla: Imagination and the Man That Invented the 20th Century'`

library_book_title [str] The book title to search for within My Library. The Library name can differ (usually shorter) to the Store name. If left blank, the `search_book_title` will be used.

default: `'Nikola Tesla'`

select_chapter_page_number [integer] The Page Number to search for within a selected book's Chapter list. Note: Accepts integers only.

default: `4`

search_word [str] The word to search for within a selected book. Note: Accepts single words only.

default: `'the'`

account [str] If you are running this workload on a device which has more than one Google account setup, then this parameter is used to select which account to select when prompted. The account requires the book to have already been purchased or payment details already associated with the account. If omitted, the first account in the list will be selected if prompted.

3.1.37 googleslides

A workload to perform standard productivity tasks with Google Slides. The workload carries out various tasks, such as creating a new presentation, adding text, images, and shapes, as well as basic editing and playing a slideshow. This workload should be able to run without a network connection.

There are two main scenarios:

1. create test: a presentation is created in-app and some editing done on it,
2. load test: a pre-existing PowerPoint file is copied onto the device for testing.

— create — Create a new file in the application and perform basic editing on it. This test also requires an image file specified by the param `test_image` to be copied onto the device.

Test description:

1. Start the app and skip the welcome screen. Dismiss the work offline banner if present.
2. Go to the app settings page and enables PowerPoint compatibility mode. This allows PowerPoint files to be created inside Google Slides.
3. Create a new PowerPoint presentation in the app (PPT compatibility mode) with a title slide and save it to device storage.
4. Insert another slide and to it insert the pushed image by picking it from the gallery.
5. Insert a final slide and add a shape to it. Resize and drag the shape to modify it.
6. Finally, navigate back to the documents list.

— load — Copy a PowerPoint presentation onto the device to test slide navigation. The PowerPoint file to be copied is given by `test_file`.

Test description:

1. From the documents list (following the create test), open the specified PowerPoint by navigating into device storage and wait for it to be loaded.
2. A navigation test is performed while the file is in editing mode (i.e. not slideshow). swiping forward to the next slide until `slide_count` swipes are performed.
3. While still in editing mode, the same action is done in the reverse direction back to the first slide.
4. Enter presentation mode by selecting to play the slideshow.
5. Swipe forward to play the slideshow, for a maximum number of `slide_count` swipes.
6. Finally, repeat the previous step in the reverse direction while still in presentation mode, navigating back to the first slide.

NOTE: There are known issues with the reliability of this workload on some targets. It MAY NOT ALWAYS WORK on your device. If you do run into problems, it might help to set `do_text_entry` parameter to `False`.

Known working APK version: 1.7.032.06

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

test_image [str] An image to be copied onto the device that will be embedded in the PowerPoint file as part of the test.

default: `'uxperf_1600x1200.jpg'`

test_file [str] If specified, the workload will copy the PowerPoint file to be used for testing onto the device. Otherwise, a file will be created inside the app.

default: `'uxperf_test_doc.pptx'`

slide_count [integer] Number of slides in aforementioned local file. Determines number of swipe actions when playing slide show.

default: `5`

do_text_entry [boolean] If set to `True`, will attempt to enter text in the first slide as part of the test. Currently seems to be problematic on some devices, most notably Samsung devices.

default: `True`

3.1.38 gunbros2

Gun Bros. 2 game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.39 hackbench

Hackbench runs a series of tests for the Linux scheduler.

For details, go to: https://github.com/linux-test-project/ltp/

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

datasize [integer] Message size in bytes.

default: `100`

groups [integer] Number of groups.

default: `10`

loops [integer] Number of loops.

default: `100`

fds [integer] Number of file descriptors.

default: `40`

extra_params [str] Extra parameters to pass in. See the hackbench man page or type *hackbench –help* for list of options.

duration [integer] Test duration in seconds.

default: `30`

3.1.40 homescreen

A workload that goes to the home screen and idles for the the specified duration.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

duration [integer] Specifies the duration, in seconds, of this workload.

default: `20`

3.1.41 hwuitest

Tests UI rendering latency on android devices

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

test [caseless_string] The test to run:

- `'shadowgrid'`: creates a grid of rounded rects that cast shadows, high CPU & GPU load
- `'rectgrid'`: creates a grid of 1x1 rects
- `'oval'`: draws 1 oval

allowed values: `'shadowgrid'`, `'rectgrid'`, `'oval'`

default: `'shadowgrid'`

loops [integer] The number of test iterations.

default: `3`

frames [integer] The number of frames to run the test over.

default: `150`

3.1.42 idle

Do nothing for the specified duration.

On android devices, this may optionally stop the Android run time, if `stop_android` is set to `True`.

Note: This workload requires the device to be rooted.

---

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

duration [integer] Specifies the duration, in seconds, of this workload.

default: `20`

stop_android [boolean] Specifies whether the Android run time should be stopped. (Can be set only for Android devices).

3.1.43 iozone

Iozone is a filesystem benchmark that runs a series of disk I/O performance tests.

Here is a list of tests that you can run in the iozone workload. The descriptions are from the official iozone document.

0 - Write Test Measure performance of writing a new file. Other tests rely on the file written by this, so it must always be enabled (WA will automatically neable this if not specified).

1 - Rewrite Test Measure performance of writing an existing file.

2 - Read Test Measure performance of reading an existing file.

3 - Reread Test Measure performance of rereading an existing file.

4 - Random Read Test Measure performance of reading a file by accessing random locations within the file.

5 - Random Write Test Measure performance of writing a file by accessing random locations within the file.

6 - Backwards Read Test Measure performance of reading a file backwards.

7 - Record Rewrite Test Measure performance of writing and rewriting a particular spot within the file.

8 - Strided Read Test Measure performance of reading a file with strided access behavior.

9 - Fwrite Test Measure performance of writing a file using the library function fwrite() that performances buffered write operations.

10 - Frewrite Test Measure performance of writing a file using the the library function fwrite() that performs buffered and blocked write operations.

11 - Fread Test Measure performance of reading a file using the library function fread() that performs buffered and blocked read operations.

12 - Freread Test Same as the Fread Test except the current file being read was read previously sometime in the past.

By default, iozone will run all tests in auto mode. To run specific tests, they must be written in the form of:

[0,1,4,5]

Please enable classifiers in your agenda or config file in order to display the results properly in the results.csv file.

The official website for iozone is at www.iozone.org.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

tests [list_of_ints] List of performance tests to run.

allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `11`, `12`

auto_mode [boolean] Run tests in auto mode.

default: `True`

timeout [integer] Timeout for the workload.

default: `14400`

file_size [integer] Fixed file size.

record_length [integer] Fixed record length.

threads [integer] Number of threads

other_params [str] Other parameter. Run iozone -h to see list of options.

3.1.44 ironman3

Iron Man 3 game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.45 krazykart

Krazy Kart Racing game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.46 linpack

The LINPACK Benchmarks are a measure of a system's floating point computing power.

http://en.wikipedia.org/wiki/LINPACK_benchmarks

From the article:

Introduced by Jack Dongarra, they measure how fast a computer solves a dense n by n system of linear equations Ax = b, which is a common task in engineering.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

output_file [str] On-device output file path.

3.1.47 linpack-cli

linpack benchmark with a command line interface

Benchmarks FLOPS (floating point operations per second).

This is the oldschool version of the bencmark. Source may be viewed here:

http://www.netlib.org/benchmark/linpackc.new

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

array_size [integer] size of arrays to be used by the benchmark.

default: `200`

3.1.48 lmbench

Run a subtest from lmbench, a suite of portable ANSI/C microbenchmarks for UNIX/POSIX.

In general, lmbench measures two key features: latency and bandwidth. This workload supports a subset of lmbench tests. lat_mem_rd can be used to measure latencies to memory (including caches). bw_mem can be used to measure bandwidth to/from memory over a range of operations.

Further details, and source code are available from:

http://sourceforge.net/projects/lmbench/.

See lmbench/bin/README for license details.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

test [str] Specifies an lmbench test to run.

allowed values: `'lat_mem_rd'`, `'bw_mem'`

default: `'lat_mem_rd'`

stride [list_or_type] Stride for lat_mem_rd test. Workload will iterate over one or more integer values.

default: `[128]`

thrash [boolean] Sets -t flag for lat_mem_rd_test

default: `True`

size [list_or_string] Data set size for lat_mem_rd bw_mem tests.

default: `'4m'`

mem_category [list_or_string] List of memory catetories for bw_mem test.

default: `('rd', 'wr', 'cp', 'frd', 'fwr', 'fcp', 'bzero', 'bcopy')`

parallelism [integer] Parallelism flag for tests that accept it.

warmup [integer] Warmup flag for tests that accept it.

repetitions [integer] Repetitions flag for tests that accept it.

force_abi [str] Override device abi with this value. Can be used to force arm32 on 64-bit devices.

run_timeout [integer] Timeout for execution of the test.

default: `900`

times [integer] Specifies the number of times the benchmark will be run in a "tight loop", i.e. without performaing setup/teardown inbetween. This parameter is distinct from "repetitions", as the latter takes place within the benchmark and produces a single result.

constraint: `value > 0`

default: `1`

taskset_mask [integer] Specifies the CPU mask the benchmark process will be pinned to.

3.1.49 manual

Yields control to the user, either for a fixed period or based on user input, to perform custom operations on the device, about which workload automation does not know of.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

duration [integer] Control of the devices is yielded for the duration (in seconds) specified. If not specified, `user_triggered` is assumed.

user_triggered [boolean] If `True`, WA will wait for user input after starting the workload; otherwise fixed duration is expected. Defaults to `True` if `duration` is not specified, and `False` otherwise.

view [str] Specifies the View of the workload. This enables instruments that require a View to be specified, such as the `fps` instrument. This is required for using "SurfaceFlinger" to collect FPS statistics and is primarily used on devices pre API level 23

default: `'SurfaceView'`

package [str] Specifies the package name of the workload. This enables instruments that require a Package to be specified, such as the `fps` instrument. This allows for "gfxinfo" to be used and is the preferred method of collection for FPS statistics on devices API level 23+

enable_logcat [boolean] If `True`, `manual` workload will collect logcat as part of the results.

3.1.50 memcpy

Runs memcpy in a loop.

This will run memcpy in a loop for a specified number of times on a buffer of a specified size. Additionally, the affinity of the test can be set to one or more specific cores.

This workload is single-threaded. It genrates no scores or metrics by itself.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

buffer_size [integer] Specifies the size, in bytes, of the buffer to be copied.

default: `5242880`

iterations [integer] Specfies the number of iterations that will be performed.

default: `1000`

cpus [list] A list of integers specifying ordinals of cores to which the affinity of the test process should be set. If not specified, all avaiable cores will be used.

3.1.51 nenamark

NenaMark is an OpenGL-ES 2.0 graphics performance benchmark for Android devices.

http://nena.se/nenamark_story

From the website:

The NenaMark2 benchmark scene averages about 45k triangles, with a span between 26k and 68k triangles. It averages 96 batches per frame and contains about 15 Mb of texture data (non-packed).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

duration [integer] Number of seconds to wait before considering the benchmark finished

default: `120`

3.1.52 octaned8

Runs the Octane d8 benchmark.

This workload runs d8 binaries built from source and placed in the dependencies folder along with test assets from https://github.com/chromium/octane which also need to be placed in an assets folder within the dependencies folder.

Original source from:

```
https://github.com/v8/v8/wiki/D8%20on%20Android
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

run_timeout [integer] Timeout, in seconds, for the test execution.

default: `180`

3.1.53 peacekeeper

Peacekeeper is a free and fast browser test that measures a browser's speed.

Note: This workload requires a network connection as well as support for one of the two currently-supported browsers. Moreover, TC2 has compatibility issue with chrome

---

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

browser [str] The browser to be benchmarked.

allowed values: `'firefox'`, `'chrome'`

default: `'firefox'`

output_file [str] The result URL of peacekeeper benchmark will be written into this file on device after completion of peacekeeper benchmark. Defaults to peacekeeper.txt in the device's `working_directory`.

peacekeeper_url [str] The URL to run the peacekeeper benchmark.

default: `'http://peacekeeper.futuremark.com/run.action'`

3.1.54 power_loadtest

power_LoadTest (part of ChromeOS autotest suite) continuously cycles through a set of browser-based activities and monitors battery drain on a device.

Note: This workload *must* be run inside a CromeOS SDK chroot.

See: https://www.chromium.org/chromium-os/testing/power-testing

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

board [str] The name of the board to be used for the test. If this is not specified, BOARD environment variable will be used.

variant [str] The variant of the test to run; If not specified, the full power_LoadTest will run (until the device battery is drained). The only other variant available in the vanilla test is "1hour", but further variants may be added by providing custom control files.

test_that_args [arguments] Extra arguments to be passed to test_that_invocation.

run_timeout [integer] Timeout, in seconds, for the test execution.

default: `86400`

3.1.55 quadrant

Quadrant is a benchmark for mobile devices, capable of measuring CPU, memory, I/O and 3D graphics performance.

http://www.aurorasoftworks.com/products/quadrant

From the website: Quadrant outputs a score for the following categories: 2D, 3D, Mem, I/O, CPU , Total.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.56 real-linpack

This version of *Linpack <http://en.wikipedia.org/wiki/LINPACK_benchmarks>* was developed by Dave Butcher. RealLinpack tries to find the number of threads that give you the maximum linpack score.

RealLinpack runs 20 runs of linpack for each number of threads and calculates the mean and confidence. It stops when the score's confidence interval drops below the current best score interval. That is, when (current_score + confidence) < (best_score - best_score_confidence)

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

max_threads [integer] The maximum number of threads that real linpack will try.

constraint: `value > 0`

default: `16`

3.1.57 realracing3

Real Racing 3 game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.58 recentfling

Tests UI jank on android devices.

For this workload to work, `recentfling.sh` and `defs.sh` must be placed in `~/.workload_automation/dependencies/recentfling/`. These can be found in the AOSP Git repository.

To change the apps that are opened at the start of the workload you will need to modify the `defs.sh` file. You will need to add your app to `dfltAppList` and then add a variable called `{app_name}Activity` with the name of the activity to launch (where `{add_name}` is the name you put into `dfltAppList`).

You can get a list of activities available on your device by running `adb shell pm list packages -f`

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

loops [integer] The number of test iterations.

default: `3`

start_apps [boolean] If set to `False`,no apps will be started before flinging through the recent apps list (in which the assumption is there are already recently started apps in the list.

default: `True`

device_name [str] If set, recentfling will use the fling parameters for this device instead of automatically guessing the device. This can also be used if the device is not supported by recentfling, but its screensize is similar to that of one that is supported.

For possible values, check your recentfling.sh. At the time of writing, valid values are: 'shamu', 'hammerhead', 'angler', 'ariel', 'mtp8996', 'bullhead' or 'volantis'.

3.1.59 rt-app

A test application that simulates configurable real-time periodic load.

rt-app is a test application that starts multiple periodic threads in order to simulate a real-time periodic load. It supports SCHED_OTHER, SCHED_FIFO, SCHED_RR as well as the AQuoSA framework and SCHED_DEADLINE.

The load is described using JSON-like config files. Below are a couple of simple examples.

Simple use case which creates a thread that run 1ms then sleep 9ms until the use case is stopped with Ctrl+C:

```
{
    "tasks" : {
        "thread0" : {
            "loop" : -1,
            "run" :   20000,
            "sleep" : 80000
        }
    },
    "global" : {
        "duration" : 2,
        "calibration" : "CPU0",
        "default_policy" : "SCHED_OTHER",
        "pi_enabled" : false,
        "lock_pages" : false,
        "logdir" : "./",
        "log_basename" : "rt-app1",
        "ftrace" : false,
        "gnuplot" : true,
    }
}
```

Simple use case with 2 threads that runs for 10 ms and wake up each other until the use case is stopped with Ctrl+C

```
{
    "tasks" : {
        "thread0" : {
            "loop" : -1,
            "run" :     10000,
            "resume" : "thread1",
            "suspend" : "thread0"
        },
        "thread1" : {
            "loop" : -1,
            "run" :     10000,
            "resume" : "thread0",
            "suspend" : "thread1"
```

```
        }
    }
}
```

Please refer to the existing configs in `$WA_ROOT/wlauto/workloads/rt_app/use_case` for more examples.

The version of rt-app currently used with this workload contains enhancements and modifications done by Linaro. The source code for this version may be obtained here:

http://git.linaro.org/power/rt-app.git

The upstream version of rt-app is hosted here:

https://github.com/scheduler-tools/rt-app

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

config [str] Use case configuration file to run with rt-app. This may be either the name of one of the "standard" configurations included with the workload. or a path to a custom JSON file provided by the user. Either way, the ".json" extension is implied and will be added automatically if not specified in the argument.

The following is the list of standard configurations currently included with the workload: browser-long.json, spreading-tasks.json, mp3-short.json, camera-short.json, browser-short.json, mp3-long.json, video-long.json, camera-long.json, taskset.json, video-short.json

default: `'taskset'`

duration [integer] Duration of the workload execution in Seconds. If specified, this will override the corresponding parameter in the JSON config.

taskset_mask [integer] Constrain execution to specific CPUs.

uninstall_on_exit [boolean] If set to `True`, rt-app binary will be uninstalled from the device at the end of the run.

force_install [boolean] If set to `True`, rt-app binary will always be deployed to the target device at the beginning of the run, regardless of whether it was already installed there.

3.1.60 shellscript

Runs an arbitrary shellscript on the device.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

script_file [str (mandatory)] The path (on the host) to the shell script file. This must be an absolute path (though it may contain ~).

argstring [str] A string that should contain arguments passed to the script.

timeout [integer] Timeout, in seconds, for the script run time.

default: `60`

3.1.61 skype

A workload to perform standard productivity tasks within Skype. The workload logs in to the Skype application, selects a recipient from the contacts list and then initiates either a voice or video call.

Test description:

1. Open Skype application
2. Log in to a pre-defined account
3. Select a recipient from the Contacts list
4. Initiate either a `voice` or `video` call for `duration` time (in seconds) Note: The actual duration of the call may not match exactly the intended duration due to the uiautomation overhead.

Skype Setup

- You must have a Skype account set up and its credentials passed as parameters into this workload
- The contact to be called must be added (and has accepted) to the account. It's possible to have multiple contacts in the list, however the contact to be called *must* be visible on initial navigation to the list.
- For video calls the contact must be able to received the call. This means that there must be a Skype client running (somewhere) with the contact logged in and that client must have been configured to auto-accept calls from the account on the device (how to set this varies between different versions of Skype and between platforms – please search online for specific instructions). https://support.skype.com/en/faq/FA3751/can-i-automatically-answer-all-my-calls-with-video-in-skype-for-windows-desktop

Known working APK version: 7.01.0.669

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

login_name [str (mandatory)] Account to use when logging into the device from which the call will be made

login_pass [str (mandatory)] Password associated with the account to log into the device

contact_name [str] This is the contact display name as it appears in the people list

default: `'Echo / Sound Test Service'`

duration [integer] This is the target duration of the call in seconds

default: `10`

action [str] Action to take - either voice call (default) or video

allowed values: `'voice'`, `'video'`

default: `'voice'`

3.1.62 smartbench

Smartbench is a multi-core friendly benchmark application that measures the overall performance of an android device. It reports both Productivity and Gaming Index.

https://play.google.com/store/apps/details?id=com.smartbench.twelve&hl=en

From the website:

It will be better prepared for the quad-core world. Unfortunately this also means it will run slower on older devices. It will also run slower on high-resolution tablet devices. All 3D tests are now rendered in full native resolutions so naturally it will stress hardware harder on these devices. This also applies to higher resolution hand-held devices.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.63 spec2000

SPEC2000 benchmarks measuring processor, memory and compiler.

http://www.spec.org/cpu2000/

From the web site:

SPEC CPU2000 is the next-generation industry-standardized CPU-intensive benchmark suite. SPEC designed CPU2000 to provide a comparative measure of compute intensive performance across the widest practical range of hardware. The implementation resulted in source code benchmarks developed from real user applications. These benchmarks measure the performance of the processor, memory and compiler on the tested system.

Note: At the moment, this workload relies on pre-built SPEC binaries (included in an asset bundle). These binaries *must* be built according to rules outlined here:

```
http://www.spec.org/cpu2000/docs/runrules.html#toc_2.0
```

in order for the results to be valid SPEC2000 results.

Note: This workload does not attempt to generate results in an admissible SPEC format. No metadata is provided (though some, but not all, of the required metdata is collected by WA elsewhere). It is upto the user to post-process results to generated SPEC-admissible results file, if that is their intention.

---

base vs peak

SPEC2000 defines two build/test configuration: base and peak. Base is supposed to use basic configuration (e.g. default compiler flags) with no tuning, and peak is specifically optimized for a system. Since this workload uses externally-built binaries, there is no way for WA to be sure what configuration is used – the user is expected to keep track of that. Be aware that base/peak also come with specific requirements for the way workloads are run (e.g. how many instances on multi-core systems):

```
http://www.spec.org/cpu2000/docs/runrules.html#toc_3
```

These are not enforced by WA, so it is again up to the user to ensure that correct workload parameters are specfied inthe agenda, if they intend to collect "official" SPEC results. (Those interested in collecting official SPEC results should also note that setting runtime parameters would violate SPEC runs rules that state that no configuration must be done to the platform after boot).

bundle structure

This workload expects the actual benchmark binaries to be provided in a tarball "bundle" that has a very specific structure. At the top level of the tarball, there should be two directories: "fp" and "int" – for each of the SPEC2000 categories. Under those, there is a sub-directory per benchmark. Each benchmark sub-directory contains three sub-sub-directorie:

- "cpus" contains a subdirectory for each supported cpu (e.g. a15) with a single executable binary for that cpu, in addition to a "generic" subdirectory that has not been optimized for a specific cpu and should run on any ARM system.
- "data" contains all additional files (input, configuration, etc) that the benchmark executable relies on.
- "scripts" contains one or more one-liner shell scripts that invoke the benchmark binary with appropriate command line parameters. The name of the script must be in the format <benchmark name>[.<variant name>].sh, i.e. name of benchmark, optionally followed by variant name, followed by ".sh" extension. If there is more than one script, then all of them must have a variant; if there is only one script the it should not contain a variant.

A typical bundle may look like this:

```
|- fp
|   |-- ammp
|   |    |-- cpus
|   |    |    |-- generic
|   |    |    |    |-- ammp
|   |    |    |-- a15
|   |    |    |    |-- ammp
|   |    |    |-- a7
|   |    |    |    |-- ammp
|   |    |-- data
|   |    |    |-- ammp.in
|   |    |-- scripts
|   |    |    |-- ammp.sh
|   |-- applu
.   .    .
.   .    .
.   .    .
|- int
.
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

benchmarks [list_or_string] Specifies the SPEC benchmarks to run.

mode [str] SPEC benchmarks can report either speed to execute or throughput/rate. In the latter case, several "threads" will be spawned.

allowed values: `'speed'`, `'rate'`

default: `'speed'`

number_of_threads [integer] Specify the number of "threads" to be used in 'rate' mode. (Note: on big.LITTLE systems this is the number of threads, for *each cluster*).

force_extract_assets [boolean] if set to `True`, will extract assets from the bundle, even if they are already extracted. Note: this option implies `force_push_assets`.

force_push_assets [boolean] If set to `True`, assets will be pushed to device even if they're already present.

timeout [integer] Timeout, in seconds, for the execution of single spec test.

default: `1200`

3.1.64 sqlitebm

Measures the performance of the sqlite database. It determines within what time the target device processes a number of SQL queries.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

3.1.65 stream

Measures memory bandwidth.

The original source code be found on: https://www.cs.virginia.edu/stream/FTP/Code/

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

threads [integer] The number of threads to execute if OpenMP is enabled

3.1.66 stress_ng

stress-ng will stress test a computer system in various selectable ways. It was designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces.

stress-ng can also measure test throughput rates; this can be useful to observe performance changes across different operating system releases or types of hardware. However, it has never been intended to be used as a precise benchmark test suite, so do NOT use it in this manner.

The official website for stress-ng is at: http://kernel.ubuntu.com/~cking/stress-ng/

Source code are available from: http://kernel.ubuntu.com/git/cking/stress-ng.git/

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

stressor [str] Stress test case name. The cases listed in allowed values come from the stable release version 0.01.32. The binary included here compiled from dev version 0.06.01. Refer to man page for the definition of each stressor.

allowed values: 'cpu', 'io', 'fork', 'switch', 'vm', 'pipe', 'yield', 'hdd', 'cache', 'sock', 'fallocate', 'flock', 'affinity', 'timer', 'dentry', 'urandom', 'sem', 'open', 'sigq', 'poll'

default: 'cpu'

threads [integer] The number of workers to run. Specifying a negative or zero value will select the number of online processors.

duration [integer] Timeout for test execution in seconds

default: 60

3.1.67 sysbench

SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load.

The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all.

Features of SysBench

- file I/O performance
- scheduler performance
- memory allocation and transfer speed
- POSIX threads implementation performance
- database server performance

See: https://github.com/akopytov/sysbench

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

timeout [integer] timeout for workload execution (adjust from default if running on a slow device and/or specifying a large value for max_requests

default: 300

test [str] sysbench test to run

allowed values: 'fileio', 'cpu', 'memory', 'threads', 'mutex'

default: 'cpu'

threads [integer] The number of threads sysbench will launch

default: 8

num_threads [integer] The number of threads sysbench will launch, overrides threads (old parameter name)

max_requests [integer] The limit for the total number of requests.

max_time [integer] The limit for the total execution time. If neither this nor max_requests is specified, this will default to 30 seconds.

file_test_mode [str] File test mode to use. This should only be specified if `test` is `"fileio"`; if that is the case and `file_test_mode` is not specified, it will default to `"seqwr"` (please see sysbench documentation for explanation of various modes).

allowed values: `'seqwr'`, `'seqrewr'`, `'seqrd'`, `'rndrd'`, `'rndwr'`, `'rndrw'`

cmd_params [str] Additional parameters to be passed to sysbench as a single stiring

force_install [boolean] Always install binary found on the host, even if already installed on device

default: `True`

taskset_mask [integer] The processes spawned by sysbench will be pinned to cores as specified by this parameter

3.1.68 telemetry

Executes Google's Telemetery benchmarking framework

Url: https://www.chromium.org/developers/telemetry

From the web site:

Telemetry is Chrome's performance testing framework. It allows you to perform arbitrary actions on a set of web pages and report metrics about it. The framework abstracts:

- Launching a browser with arbitrary flags on any platform.
- Opening a tab and navigating to the page under test.
- Fetching data via the Inspector timeline and traces.
- Using Web Page Replay to cache real-world websites so they don't change when used in benchmarks.

Design Principles

- Write one performance test that runs on all platforms - Windows, Mac, Linux, Chrome OS, and Android for both Chrome and ContentShell.
- Runs on browser binaries, without a full Chromium checkout, and without having to build the browser yourself.
- Use WebPageReplay to get repeatable test results.
- Clean architecture for writing benchmarks that keeps measurements and use cases separate.
- Run on non-Chrome browsers for comparative studies.

This instrument runs telemetry via its `run_benchmark` script (which must be in PATH or specified using `run_benchmark_path` parameter) and parses metrics from the resulting output.

device setup

The device setup will depend on whether you're running a test image (in which case little or no setup should be necessary)

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

run_benchmark_path [str] This is the path to run_benchmark script which runs a Telemetry benchmark. If not specified, WA will look for Telemetry in its dependencies; if not found there, Telemetry will be downloaded.

test [str] Specifies the telemetry test to run.

default: `'page_cycler.top_10_mobile'`

run_benchmark_params [str] Additional paramters to be passed to `run_benchmark`.

run_timeout [integer] Timeout for execution of the test.

default: `900`

extract_fps [boolean] if `True`, FPS for the run will be computed from the trace (must be enabled).

target_config [str] Manually specify target configuration for telemetry. This must contain –browser option plus any addition options Telemetry requires for a particular target (e.g. –device or –remote)

3.1.69 templerun

Templerun game.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.70 thechase

The Chase demo showcasing the capabilities of Unity game engine.

This demo, is a static video-like game demo, that demonstrates advanced features of the unity game engine. It loops continuously until terminated.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

duration [integer] Duration, in seconds, note that the demo loops the same (roughly) 60 second sceene until stopped.

default: `70`

3.1.71 truckerparking3d

Trucker Parking 3D game.

(yes, apparently that's a thing...)

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `500`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

idle_time [integer] The time you wish the device to remain idle for (if a value is given then this overrides any .run revent file).

check_states [boolean] Use visual state detection to verify the state of the workload after setup and run

assets_push_timeout [integer] Timeout used during deployment of the assets package (if there is one).

default: `500`

3.1.72 vellamo

Android benchmark designed by Qualcomm.

Vellamo began as a mobile web benchmarking tool that today has expanded to include three primary chapters. The Browser Chapter evaluates mobile web browser performance, the Multicore chapter measures the synergy of multiple CPU cores, and the Metal Chapter measures the CPU subsystem performance of mobile processors. Through click-and-go test suites, organized by chapter, Vellamo is designed to evaluate: UX, 3D graphics, and memory read/write and peak bandwidth performance, and much more!

Note: Vellamo v3.0 fails to run on Juno

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

version [str] Specify the version of Vellamo to be run. If not specified, the latest available version will be used.

allowed values: `'3.2.4'`, `'2.0.3'`, `'3.0'`

default: `'3.2.4'`

benchmarks [list_of_strs] Specify which benchmark sections of Vellamo to be run. Only valid on version 3.0 and newer. NOTE: Browser benchmark can be problematic and seem to hang,just wait and it will progress after ~5 minutes

allowed values: `'Browser'`, `'Metal'`, `'Multi'`

default: `['Browser', 'Metal', 'Multi']`

browser [integer] Specify which of the installed browsers will be used for the tests. The number refers to the order in which browsers are listed by Vellamo. E.g. `1` will select the first browser listed, `2` – the second, etc. Only valid for version `3.0`.

default: `1`

3.1.73 video

Plays a video file using the standard android video player for a predetermined duration.

The video can be specified either using `resolution` workload parameter, in which case Big Buck Bunny MP4 video of that resolution will be downloaded and used, or using `filename` parameter, in which case the video file specified will be used.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

play_duration [integer] Playback duration of the video file. This become the duration of the workload.

default: `20`

resolution [str] Specifies which resolution video file to play.

allowed values: `'480p'`, `'720p'`, `'1080p'`

default: `'720p'`

filename [str] The name of the video file to play. This can be either a path to the file anywhere on your file system, or it could be just a name, in which case, the workload will look for it in `~/.workloads_automation/dependency/video` *Note*: either resolution or filename should be specified, but not both!

force_dependency_push [boolean] If true, video will always be pushed to device, regardless of whether the file is already on the device. Default is `False`.

3.1.74 videostreaming

Uses the FREEdi video player to search, stream and play the specified video content from YouTube.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

video_name [str] Name of the video to be played.

resolution [str] Resolution of the video to be played. If video_name is setthis setting will be ignored

allowed values: `'320p'`, `'720p'`, `'1080p'`

default: `'320p'`

sampling_interval [integer] Time interval, in seconds, after which the status of the video playback to be monitoreThe elapsed time of the video playback is monitored after after every `sampling_interval` seconds and compared against the actual time elapsed and the previous sampling point. If the video elapsed time is less that (sampling time - `tolerance`) , then the playback is aborted as the video has not been playing continuously.

default: `20`

tolerance [integer] Specifies the amount, in seconds, by which sampling time is allowed to deviate from elapsed video playback time. If the delta is greater than this value (which could happen due to poor network connection), workload result will be invalidated.

default: `3`

run_timeout [integer] The duration in second for which to play the video

default: `200`

3.1.75 youtube

A workload to perform standard productivity tasks within YouTube.

The workload plays a video from the app, determined by the `video_source` parameter. While the video is playing, a some common actions are done such as video seeking, pausing playback and navigating the comments section.

Test description: The `video_source` parameter determines where the video to be played will be found in the app. Possible values are `search`, `home`, `my_videos`, and `trending`.

-A. search - Goes to the search view, does a search for the given term, and plays the first video in the results. The parameter `search_term` must also be provided in the agenda for this to work. This is the default mode.

-B. home - Scrolls down once on the app's home page to avoid ads (if present, would be first video), then select and plays the video that appears at the top of the list.

-C. my_videos - Goes to the 'My Videos' section of the user's account page and plays a video from there. The user must have at least one uploaded video for this to work.

-D. trending - Goes to the 'Trending Videos' section of the app, and plays the first video in the trending videos list.

For the selected video source, the following test steps are performed:

1. Navigate to the general app settings page to disable autoplay. This improves test stability and predictability by preventing screen transition to load a new video while in the middle of the test.
2. Select the video from the source specified above, and dismiss any potential embedded advert that may pop-up before the actual video.
3. Let the video play for a few seconds, pause it, then resume.

4. Expand the info card that shows video metadata, then collapse it again.
5. Scroll down to the end of related videos and comments under the info card, and then back up to the start. A maximum of 5 swipe actions is performed in either direction.

Known working APK version: 11.19.56

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

install_timeout [integer] Timeout for the installation of the apk.

default: `300`

check_apk [boolean] When set to True the APK file on the host will be prefered if it is a valid version and ABI, if not it will fall back to the version on the targer. When set to False the target version is prefered.

default: `True`

force_install [boolean] Always re-install the APK, even if matching version is found already installed on the device. Runs `adb install -r` to ensure existing APK is replaced. When this is set, check_apk is ignored.

uninstall_apk [boolean] If `True`, will uninstall workload's APK as part of teardown.

exact_abi [boolean] If `True`, workload will check that the APK matches the target device ABI, otherwise any APK found will be used.

clear_data_on_reset [boolean] If set to `False`, this will prevent WA from clearing package data for this workload prior to running it.

default: `True`

markers_enabled [boolean] If `True`, UX_PERF action markers will be emitted to logcat during the test run.

clean_assets [boolean] If `True` pushed assets will be deleted at the end of each iteration

force_push_assets [boolean] If `True` always push assets on each iteration, even if the assets already exists in the device path

video_source [str] Determines where to play the video from. This can either be from the YouTube home, my videos section, trending videos or found in search.

allowed values: `'home'`, `'my_videos'`, `'search'`, `'trending'`

default: `'search'`

search_term [str] The search term to use when `video_source` is set to `search`. Ignored otherwise.

default: `'Big Buck Bunny 60fps 4K - Official Blender Foundation Short Film'`

3.2 Instruments

3.2.1 acmecape

Instrumetnation for the BayLibre ACME cape for power/energy measurment.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

iio_capture [str] Path to the iio-capture binary will be taken from the environment, if not specfied.

host [str] Host name (or IP address) of the ACME cape board.

default: `'baylibre-acme.local'`

iio_device : str

default: `'iio:device0'`

buffer_size [integer] Size of the capture buffer (in KB).

default: `256`

3.2.2 cci_pmu_logger

This instrument allows collecting CCI counter data.

It relies on the pmu_logger.ko kernel driver, the source for which is included with Workload Automation (see inside `wlauto/external` directory). You will need to build this against your specific kernel. Once compiled, it needs to be placed in the dependencies directory (usually `~/.workload_uatomation/dependencies`).

Note: When compling pmu_logger.ko for a new hardware platform, you may need to modify CCI_BASE inside pmu_logger.c to contain the base address of where CCI is mapped in memory on your device.

This instrument relies on `trace-cmd` instrument to also be enabled. You should enable at least `'bprint'` trace event.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

events [list] A list of strings, each representing an event to be counted. The length of the list cannot exceed the number of PMU counters available (4 in CCI-400). If this is not specified, shareable read transactions and snoop hits on both clusters will be counted by default. E.g. `['0x63', '0x83']`.

default: `['0x63', '0x6A', '0x83', '0x8A']`

event_labels [list] A list of labels to be used when reporting PMU counts. If specified, this must be of the same length as `cci_pmu_events`. If not specified, events will be labeled "event_<event_number>".

period [integer] The period (in jiffies) between counter reads.

default: `10`

install_module [boolean] Specifies whether pmu_logger has been compiled as a .ko module that needs to be installed by the instrument. (.ko binary must be in /home/docs/.workload_automation/dependencies). If this is set to `False`, it will be assumed that pmu_logger has been compiled into the kernel, or that it has been installed prior to the invocation of WA.

default: `True`

3.2.3 coreutil

Measures CPU core activity during workload execution in terms of the percentage of time a number of cores were utilized above the specfied threshold.

This workload generates `coreutil.csv` report in the workload's output directory. The report is formatted as follows:

```
<threshold,1core,2core,3core,4core
18.098132,38.650248000000005,10.736180000000001,3.6809760000000002,28.834312000000001
```

Interpretation of the result:

- 38.65% of total time only single core is running above or equal to threshold value
- 10.736% of total time two cores are running simultaneously above or equal to threshold value
- 3.6809% of total time three cores are running simultaneously above or equal to threshold value
- 28.8314% of total time four cores are running simultaneously above or equal to threshold value
- 18.098% of time all core are running below threshold value.

..note : This instrument doesn't work on ARM big.LITTLE IKS implementation

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

threshold [integer] Cores with percentage utilization above this value will be considered as "utilized". This value may need to be adjusted based on the background activity and the intensity of the workload being instrumented (e.g. it may need to be lowered for low-intensity workloads such as video playback).

constraint: `0 < value <= 100`

default: `50`

3.2.4 cpufreq

Collects dynamic frequency (DVFS) settings before and after workload execution.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

paths [list_of_strs] A list of paths to be pulled from the device. These could be directories as well as files.

use_tmpfs [boolean] Specifies whether tmpfs should be used to cache sysfile trees and then pull them down as a tarball. This is significantly faster then just copying the directory trees from the device directly, bur requres root and may not work on all devices. Defaults to `True` if the device is rooted and `False` if it is not.

tmpfs_mount_point [str] Mount point for tmpfs partition used to store snapshots of paths.

tmpfs_size [str] Size of the tempfs partition.

default: `'32m'`

3.2.5 daq

DAQ instrument obtains the power consumption of the target device's core measured by National Instruments Data Acquisition(DAQ) device.

WA communicates with a DAQ device server running on a Windows machine (Please refer to *DAQ Server Guide*) over a network. You must specify the IP address and port the server is listening on in the config file as follows

```
daq_server_host = '10.1.197.176'
daq_server_port = 45677
```

These values will be output by the server when you run it on Windows.

You must also specify the values of resistors (in Ohms) across which the voltages are measured (Please refer to *DAQ Server Guide*). The values should be specified as a list with an entry for each resistor, e.g.:

```
daq_resistor_values = [0.005, 0.005]
```

In addition to this mandatory configuration, you can also optionally specify the following:

```
:daq_labels: Labels to be used for ports. Defaults to ``'PORT_<pnum>'``, where
             'pnum' is the number of the port.
:daq_device_id: The ID under which the DAQ is registered with the driver.
                Defaults to ``'Dev1'``.
:daq_v_range: Specifies the voltage range for the SOC voltage channel on the DAQ
              (please refer to :ref:`daq_setup` for details). Defaults to ``2.5``.
:daq_dv_range: Specifies the voltage range for the resistor voltage channel on
               the DAQ (please refer to :ref:`daq_setup` for details).
               Defaults to ``0.2``.
:daq_sampling_rate: DAQ sampling rate. DAQ will take this many samples each
                    second. Please note that this maybe limitted by your DAQ model
                    and then number of ports you're measuring (again, see
                    :ref:`daq_setup`). Defaults to ``10000``.
:daq_channel_map: Represents mapping from  logical AI channel number to physical
                  connector on the DAQ (varies between DAQ models). The default
                  assumes DAQ 6363 and similar with AI channels on connectors
                  0-7 and 16-23.
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

server_host [str] The host address of the machine that runs the daq Server which the instrument communicates with.

default: `'localhost'`

server_port [integer] The port number for daq Server in which daq instrument communicates with.

default: `45677`

device_id [str] The ID under which the DAQ is registered with the driver.

default: `'Dev1'`

v_range [float] Specifies the voltage range for the SOC voltage channel on the DAQ (please refer to *DAQ Server Guide* for details).

default: `2.5`

dv_range [float] Specifies the voltage range for the resistor voltage channel on the DAQ (please refer to *DAQ Server Guide* for details).

default: `0.2`

sampling_rate [integer] DAQ sampling rate. DAQ will take this many samples each second. Please note that this maybe limitted by your DAQ model and then number of ports you're measuring (again, see *DAQ Server Guide*)

default: `10000`

resistor_values [list (mandatory)] The values of resistors (in Ohms) across which the voltages are measured on each port.

channel_map [list_of_ints] Represents mapping from logical AI channel number to physical connector on the DAQ (varies between DAQ models). The default assumes DAQ 6363 and similar with AI channels on connectors 0-7 and 16-23.

default: `(0, 1, 2, 3, 4, 5, 6, 7, 16, 17, 18, 19, 20, 21, 22, 23)`

labels [list_of_strs] List of port labels. If specified, the lenght of the list must match the length of `resistor_values`. Defaults to "PORT_<pnum>", where "pnum" is the number of the port.

negative_samples [str] Specifies how negative power samples should be handled. The following methods are possible:

keep keep them as they are

zero turn negative values to zero

drop drop samples if they contain negative values. *warning:* this may result in port files containing different numbers of samples

abs take the absoulte value of negave samples

allowed values: `'keep'`, `'zero'`, `'drop'`, `'abs'`

default: `'keep'`

gpio_sync [integer] If specified, the instrument will simultaneously set the specified GPIO pin high and put a marker into ftrace. This is to facillitate syncing kernel trace events to DAQ power trace.

constraint: `value > 0`

merge_channels [dict_or_bool] If set to `True`, channels with consecutive letter suffixes will be summed. e.g. If you have channels A7a, A7b, A7c, A15a, A15b they will be summed to A7, A15

You can also manually specify the name of channels to be merged and the name of the result like so:

merge_channels: A15: [A15dvfs, A15ram] NonCPU: [GPU, RoS, Mem]

In the above exaples the DAQ channels labeled A15a and A15b will be summed together with the results being saved as 'channel' ''a''. A7, GPU and RoS will be summed to 'c'

3.2.6 delay

This instrument introduces a delay before executing either an iteration or all iterations for a spec.

The delay may be specified as either a fixed period or a temperature threshold that must be reached.

Optionally, if an active cooling solution is employed to speed up temperature drop between runs, it may be controlled using this instrument.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

temperature_file [str] Full path to the sysfile on the device that contains the device's temperature.

default: `'/sys/devices/virtual/thermal/thermal_zone0/temp'`

temperature_timeout [integer] The timeout after which the instrument will stop waiting even if the specified threshold temperature is not reached. If this timeout is hit, then a warning will be logged stating the actual temperature at which the timeout has ended.

default: `600`

temperature_poll_period [integer] How long to sleep (in seconds) between polling current device temperature.

default: `5`

temperature_between_specs [integer] Temperature (in device-specific units) the device must cool down to before the iteration spec will be run.

Note: This cannot be specified at the same time as `fixed_between_specs`

---

temperature_between_iterations [integer] Temperature (in device-specific units) the device must cool down to before the next spec will be run.

Note: This cannot be specified at the same time as `fixed_between_iterations`

---

temperature_before_start [integer] Temperature (in device-specific units) the device must cool down to just before the actual workload execution (after setup has been performed).

Note: This cannot be specified at the same time as `fixed_between_iterations`

---

fixed_between_specs [integer] How long to sleep (in seconds) after all iterations for a workload spec have executed.

Note: This cannot be specified at the same time as `temperature_between_specs`

---

fixed_between_iterations [integer] How long to sleep (in seconds) after each iterations for a workload spec has executed.

Note: This cannot be specified at the same time as `temperature_between_iterations`

---

fixed_before_start [integer] How long to sleep (in seconds) after setup for an iteration has been perfromed but before running the workload.

Note: This cannot be specified at the same time as `temperature_before_start`

---

active_cooling [boolean] This instrument supports an active cooling solution while waiting for the device temperature to drop to the threshold. The solution involves an mbed controlling a fan. The mbed is signaled over a serial port. If this solution is present in the setup, this should be set to `True`.

3.2.7 dmesg

Collected dmesg output before and during the run.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

loglevel [integer] Set loglevel for console output.

allowed values: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`

3.2.8 energy_model

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

device_name [caseless_string] The name of the device to be used in generating the model. If not specified, `device.name` will be used.

big_core [caseless_string] The name of the "big" core in the big.LITTLE system; must match one of the values in `device.core_names`.

performance_metric [caseless_string (mandatory)] Metric to be used as the performance indicator.

power_metric [list_or_caseless_string] Metric to be used as the power indicator. The value may contain a `{core}` format specifier that will be replaced with names of big and little cores to drive the name of the metric for that cluster. Ether this or `energy_metric` must be specified but not both.

energy_metric [list_or_caseless_string] Metric to be used as the energy indicator. The value may contain a `{core}` format specifier that will be replaced with names of big and little cores to drive the name of the metric for that cluster. this metric will be used to derive power by deviding through by execution time. Either this or `power_metric` must be specified, but not both.

power_scaling_factor [float] Power model specfies power in milliWatts. This is a scaling factor that power_metric values will be multiplied by to get milliWatts.

default: `1.0`

big_frequencies [list_of_ints] List of frequencies to be used for big cores. These frequencies must be supported by the cores. If this is not specified, all available frequencies for the core (as read from cpufreq) will be used.

little_frequencies [list_of_ints] List of frequencies to be used for little cores. These frequencies must be supported by the cores. If this is not specified, all available frequencies for the core (as read from cpufreq) will be used.

idle_workload [str] Workload to be used while measuring idle power.

default: `'idle'`

idle_workload_params [dict] Parameter to pass to the idle workload.

first_cluster_idle_state [integer] The index of the first cluster idle state on the device. Previous states are assumed to be core idles. The default is `-1`, i.e. only the last idle state is assumed to affect the entire cluster.

default: `-1`

no_hotplug [boolean] This options allows running the instrument without hotpluging cores on and off. Disabling hotplugging will most likely produce a less accurate power model.

num_of_freqs_to_thermal_adjust [integer] The number of frequencies begining from the highest, to be adjusted for the thermal effect.

big_opps [opp_table] OPP table mapping frequency to voltage (kHz –> mV) for the big cluster.

little_opps [opp_table] OPP table mapping frequency to voltage (kHz –> mV) for the little cluster.

big_leakage [integer] Leakage factor for the big cluster (this is specific to a particular core implementation).

default: `120`

little_leakage [integer] Leakage factor for the little cluster (this is specific to a particular core implementation).

default: `60`

3.2.9 energy_probe

Collects power traces using the ARM energy probe.

This instrument requires `caiman` utility to be installed in the workload automation host and be in the PATH. Caiman is part of DS-5 and should be in `/path/to/DS-5/bin/` . Energy probe can simultaneously collect energy from up to 3 power rails.

To connect the energy probe on a rail, connect the white wire to the pin that is closer to the Voltage source and the black wire to the pin that is closer to the load (the SoC or the device you are probing). Between the pins there should be a shunt resistor of known resistance in the range of 5 to 20 mOhm. The resistance of the shunt resistors is a mandatory parameter `resistor_values`.

Note: This instrument can process results a lot faster if python pandas is installed.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

resistor_values [list_of_numbers] The value of shunt resistors. This is a mandatory parameter.

labels [list] Meaningful labels for each of the monitored rails.

device_entry [str] Path to /dev entry for the energy probe (it should be /dev/ttyACMx)

default: `'/dev/ttyACM0'`

3.2.10 execution_time

Measure how long it took to execute the run() methods of a Workload.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.2.11 file_poller

Polls the given files at a set sample interval. The values are output in CSV format.

This instrument places a file called poller.csv in each iterations result directory. This file will contain a timestamp column which will be in uS, the rest of the columns will be the contents of the polled files at that time.

This instrument will strip any commas or new lines for the files' values before writing them.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

sample_interval [integer] The interval between samples in mS.

default: `1000`

files [list_or_string (mandatory)] A list of paths to the files to be polled

labels [list_or_string] A list of lables to be used in the CSV output for the corresponding files. This cannot be used if a * wildcard is used in a path.

as_root [boolean] Whether or not the poller will be run as root. This should be used when the file you need to poll can only be accessed by root.

3.2.12 fps

Measures Frames Per Second (FPS) and associated metrics for a workload.

Note: This instrument depends on pandas Python library (which is not part of standard WA dependencies), so you will need to install that first, before you can use it.

Android L and below use SurfaceFlinger to calculate the FPS data. Android M and above use gfxinfo to calculate the FPS data.

SurfaceFlinger: The view is specified by the workload as `view` attribute. This defaults to `'SurfaceView'` for game workloads, and `None` for non-game workloads (as for them FPS mesurement usually doesn't make sense). Individual workloads may override this.

gfxinfo: The view is specified by the workload as `package` attribute. This is because gfxinfo already processes for all views in a package.

This instrument adds four metrics to the results:

FPS Frames Per Second. This is the frame rate of the workload.

frame_count The total number of frames rendered during the execution of the workload.

janks The number of "janks" that occured during execution of the workload. Janks are sudden shifts in frame rate. They result in a "stuttery" UI. See http://jankfree.org/jank-busters-io

not_at_vsync The number of frames that did not render in a single vsync cycle.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

drop_threshold [numeric] Data points below this FPS will be dropped as they do not constitute "real" gameplay. The assumption being that while actually running, the FPS in the game will not drop below X frames per second, except on loading screens, menus, etc, which should not contribute to FPS calculation.

default: `5`

keep_raw [boolean] If set to `True`, this will keep the raw dumpsys output in the results directory (this is maily used for debugging) Note: frames.csv with collected frames data will always be generated regardless of this setting.

generate_csv [boolean] If set to `True`, this will produce temporal fps data in the results directory, in a file named fps.csv Note: fps data will appear as discrete step-like values in order to produce a more meainingfull representation,a rolling mean can be applied.

default: `True`

crash_check [boolean] Specifies wither the instrument should check for crashed content by examining frame data. If this is set, `execution_time` instrument must also be installed. The check is performed by using the measured FPS and exection time to estimate the expected frames cound and comparing that against the measured frames count. The the ratio of measured/expected is too low, then it is assumed that the content has crashed part way during the run. What is "too low" is determined by `crash_threshold`.

Note: This is not 100% fool-proof. If the crash occurs sufficiently close to workload's termination, it may not be detected. If this is expected, the threshold may be adjusted up to compensate.

---

default: `True`

crash_threshold [float] Specifies the threshold used to decided whether a measured/expected frames ration indicates a content crash. E.g. a value of `0.75` means the number of actual frames counted is a quarter lower than expected, it will treated as a content crash.

default: `0.7`

dumpsys_period [float] Specifies the time period between calls to `dumpsys SurfaceFlinger --latency` in seconds when collecting frame data. Using a lower value improves the granularity of timings when recording actions that take a short time to complete. Note, this will produce duplicate frame data in the raw dumpsys output, however, this is filtered out in frames.csv. It may also affect the overall load on the system.

The default value of 2 seconds corresponds with the NUM_FRAME_RECORDS in android/services/surfaceflinger/FrameTracker.h (as of the time of writing currently 128) and a frame rate of 60 fps that is applicable to most devices.

constraint: `value > 0`

default: `2`

force_surfaceflinger [boolean] By default, the method to capture fps data is based on Android version. If this is set to true, force the instrument to use the SurfaceFlinger method regardless of its Android version.

3.2.13 freq_sweep

Sweeps workloads through all available frequencies on a device.

When enabled this instrument will take all workloads specified in an agenda and run them at all available frequencies for all clusters.

Recommendations:

- Setting the runner to 'by_spec' increases the chance of successfully completing an agenda without encountering hotplug issues
- If possible disable dynamic hotplug on the target device
- This instrument does not automatically pin workloads to the cores being swept since it is not aware of what the workloads do. To achieve this use the workload's taskset parameter (if it has one).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

sweeps [list] By default this instrument will sweep across all available frequencies for all available clusters. If you wish to only sweep across certain frequencies on particular clusters you can do so by specifying this parameter.

Sweeps should be a lists of dictionaries that can contain:

- Cluster (mandatory): The name of the cluster this sweep will be performed on. E.g *A7*
- Frequencies: A list of frequencies (in KHz) to use. If this is not provided all frequencies available for this cluster will be used. E.g: *[800000, 900000, 100000]*
- label: Workload specs will be named *{spec id}_{label}_{frequency}*. If a label is not provided it will be named *sweep{sweep No.}*

Example sweep specification:

```
freq_sweep:
    sweeps:
        - cluster: A53
          label: littles
          frequencies: [800000, 900000, 100000]
        - cluster: A57
          label: bigs
```

3.2.14 hwmon

Hardware Monitor (hwmon) is a generic Linux kernel subsystem, providing access to hardware monitoring components like temperature or voltage/current sensors.

The following web page has more information:

http://blogs.arm.com/software-enablement/925-linux-hwmon-power-management-and-arm-ds-5-streamline/

You can specify which sensors HwmonInstrument looks for by specifying hwmon_sensors in your config.py, e.g.

```
hwmon_sensors = ['energy', 'temp']
```

If this setting is not specified, it will look for all sensors it knows about. Current valid values are:

```
:energy: Collect energy measurements and report energy consumed
         during run execution (the diff of before and after readings)
         in Joules.
:temp: Collect temperature measurements and report the before and
       after readings in degrees Celsius.
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

sensors [list_of_strs] The kinds of sensors hwmon instrument will look for

allowed values: `'energy'`, `'temp'`

default: `['energy', 'temp']`

3.2.15 interrupts

Pulls the `/proc/interrupts` file before and after workload execution and diffs them to show what interrupts occurred during that time.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.2.16 juno_energy

Collects internal energy meter measurements from Juno development board.

This instrument was created because (at the time of creation) Juno's energy meter measurements aren't exposed through HWMON or similar standardized mechanism, necessitating a dedicated instrument to access them.

This instrument, and the `readenergy` executable it relies on are very much tied to the Juno platform and are not expected to work on other boards.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

period [float] Specifies the time, in Seconds, between polling energy counters.

default: `0.1`

strict [boolean] Setting this to `False` will omit the check that the `device` is `"juno"`. This is useful if the underlying board is actually Juno but WA connects via a different interface (e.g. `generic_linux`).

default: `True`

3.2.17 netstats

Measures transmit/receive network traffic on an Android divice on per-package basis.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

packages [list_of_strs] List of Android packages who's traffic will be monitored. If unspecified, all packages in the device will be monitorred.

period [integer] Polling period for instrumentation on the device. Traffic statistics will be updated every `period` seconds.

default: `5`

force_reinstall [boolean] If `True`, instrumentation APK will always be re-installed even if it already installed on the device.

uninstall_on_completion [boolean] If `True`, instrumentation will be uninstalled upon run completion.

3.2.18 perf

Perf is a Linux profiling with performance counters.

Performance counters are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted. They form a basis for profiling applications to trace dynamic control flow and identify hotspots.

pref accepts options and events. If no option is given the default '-a' is used. For events, the default events are migrations and cs. They both can be specified in the config file.

Events must be provided as a list that contains them and they will look like this

```
perf_events = ['migrations', 'cs']
```

Events can be obtained by typing the following in the command line on the device

```
perf list
```

Whereas options, they can be provided as a single string as following

```
perf_options = '-a -i'
```

Options can be obtained by running the following in the command line

```
man perf-record
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

events [list_of_strs] Specifies the events to be counted.

constraint: `must not be empty.`

default: `['migrations', 'cs']`

optionstring [list_or_string] Specifies options to be used for the perf command. This may be a list of option strings, in which case, multiple instances of perf will be kicked off – one for each option string. This may be used to e.g. collected different events from different big.LITTLE clusters.

default: `'-a'`

labels [list_of_strs] Provides labels for pref output. If specified, the number of labels must match the number of `optionstrings`.

force_install [boolean] always install perf binary even if perf is already present on the device.

3.2.19 screenon

Ensure screen is on before each iteration on Android devices.

A very basic instrument that checks that the screen is on on android devices. Optionally, it call poll the device periodically to ensure that the screen is still on.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

polling_period [integer] Set this to a non-zero value to enable periodic (every `polling_period` seconds) polling of the screen on the device to ensure it is on during a run.

3.2.20 servo_power

Collects power traces using the Chromium OS Servo Board.

Servo is a debug board used for Chromium OS test and development. Among other uses, it allows access to the built in power monitors (if present) of a Chrome OS device. More information on Servo board can be found in the link bellow:

https://www.chromium.org/chromium-os/servo

In order to use this instrument you need to be a sudoer and you need a chroot environment. More information on the chroot environment can be found on the link bellow:

https://www.chromium.org/chromium-os/developer-guide

If you wish to run servod on a remote machine you will need to allow it to accept external connections using the *–host* command line option, like so: *sudo servod -b some_board -c some_board.xml –host=’‘*

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

power_domains [list_of_strs] The names of power domains to be monitored by the instrument using servod.

labels [list_of_strs] Meaningful labels for each of the monitored domains.

chroot_path [str] Path to chroot direcory on the host.

sampling_rate [integer] Samples per second.

default: `10`

board_name [str (mandatory)] The name of the board under test.

autostart [boolean] Automatically start *servod*. Set to *False* if you want to use an already running *servod* instance or a remote servo

default: `True`

host [str] When *autostart* is set to *False* you can specify the host on which *servod* is running allowing you to remotelly access as servo board.

if *autostart* is *True* this parameter is ignored and *localhost* is used instead

default: `'localhost'`

port [integer] When *autostart* is set to false you must provide the port that *servod* is running on

If *autostart* is *True* this parameter is ignored and the port output during the startup of *servod* will be used.

default: `9999`

vid [str] When more than one servo is plugged in, you must provide a vid/pid pair to identify the servio you wish to use.

pid [str] When more than one servo is plugged in, you must provide a vid/pid pair to identify the servio you wish to use.

3.2.21 streamline

Collect Streamline traces from the device.

Note: This instrument supports streamline that comes with DS-5 5.17 and later earlier versions of streamline may not work correctly (or at all).

This Instrument allows collecting streamline traces (such as PMU counter values) from the device. It assumes you have DS-5 (which Streamline is part of) installed on your system, and that streamline command is somewhere in PATH.

Streamline works by connecting to gator service on the device. gator comes in two parts a driver (gator.ko) and daemon (gatord). The driver needs to be compiled against your kernel and both driver and daemon need to be compatible with your version of Streamline. The best way to ensure compatibility is to build them from source which came with your DS-5. gator source can be found in

```
/usr/local/DS-5/arm/gator
```

(the exact path may vary depending of where you have installed DS-5.) Please refer to the README the accompanies the source for instructions on how to build it.

Once you have built the driver and the daemon, place the binaries into your ~/.workload_automation/streamline/ directory (if you haven't tried running WA with this instrument before, the streamline/ subdirectory might not exist, in which case you will need to create it.

In order to specify which events should be captured, you need to provide a configuration.xml for the gator. The easiest way to obtain this file is to export it from event configuration dialog in DS-5 streamline GUI. The file should be called "configuration.xml" and it be placed in the same directory as the gator binaries.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

port [str] Specifies the port on which streamline will connect to gator

default: `'8080'`

configxml [str] streamline configuration XML file to be used. This must be an absolute path, though it may count the user home symbol (~)

report [boolean] Specifies whether a report should be generated from streamline data.

report_options [str] A string with options that will be added to streamline -report command.

default: `'-format csv'`

3.2.22 sysfs_extractor

Collects the contest of a set of directories, before and after workload execution and diffs the result.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

paths [list_of_strs (mandatory)] A list of paths to be pulled from the device. These could be directories as well as files.

use_tmpfs [boolean] Specifies whether tmpfs should be used to cache sysfile trees and then pull them down as a tarball. This is significantly faster then just copying the directory trees from the device directly, bur requres root and may not work on all devices. Defaults to `True` if the device is rooted and `False` if it is not.

tmpfs_mount_point [str] Mount point for tmpfs partition used to store snapshots of paths.

tmpfs_size [str] Size of the tempfs partition.

default: `'32m'`

3.2.23 systrace

This instrument uses systrace.py from the android SDK to dump atrace output.

Note: This is unlikely to work on devices that have an android build built before 15-May-2015. Before this date there was a bug with running atrace asynchronously.

From developer.android.com: The Systrace tool helps analyze the performance of your application by capturing and displaying execution times of your applications processes and other Android system processes. The tool combines data from the Android kernel such as the CPU scheduler, disk activity, and application threads to generate an HTML report that shows an overall picture of an Android device's system processes for a given period of time.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

buffer_size [integer] Use a trace buffer size of N kilobytes. This option lets you limit the total size of the data collected during a trace.

default: `1024`

use_circular_buffer [boolean] When true trace data will be put into a circular buffer such that when it overflows it will start overwriting the beginning of the buffer.

kernel_functions [list_of_strs] Specify the names of kernel functions to trace.

categories [list_of_strs] A list of the categories you wish to trace.

default: `['freq', 'sched']`

app_names [list_of_strs] Enable tracing for applications, specified as a comma-separated list of package names. The apps must contain tracing instrumentation calls from the Trace class. For more information, see http://developer.android.com/tools/debugging/systrace.html#app-trace

ignore_signals [boolean] This will cause atrace to ignore `SIGHUP`, `SIGINT`, `SIGQUIT` and `SIGTERM`.

compress_trace [boolean] Compresses atrace output. This *greatly* decreases the time it takes to pull results from a device but the resulting txt file is not human readable.

default: `True`

3.2.24 trace-cmd

trace-cmd is an instrument which interacts with ftrace Linux kernel internal tracer

From trace-cmd man page:

trace-cmd command interacts with the ftrace tracer that is built inside the Linux kernel. It interfaces with the ftrace specific files found in the debugfs file system under the tracing directory.

trace-cmd reads a list of events it will trace, which can be specified in the config file as follows

```
trace_events = ['irq*', 'power*']
```

If no event is specified, a default set of events that are generally considered useful for debugging/profiling purposes will be enabled.

The list of available events can be obtained by rooting and running the following command line on the device

```
trace-cmd list
```

You may also specify `trace_buffer_size` setting which must be an integer that will be used to set the ftrace buffer size. It will be interpreted as KB:

```
trace_cmd_buffer_size = 8000
```

The maximum buffer size varies from device to device, but there is a maximum and trying to set buffer size beyond that will fail. If you plan on collecting a lot of trace over long periods of time, the buffer size will not be enough and you will only get trace for the last portion of your run. To deal with this you can set the `trace_mode` setting to `'record'` (the default is `'start'`):

```
trace_cmd_mode = 'record'
```

This will cause trace-cmd to trace into file(s) on disk, rather than the buffer, and so the limit for the max size of the trace is set by the storage available on device. Bear in mind that `'record'` mode *is* more intrusive than the default, so if you do not plan on generating a lot of trace, it is best to use the default `'start'` mode.

Note: Mode names correspond to the underlying trace-cmd executable's command used to implement them. You can find out more about what is happening in each case from trace-cmd documentation: https://lwn.net/Articles/341902/.

This instrument comes with an trace-cmd binary that will be copied and used on the device, however post-processing will be, by default, done on-host and you must have trace-cmd installed and in your path. On Ubuntu systems, this may be done with:

```
sudo apt-get install trace-cmd
```

Alternatively, you may set `report_on_target` parameter to `True` to enable on-target processing (this is useful when running on non-Linux hosts, but is likely to take longer and may fail on particularly resource-constrained targets).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

events [list] Specifies the list of events to be traced. Each event in the list will be passed to trace-cmd with -e parameter and must be in the format accepted by trace-cmd.

default: `['sched*', 'irq*', 'power*']`

mode [str] Trace can be collected using either 'start' or 'record' trace-cmd commands. In 'start' mode, trace will be collected into the ftrace buffer; in 'record' mode, trace will be written into a file on the device's file system. 'start' mode is (in theory) less intrusive than 'record' mode, however it is limited by the size of the ftrace buffer (which is configurable – see `buffer_size` – but only up to a point) and that may overflow for long-running workloads, which will result in dropped events.

allowed values: `'start'`, `'record'`

default: `'start'`

buffer_size [integer] Attempt to set ftrace buffer size to the specified value (in KB). Default buffer size may need to be increased for long-running workloads, or if a large number of events have been enabled. Note: there is a maximum size that the buffer can be set, and that varies from device to device. Attempting to set buffer size higher than this will fail. In that case, this instrument will set the size to the highest possible value by going down from the specified size in `buffer_size_step` intervals.

buffer_size_step [integer] Defines the decremental step used if the specified `buffer_size` could not be set. This will be subtracted form the buffer size until set succeeds or size is reduced to 1MB.

default: `1000`

buffer_size_file [str] Path to the debugs file that may be used to set ftrace buffer size. This should need to be modified for the vast majority devices.

default: `'/sys/kernel/debug/tracing/buffer_size_kb'`

report [boolean] Specifies whether reporting should be performed once the binary trace has been generated.

default: `True`

no_install [boolean] Do not install the bundled trace-cmd and use the one on the device instead. If there is not already a trace-cmd on the device, an error is raised.

report_on_target [boolean] When enabled generation of reports will be done host-side because the generated file is very large. If trace-cmd is not available on the host device this setting and be disabled and the report will be generated on the target device.

Note: This requires the latest version of trace-cmd to be installed on the host (the one in your distribution's repos may be too old).

3.3 Result_processors

3.3.1 cpustates

Process power ftrace to produce CPU state and parallelism stats.

Parses trace-cmd output to extract power events and uses those to generate statistics about parallelism and frequency/idle core residency.

Note: trace-cmd instrument must be enabled and configured to collect at least `power:cpu_idle` and `power:cpu_frequency` events. Reporting should also be enabled (it is by default) as `cpustate` parses the text version of the trace. Finally, the device should have `cpuidle` module installed.

This generates two reports for the run:

parallel.csv

Shows what percentage of time was spent with N cores active (for N from 0 to the total number of cores), for a cluster or for a system as a whole. It contain the following columns:

workload The workload label

iteration iteration that was run

cluster The cluster for which statics are reported. The value of `"all"` indicates that this row reports statistics for the whole system.

number_of_cores number of cores active. `0` indicates the cluster was idle.

total_time Total time spent in this state during workload execution

%time Percentage of total workload execution time spent in this state

%running_time Percentage of the time the cluster was active (i.e. ignoring time the cluster was idling) spent in this state.

cpustate.csv

Shows percentage of the time a core spent in a particular power state. The first column names the state is followed by a column for each core. Power states include available DVFS frequencies (for heterogeneous systems, this is the union of frequencies supported by different core types) and idle states. Some shallow states (e.g. ARM WFI) will consume different amount of power depending on the current OPP. For such states, there will be an entry for each opp. `"unknown"` indicates the percentage of time for which a state could not be established from the trace. This is usually due to core state being unknown at the beginning of the trace, but may also be caused by dropped events in the middle of the trace.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

first_cluster_state [integer] The first idle state which is common to a cluster.

default: `2`

first_system_state [integer] The first idle state which is common to all cores.

default: `3`

write_iteration_reports [boolean] By default, this instrument will generate reports for the entire run in the overall output directory. Enabling this option will, in addition, create reports in each iteration's output directory. The formats of these reports will be similar to the overall report, except they won't mention the workload name or iteration number (as that is implied by their location).

use_ratios [boolean] By default proportional values will be reported as percentages, if this flag is enabled, they will be reported as ratios instead.

create_timeline [boolean] Create a CSV with the timeline of core power states over the course of the run as well as the usual stats reports.

default: `True`

create_utilization_timeline [boolean] Create a CSV with the timeline of cpu(s) utilisation over the course of the run as well as the usual stats reports. The values generated are floating point numbers, normalised based on the maximum frequency of the cluster.

start_marker_handling [str] The trace-cmd instrument inserts a marker into the trace to indicate the beginning of workload execution. In some cases, this marker may be missing in the final output (e.g. due to trace buffer overrun). This parameter specifies how a missing start marker will be handled:

ignore The start marker will be ignored. All events in the trace will be used.

error An error will be raised if the start marker is not found in the trace.

try If the start marker is not found, all events in the trace will be used.

allowed values: `'ignore'`, `'try'`, `'error'`

default: `'try'`

no_idle [boolean] Indicate that there will be no idle transitions in the trace. By default, a core will be reported as being in an "unknown" state until the first idle transtion for that core. Normally, this is not an issue, as cores are "nudged" as part of the setup to ensure that there is an idle transtion before the meassured region. However, if all idle states for the core have been disabled, or if the kernel does not have cpuidle, the nudge will not result in an idle transition, which would cause the cores to be reported to be in "unknown" state for the entire execution.

If this parameter is set to `True`, the processor will assuming that cores are running prior to the begining of the issue, and they will leave unknown state on the first frequency transition.

3.3.2 csv

Creates a `results.csv` in the output directory containing results for all iterations in CSV format, each line containing a single metric.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

use_all_classifiers [boolean] If set to `True`, this will add a column for every classifier that features in at least one collected metric.

Note: This cannot be `True` if `extra_columns` is set.

---

extra_columns [list_of_strs] List of classifiers to use as columns.

Note: This cannot be set if `use_all_classifiers` is `True`.

3.3.3 dvfs

Reports DVFS state residency data form ftrace power events.

This generates a `dvfs.csv` in the top-level results directory that, for each workload iteration, reports the percentage of time each CPU core spent in each of the DVFS frequency states (P-states), as well as percentage of the time spent in idle, during the execution of the workload.

Note: `trace-cmd` instrument *MUST* be enabled in the instrumentation, and at least `'power*'` events must be enabled.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.4 ipynb_exporter

Generates an IPython notebook from a template with the results and runs it. Optionally it can show the resulting notebook in a web browser. It can also generate a PDF from the notebook.

The template syntax is that of jinja2 and the template should generate a valid ipython notebook. The templates receives `result` and `context` which correspond to the RunResult and ExecutionContext respectively. You can use those in your ipython notebook template to extract any information you want to parse or show.

This results_processor depends on `ipython` and `python-jinja2` being installed on the system.

For example, a simple template that plots a bar graph of the results is:

```
{
 "metadata": {
  "name": ""
 },
 "nbformat": 3,
 "nbformat_minor": 0,
 "worksheets": [
  {
   "cells": [
    {
     "cell_type": "code",
     "collapsed": false,
     "input": [
      "%pylab inline"
     ],
     "language": "python",
     "metadata": {},
     "outputs": [],
     "prompt_number": 1
    },
    {
```

```
     "cell_type": "code",
     "collapsed": false,
     "input": [
      "results = {",
      {% for ir in result.iteration_results -%}
        {% for metric in ir.metrics -%}
          {% if metric.name in ir.workload.summary_metrics or not ir.workload.summary_
↪metrics -%}
      "\"{{ ir.spec.label }}_{{ ir.id }}_{{ ir.iteration }}_{{ metric.name }}\": {{␣
↪metric.value }}, ",
          {%- endif %}
        {%- endfor %}
      {%- endfor %}
      "}\n",
      "width = 0.7\n",
      "ind = np.arange(len(results))"
     ],
     "language": "python",
     "metadata": {},
     "outputs": [],
     "prompt_number": 2
    },
    {
     "cell_type": "code",
     "collapsed": false,
     "input": [
      "fig, ax = plt.subplots()\n",
      "ax.bar(ind, results.values(), width)\n",
      "ax.set_xticks(ind + width/2)\n",
      "_ = ax.set_xticklabels(results.keys())"
     ],
     "language": "python",
     "metadata": {},
     "outputs": [],
     "prompt_number": 3
    }
   ],
   "metadata": {}
  }
 ]
}
```

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

notebook_template [file_path] Filename of the ipython notebook template. If no *notebook_template* is specified, the example template above is used.

default: `'template.ipynb'`

notebook_name_prefix [str] Prefix of the name of the notebook. The date, time and `.ipynb` are appended to form the notebook filename. E.g. if notebook_name_prefix is `result_` then a run on 13th April 2015 at 9:54 would generate a notebook called `result_150413-095400.ipynb`. When generating a PDF, the resulting file will have the same name, but ending in `.pdf`.

default: `'result_'`

show_notebook [boolean] Open a web browser with the resulting notebook.

notebook_directory [file_path] Path to the notebooks directory served by the ipython notebook server. You must set it if `show_notebook` is selected. The ipython notebook will be copied here if specified.

notebook_url [str] URL of the notebook on the IPython server. If not specified, it will be assumed to be in the root notebooks location on localhost, served on port 8888. Only needed if `show_notebook` is selected.

Note: the URL should not contain the final part (the notebook name) which will be populated automatically.

---

default: `'http://localhost:8888/notebooks'`

convert_to_html [boolean] Convert the resulting notebook to HTML.

show_html [boolean] Open the exported html notebook at the end of the run. This can only be selected if convert_to_html has also been selected.

convert_to_pdf [boolean] Convert the resulting notebook to PDF.

show_pdf [boolean] Open the pdf at the end of the run. This can only be selected if convert_to_pdf has also been selected.

3.3.5 json

Creates a `results.json` in the output directory containing results for all iterations in JSON format.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.6 mongodb

Uploads run results to a MongoDB instance.

MongoDB is a popular document-based data store (NoSQL database).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

uri [str] Connection URI. If specified, this will be used for connecting to the backend, and host/port parameters will be ignored.

host [str (mandatory)] IP address/name of the machinge hosting the MongoDB server.

default: `'localhost'`

port [integer (mandatory)] Port on which the MongoDB server is listening.

default: `27017`

db [str (mandatory)] Database on the server used to store WA results.

default: `'wa'`

extra_params [dict] Additional connection parameters may be specfied using this (see pymongo documentation.

authentication [dict] If specified, this will be passed to db.authenticate() upon connection; please pymongo documentaion authentication examples for detail.

3.3.7 notify

Display a desktop notification when the run finishes

Notifications only work in linux systems. It uses the generic freedesktop notification specification. For this results processor to work, you need to have python-notify installed in your system.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.8 sqlite

Stores results in an sqlite database.

This may be used accumulate results of multiple runs in a single file.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

database [str] Full path to the sqlite database to be used. If this is not specified then a new database file will be created in the output directory. This setting can be used to accumulate results from multiple runs in a single database. If the specified file does not exist, it will be created, however the directory of the file must exist.

Note: The value must resolve to an absolute path, relative paths are not allowed; however the value may contain environment variables and/or the home reference ~.

---

overwrite [boolean] If `True`, this will overwrite the database file if it already exists. If `False` (the default) data will be added to the existing file (provided schema versions match – otherwise an error will be raised).

3.3.9 standard

Creates a `result.txt` file for every iteration that contains metrics for that iteration.

The metrics are written in

```
metric = value [units]
```

format.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.10 status

Outputs a txt file containing general status information about which runs failed and which were successful

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.11 summary_csv

Similar to csv result processor, but only contains workloads' summary metrics.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

3.3.12 syeg_csv

Generates a CSV results file in the format expected by SYEG toolchain.

Multiple iterations get parsed into columns, adds additional columns for mean and standard deviation, append number of threads to metric names (where applicable) and add some metadata based on external mapping files.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

outfile [str] The name of the output CSV file.

default: `'syeg_out.csv'`

3.3.13 uxperf

Parse logcat for UX_PERF markers to produce performance metrics for workload actions using specified instrumentation.

An action represents a series of UI interactions to capture.

NOTE: The UX_PERF markers are turned off by default and must be enabled in a agenda file by setting `markers_enabled` for the workload to `True`.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

add_timings [boolean] If set to `True`, add per-action timings to result metrics.'

default: `True`

add_frames [boolean] If set to `True`, add per-action frame statistics to result metrics. i.e. fps, frame_count, jank and not_at_vsync.

NOTE: This option requires the fps instrument to be enabled.

drop_threshold [numeric] Data points below this FPS will be dropped as they do not constitute "real" gameplay. The assumption being that while actually running, the FPS in the game will not drop below X frames per second, except on loading screens, menus, etc, which should not contribute to FPS calculation.

default: `5`

generate_csv [boolean] If set to `True`, this will produce temporal per-action fps data in the results directory, in a file named <action>_fps.csv.

Note: per-action fps data will appear as discrete step-like values in order to produce a more meainingfull representation, a rolling mean can be applied.

default: `True`

3.4 Devices

3.4.1 Nexus10

Nexus10 is a 10 inch tablet device, which has dual-core A15.

To be able to use Nexus10 in WA, the following must be true:

- USB Debugging Mode is enabled.
- Generate USB debugging authorisation for the host machine

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['A15', 'A15']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

> **Note:** most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/sdcard/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

3.4.2 Nexus5

Adapter for Nexus 5.

To be able to use Nexus5 in WA, the following must be true:

- USB Debugging Mode is enabled.
- Generate USB debugging authorisation for the host machine

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['krait400', 'krait400', 'krait400', 'krait400']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

> **Note:** most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

allowed values: 'unknown', 'smp', 'hmp', 'iks', 'ea', 'other'

default: 'unknown'

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. scheduler != 'iks'). If left unset for IKS devices, it will default to 800000, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: ['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']

binaries_directory [str] Location of binaries on the device.

default: '/data/local/tmp/wa-bin'

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: '/sdcard/wa-working'

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: r'^.*(shell|root)@.*:/\S* [#$] '

package_data_directory [str] Location of of data for an installed package (APK).

default: '/data/data'

external_storage_directory [str] Mount point for external storage.

default: '/sdcard'

connection [str] Specified the nature of adb connection.

allowed values: 'usb', 'ethernet'

default: 'usb'

logcat_poll_period [integer] If specified and is not 0, logcat will be polled every logcat_poll_period seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: None, 'horizontal', 'vertical'

3.4.3 Note3

Adapter for Galaxy Note 3.

To be able to use Note3 in WA, the following must be true:

- USB Debugging Mode is enabled.
- Generate USB debugging authorisation for the host machine

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['A15', 'A15', 'A15', 'A15']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/storage/sdcard0/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

3.4.4 TC2

TC2 is a development board, which has three A7 cores and two A15 cores.

TC2 has a number of boot parameters which are:

root_mount Defaults to '/media/VEMSD'

boot_firmware It has only two boot firmware options, which are uefi and bootmon. Defaults to 'uefi'.

fs_medium Defaults to 'usb'.

device_working_directory The direcitory that WA will be using to copy files to. Defaults to 'data/local/usecase'

serial_device The serial device which TC2 is connected to. Defaults to '/dev/ttyS0'.

serial_baud Defaults to 38400.

serial_max_timeout Serial timeout value in seconds. Defaults to 600.

serial_log Defaults to standard output.

init_timeout The timeout in seconds to init the device. Defaults set to 30.

always_delete_uefi_entry If true, it will delete the ufi entry. Defaults to True.

psci_enable Enabling the psci. Defaults to True.

host_working_directory The host working directory. Defaults to None.

disable_boot_configuration Disables boot configuration through images.txt and board.txt. When this is `True`, those two files will not be overwritten in VEMSD. This option may be necessary if the firmware version in the `TC2` is not compatible with the templates in WA. Please note that enabling this will prevent you form being able to set `boot_firmware` and `mode` parameters. Defaults to `False`.

TC2 can also have a number of different booting mode, which are:

mp_a7_only Only the A7 cluster.

mp_a7_bootcluster Both A7 and A15 clusters, but it boots on A7 cluster.

mp_a15_only Only the A15 cluster.

mp_a15_bootcluster Both A7 and A15 clusters, but it boots on A15 clusters.

iks_cpu Only A7 cluster with only 2 cpus.

iks_a15 Only A15 cluster.

iks_a7 Same as iks_cpu

iks_ns_a15 Both A7 and A15 clusters.

iks_ns_a7 Both A7 and A15 clusters.

The difference between mp and iks is the scheduling policy.

TC2 takes the following runtime parameters

a7_cores Number of active A7 cores.

a15_cores Number of active A15 cores.

a7_governor CPUFreq governor for the A7 cluster.

a15_governor CPUFreq governor for the A15 cluster.

a7_min_frequency Minimum CPU frequency for the A7 cluster.

a15_min_frequency Minimum CPU frequency for the A15 cluster.

a7_max_frequency Maximum CPU frequency for the A7 cluster.

a15_max_frequency Maximum CPU frequency for the A7 cluster.

irq_affinity lambda x: Which cluster will receive IRQs.

cpuidle Whether idle states should be enabled.

sysfile_values A dict mapping a complete file path to the value that should be echo'd into it. By default, the file will be subsequently read to verify that the value was written into it with DeviceError raised otherwise. For write-only files, this check can be disabled by appending a `!` to the end of the file path.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings] This parameter will be ignored for TC2

core_clusters [list_of_ints] This parameter will be ignored for TC2

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'hmp'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/sdcard/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

3.4.5 XE503C12

A developer-unlocked Samsung XE503C12 running sshd.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['a15', 'a15', 'a15', 'a15']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of executable binaries on this device (must be in PATH).

default: `'/home/chronos/bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

host [str (mandatory)] Host name or IP address for the device.

username [str (mandatory)] User name for the account on the device.

default: `'chronos'`

password [str] Password for the account on the device (for password-based auth).

keyfile [str] Keyfile to be used for key-based authentication.

port [integer] SSH port number on the device.

default: `22`

password_prompt [str] Prompt presented by sudo when requesting the password.

default: `'Password:'`

use_telnet [boolean] Optionally, telnet may be used instead of ssh, though this is discouraged.

boot_timeout [integer] How long to try to connect to the device after a reboot.

default: `120`

3.4.6 chromeos_test_image

Chrome OS test image device. Use this if you are working on a Chrome OS device with a test image. An off the shelf device will not work with this device interface.

More information on how to build a Chrome OS test image can be found here:

https://www.chromium.org/chromium-os/developer-guide#TOC-Build-a-disk-image-for-your-board

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of executable binaries on this device (must be in PATH).

default: `'/usr/local/bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/home/root/wa-working'`

host [str (mandatory)] Host name or IP address for the device.

username [str (mandatory)] User name for the account on the device.

default: `'root'`

password [str] Password for the account on the device (for password-based auth).

keyfile [str] Keyfile to be used for key-based authentication.

port [integer] SSH port number on the device.

default: `22`

password_prompt [str] Prompt presented by sudo when requesting the password.

default: `'Password:'`

use_telnet [boolean] Optionally, telnet may be used instead of ssh, though this is discouraged.

boot_timeout [integer] How long to try to connect to the device after a reboot.

default: `120`

3.4.7 gem5_android

Implements gem5 Android device.

This class allows a user to connect WA to a simulation using gem5. The connection to the device is made using the telnet connection of the simulator, and is used for all commands. The simulator does not have ADB support, and therefore we need to fall back to using standard shell commands.

Files are copied into the simulation using a VirtIO 9P device in gem5. Files are copied out of the simulated environment using the m5 writefile command within the simulated system.

When starting the workload run, the simulator is automatically started by Workload Automation, and a connection to the simulator is established. WA will then wait for Android to boot on the simulated system (which can take hours), prior to executing any other commands on the device. It is also possible to resume from a checkpoint when starting the simulation. To do this, please append the relevant checkpoint commands from the gem5 simulation script to the gem5_discription argument in the agenda.

Host system requirements:

- VirtIO support. We rely on diod on the host system. This can be installed on ubuntu using the following command:

  sudo apt-get install diod

Guest requirements:

- VirtIO support. We rely on VirtIO to move files into the simulation. Please make sure that the following are set in the kernel configuration:

  CONFIG_NET_9P=y

  CONFIG_NET_9P_VIRTIO=y

  CONFIG_9P_FS=y

  CONFIG_9P_FS_POSIX_ACL=y

  CONFIG_9P_FS_SECURITY=y

  CONFIG_VIRTIO_BLK=y

- m5 binary. Please make sure that the m5 binary is on the device and can by found in the path.

parameters

gem5_binary [str] Command used to execute gem5. Adjust according to needs.

default: `'./build/ARM/gem5.fast'`

gem5_args [arguments (mandatory)] Command line passed to the gem5 simulation. This command line is used to set up the simulated system, and should be the same as used for a standard gem5 simulation without workload automation. Note that this is simulation script specific and will hence need to be tailored to each particular use case.

gem5_vio_args [arguments (mandatory)] gem5 VirtIO command line used to enable the VirtIO device in the simulated system. At the very least, the root parameter of the VirtIO9PDiod device must be exposed on the command line. Please set this root mount to {}, as it will be replaced with the directory used by Workload Automation at runtime.

constraint: `"{}" in str(value)`

temp_dir [str] Temporary directory used to pass files into the gem5 simulation. Workload Automation will automatically create a directory in this folder, and will remove it again once the simulation completes.

default: `'/tmp'`

checkpoint [boolean] This parameter tells Workload Automation to create a checkpoint of the simulated system once the guest system has finished booting. This checkpoint can then be used at a later stage by other WA runs to avoid booting the guest system a second time. Set to True to take a checkpoint of the simulated system post boot.

run_delay [integer] This sets the time that the system should sleep in the simulated system prior to running and workloads or taking checkpoints. This allows the system to quieten down prior to running the workloads. When this is combined with the checkpoint_post_boot option, it allows the checkpoint to be created post-sleep, and therefore the set of workloads resuming from this checkpoint will not be required to sleep.

constraint: `value >= 0`

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: ['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']

binaries_directory [str] Location of binaries on the device.

default: '/data/local/tmp/wa-bin'

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: '/sdcard/wa-working'

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: r'^.*(shell|root)@.*:/\S* [#$] '

package_data_directory [str] Location of of data for an installed package (APK).

default: '/data/data'

external_storage_directory [str] Mount point for external storage.

default: '/sdcard'

connection [str] Specified the nature of adb connection.

allowed values: 'usb', 'ethernet'

default: 'usb'

logcat_poll_period [integer] If specified and is not 0, logcat will be polled every logcat_poll_period seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: None, 'horizontal', 'vertical'

3.4.8 gem5_linux

Implements gem5 Linux device.

This class allows a user to connect WA to a simulation using gem5. The connection to the device is made using the telnet connection of the simulator, and is used for all commands. The simulator does not have ADB support, and therefore we need to fall back to using standard shell commands.

Files are copied into the simulation using a VirtIO 9P device in gem5. Files are copied out of the simulated environment using the m5 writefile command within the simulated system.

When starting the workload run, the simulator is automatically started by Workload Automation, and a connection to the simulator is established. WA will then wait for Android to boot on the simulated system (which can take hours), prior to executing any other commands on the device. It is also possible to resume from a checkpoint when starting the simulation. To do this, please append the relevant checkpoint commands from the gem5 simulation script to the gem5_discription argument in the agenda.

Host system requirements:

- VirtIO support. We rely on diod on the host system. This can be installed on ubuntu using the following command:

  sudo apt-get install diod

Guest requirements:

- VirtIO support. We rely on VirtIO to move files into the simulation. Please make sure that the following are set in the kernel configuration:

  CONFIG_NET_9P=y

  CONFIG_NET_9P_VIRTIO=y

  CONFIG_9P_FS=y

  CONFIG_9P_FS_POSIX_ACL=y

  CONFIG_9P_FS_SECURITY=y

  CONFIG_VIRTIO_BLK=y

- m5 binary. Please make sure that the m5 binary is on the device and can by found in the path.

parameters

gem5_binary [str] Command used to execute gem5. Adjust according to needs.

default: `'./build/ARM/gem5.fast'`

gem5_args [arguments (mandatory)] Command line passed to the gem5 simulation. This command line is used to set up the simulated system, and should be the same as used for a standard gem5 simulation without workload automation. Note that this is simulation script specific and will hence need to be tailored to each particular use case.

gem5_vio_args [arguments (mandatory)] gem5 VirtIO command line used to enable the VirtIO device in the simulated system. At the very least, the root parameter of the VirtIO9PDiod device must be exposed on the command line. Please set this root mount to {}, as it will be replaced with the directory used by Workload Automation at runtime.

constraint: `"{}" in str(value)`

temp_dir [str] Temporary directory used to pass files into the gem5 simulation. Workload Automation will automatically create a directory in this folder, and will remove it again once the simulation completes.

default: `'/tmp'`

checkpoint [boolean] This parameter tells Workload Automation to create a checkpoint of the simulated system once the guest system has finished booting. This checkpoint can then be used at a later stage by other WA runs to avoid booting the guest system a second time. Set to True to take a checkpoint of the simulated system post boot.

run_delay [integer] This sets the time that the system should sleep in the simulated system prior to running and workloads or taking checkpoints. This allows the system to quieten down prior to running the workloads. When this is combined with the checkpoint_post_boot option, it allows the checkpoint to be created post-sleep, and therefore the set of workloads resuming from this checkpoint will not be required to sleep.

constraint: `value >= 0`

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of executable binaries on this device (must be in PATH).

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

host [str (mandatory)] Host name or IP address for the device.

default: `'localhost'`

username [str (mandatory)] User name for the account on the device.

password [str] Password for the account on the device (for password-based auth).

keyfile [str] Keyfile to be used for key-based authentication.

port [integer] SSH port number on the device.

default: `22`

password_prompt [str] Prompt presented by sudo when requesting the password.

default: `'[sudo] password'`

use_telnet [boolean] Optionally, telnet may be used instead of ssh, though this is discouraged.

boot_timeout [integer] How long to try to connect to the device after a reboot.

default: `120`

login_prompt : list_of_strs

default: `['login:', 'AEL login:', 'username:']`

login_password_prompt : list_of_strs

default: `['password:']`

3.4.9 generic_android

A generic Android device interface. Use this if you do not have an interface for your device.

This should allow basic WA functionality on most Android devices using adb over USB. Some additional configuration may be required for some WA extensions (e.g. configuring `core_names` and `core_clusters`).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

> **Note:** most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

allowed values: 'unknown', 'smp', 'hmp', 'iks', 'ea', 'other'

default: 'unknown'

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. scheduler != 'iks'). If left unset for IKS devices, it will default to 800000, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: ['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']

binaries_directory [str] Location of binaries on the device.

default: '/data/local/tmp/wa-bin'

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: '/sdcard/wa-working'

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: r'^.*(shell|root)@.*:/\S* [#$] '

package_data_directory [str] Location of of data for an installed package (APK).

default: '/data/data'

external_storage_directory [str] Mount point for external storage.

default: '/sdcard'

connection [str] Specified the nature of adb connection.

allowed values: 'usb', 'ethernet'

default: 'usb'

logcat_poll_period [integer] If specified and is not 0, logcat will be polled every logcat_poll_period seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: None, 'horizontal', 'vertical'

3.4.10 generic_linux

A generic Linux device interface. Use this if you do not have an interface for your device.

This should allow basic WA functionality on most Linux devices with SSH access configured. Some additional configuration may be required for some WA extensions (e.g. configuring core_names and core_clusters).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

> **unknown** A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.
>
> **smp** A standard single-core or Symmetric Multi-Processing system.
>
> **hmp** ARM Heterogeneous Multi-Processing system.
>
> **iks** Linaro In-Kernel Switcher.
>
> **ea** ARM Energy-Aware scheduler.
>
> **other** Any other system not covered by the above.
>
> ---
>
> **Note:** most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.
>
> ---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of executable binaries on this device (must be in PATH).

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

host [str (mandatory)] Host name or IP address for the device.

username [str (mandatory)] User name for the account on the device.

password [str] Password for the account on the device (for password-based auth).

keyfile [str] Keyfile to be used for key-based authentication.

port [integer] SSH port number on the device.

default: `22`

password_prompt [str] Prompt presented by sudo when requesting the password.

default: `'[sudo] password'`

use_telnet [boolean] Optionally, telnet may be used instead of ssh, though this is discouraged.

boot_timeout [integer] How long to try to connect to the device after a reboot.

default: `120`

3.4.11 juno

ARM Juno next generation big.LITTLE development platform.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['a53', 'a53', 'a53', 'a53', 'a57', 'a57']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0, 1, 1]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'hmp'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/sdcard/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

retries [integer] Specifies the number of times the device will attempt to recover (normally, with a hard reset) if it detects that something went wrong.

default: `2`

microsd_mount_point [str] Location at which the device's MicroSD card will be mounted.

default: `'/media/JUNO'`

port [str] Serial port on which the device is connected.

default: `'/dev/ttyS0'`

baudrate [integer] Serial connection baud.

default: `115200`

timeout [integer] Serial connection timeout.

default: `300`

bootloader [str] Bootloader used on the device.

allowed values: `'uefi'`, `'u-boot'`

default: `'uefi'`

actually_disconnect [boolean] Actually perfom "adb disconnect" on closing the connection to the device.

uefi_entry [str] The name of the entry to use (will be created if does not exist).

default: `'WA'`

uefi_config [UefiConfig] Specifies the configuration for the UEFI entry for his device. In an entry specified by `uefi_entry` parameter doesn't exist in UEFI menu, it will be created using this config. This configuration will also be used, when flashing new images.

default: `{'fdt_support': True, 'image_name': 'Image', 'image_args': None}`

bootargs [str] Default boot arguments to use when boot_arguments were not.

3.4.12 meizumx6

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/sdcard/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

3.4.13 odroidxu3

HardKernel Odroid XU3 development board.

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['a7', 'a7', 'a7', 'a7', 'a15', 'a15', 'a15', 'a15']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0, 1, 1, 1, 1]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of binaries on the device.

default: `'/data/local/tmp/wa-bin'`

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

default: `'/data/local/wa-working'`

adb_name [str] The unique ID of the device as output by "adb devices".

default: `'BABABEEFBABABEEF'`

android_prompt [regex] The format of matching the shell prompt in Android.

default: `r'^.*(shell|root)@.*:/\S* [#$] '`

package_data_directory [str] Location of of data for an installed package (APK).

default: `'/data/data'`

external_storage_directory [str] Mount point for external storage.

default: `'/sdcard'`

connection [str] Specified the nature of adb connection.

allowed values: `'usb'`, `'ethernet'`

default: `'usb'`

logcat_poll_period [integer] If specified and is not `0`, logcat will be polled every `logcat_poll_period` seconds, and buffered on the host. This can be used if a lot of output is expected in logcat and the fixed logcat buffer on the device is not big enough. The trade off is that this introduces some minor runtime overhead. Not set by default.

enable_screen_check [boolean] Specified whether the device should make sure that the screen is on during initialization.

swipe_to_unlock [str] If set a swipe of the specified direction will be performed. This should unlock the screen.

allowed values: `None`, `'horizontal'`, `'vertical'`

port [str] Serial port on which the device is connected

default: `'/dev/ttyUSB0'`

baudrate [integer] Serial connection baud rate

default: `115200`

3.4.14 odroidxu3_linux

HardKernel Odroid XU3 development board (Ubuntu image).

parameters

modules [list] Lists the modules to be loaded by this extension. A module is a plug-in that further extends functionality of an extension.

core_names [list_of_caseless_strings (mandatory)] This is a list of all cpu cores on the device with each element being the core type, e.g. `['a7', 'a7', 'a15']`. The order of the cores must match the order they are listed in `'/sys/devices/system/cpu'`. So in this case, `'cpu0'` must be an A7 core, and `'cpu2'` an A15.'

default: `['a7', 'a7', 'a7', 'a7', 'a15', 'a15', 'a15', 'a15']`

core_clusters [list_of_ints (mandatory)] This is a list indicating the cluster affinity of the CPU cores, each element correponding to the cluster ID of the core coresponding to its index. E.g. `[0, 0, 1]` indicates that cpu0 and cpu1 are on cluster 0, while cpu2 is on cluster 1. If this is not specified, this will be inferred from `core_names` if possible (assuming all cores with the same name are on the same cluster).

default: `[0, 0, 0, 0, 1, 1, 1, 1]`

scheduler [str] Specifies the type of multi-core scheduling model utilized in the device. The value must be one of the following:

unknown A generic Device interface is used to interact with the underlying device and the underlying scheduling model is unkown.

smp A standard single-core or Symmetric Multi-Processing system.

hmp ARM Heterogeneous Multi-Processing system.

iks Linaro In-Kernel Switcher.

ea ARM Energy-Aware scheduler.

other Any other system not covered by the above.

Note: most currently-available systems would fall under `smp` rather than this value. `other` is there to future-proof against new schemes not yet covered by WA.

---

allowed values: `'unknown'`, `'smp'`, `'hmp'`, `'iks'`, `'ea'`, `'other'`

default: `'unknown'`

iks_switch_frequency [integer] This is the switching frequency, in kilohertz, of IKS devices. This parameter *MUST NOT* be set for non-IKS device (i.e. `scheduler != 'iks'`). If left unset for IKS devices, it will default to `800000`, i.e. 800MHz.

property_files [list_of_strs] A list of paths to files containing static OS properties. These will be pulled into the __meta directory in output for each run in order to provide information about the platfrom. These paths do not have to exist and will be ignored if the path is not present on a particular device.

default: `['/etc/arch-release', '/etc/debian_version', '/etc/lsb-release', '/proc/config.gz', '/proc/cmdline', '/proc/cpuinfo', '/proc/version', '/proc/zconfig', '/sys/kernel/debug/sched_features', '/sys/kernel/hmp']`

binaries_directory [str] Location of executable binaries on this device (must be in PATH).

working_directory [str] Working directory to be used by WA. This must be in a location where the specified user has write permissions. This will default to /home/<username>/wa (or to /root/wa, if username is 'root').

host [str (mandatory)] Host name or IP address for the device.

username [str (mandatory)] User name for the account on the device.

password [str] Password for the account on the device (for password-based auth).

keyfile [str] Keyfile to be used for key-based authentication.

port [integer] SSH port number on the device.

default: `22`

password_prompt [str] Prompt presented by sudo when requesting the password.

default: `'[sudo] password'`

use_telnet [boolean] Optionally, telnet may be used instead of ssh, though this is discouraged.

boot_timeout [integer] How long to try to connect to the device after a reboot.

default: `120`

CHAPTER 4

In-depth

This section contains more advanced topics, such how to write your own extensions and detailed descriptions of how WA functions under the hood.

4.1 Conventions

4.1.1 Interface Definitions

Throughout this documentation a number of stubbed-out class definitions will be presented showing an interface defined by a base class that needs to be implemented by the deriving classes. The following conventions will be used when presenting such an interface:

- Methods shown raising `NotImplementedError` are abstract and *must* be overridden by subclasses.
- Methods with `pass` in their body *may* be (but do not need to be) overridden by subclasses. If not overridden, these methods will default to the base class implementation, which may or may not be a no-op (the `pass` in the interface specification does not necessarily mean that the method does not have an actual implementation in the base class).

  Note: If you *do* override these methods you must remember to call the base class' version inside your implementation as well.

- Attributes who's value is shown as `None` *must* be redefined by the subclasses with an appropriate value.
- Attributes who's value is shown as something other than `None` (including empty strings/lists/dicts) *may* be (but do not need to be) overridden by subclasses. If not overridden, they will default to the value shown.

Keep in mind that the above convention applies only when showing interface definitions and may not apply elsewhere in the documentation. Also, in the interest of clarity, only the relevant parts of the base class definitions will be shown some members (such as internal methods) may be omitted.

4.1.2 Code Snippets

Code snippets provided are intended to be valid Python code, and to be complete. However, for the sake of clarity, in some cases only the relevant parts will be shown with some details omitted (details that may necessary to validity of the code but not to understanding of the concept being illustrated). In such cases, a commented ellipsis will be used to indicate that parts of the code have been dropped. E.g.

```
# ...

def update_result(self, context):
    # ...
    context.result.add_metric('energy', 23.6, 'Joules', lower_is_better=True)

# ...
```

4.1.3 Core Class Names

When core classes are referenced throughout the documentation, usually their fully-qualified names are given e.g. *wlauto.core.workload.Workload*. This is done so that Sphinx can resolve them and provide a link. While implementing extensions, however, you should *not* be importing anything directly form under *wlauto.core*. Instead, classes you are meant to instantiate or subclass have been aliased in the root *wlauto* package, and should be imported from there, e.g.

```
from wlauto import Workload
```

All examples given in the documentation follow this convention. Please note that this only applies to the *wlauto.core* subpackage; all other classes should be imported for their corresponding subpackages.

4.2 Writing Extensions

Workload Automation offers several extension points (or plugin types).The most interesting of these are

workloads These are the tasks that get executed and measured on the device. These can be benchmarks, high-level use cases, or pretty much anything else.

devices These are interfaces to the physical devices (development boards or end-user devices, such as smartphones) that use cases run on. Typically each model of a physical device would require its own interface class (though some functionality may be reused by subclassing from an existing base).

instruments Instruments allow collecting additional data from workload execution (e.g. system traces). Instruments are not specific to a particular Workload. Instruments can hook into any stage of workload execution.

result processors These are used to format the results of workload execution once they have been collected. Depending on the callback used, these will run either after each iteration or at the end of the run, after all of the results have been collected.

You create an extension by subclassing the appropriate base class, defining appropriate methods and attributes, and putting the .py file with the class into an appropriate subdirectory under `~/.workload_automation` (there is one for each extension type).

4.2.1 Extension Basics

This sub-section covers things common to implementing extensions of all types. It is recommended you familiarize yourself with the information here before proceeding onto guidance for specific extension types.

To create an extension, you basically subclass an appropriate base class and them implement the appropriate methods

The Context

The majority of methods in extensions accept a context argument. This is an instance of *wlauto.core.execution.ExecutionContext*. If contains of information about current state of execution of WA and keeps track of things like which workload is currently running and the current iteration.

Notable attributes of the context are

context.spec the current workload specification being executed. This is an instance of *wlauto.core.configuration.WorkloadRunSpec* and defines the workload and the parameters under which it is being executed.

context.workload `Workload` object that is currently being executed.

context.current_iteration The current iteration of the spec that is being executed. Note that this is the iteration for that spec, i.e. the number of times that spec has been run, *not* the total number of all iterations have been executed so far.

context.result This is the result object for the current iteration. This is an instance of *wlauto.core.result.IterationResult*. It contains the status of the iteration as well as the metrics and artifacts generated by the workload and enable instrumentation.

context.device The device interface object that can be used to interact with the device. Note that workloads and instruments have their own device attribute and they should be using that instead.

In addition to these, context also defines a few useful paths (see below).

Paths

You should avoid using hard-coded absolute paths in your extensions whenever possible, as they make your code too dependent on a particular environment and may mean having to make adjustments when moving to new (host and/or device) platforms. To help avoid hard-coded absolute paths, WA automation defines a number of standard locations. You should strive to define your paths relative to one of those.

On the host

Host paths are available through the context object, which is passed to most extension methods.

context.run_output_directory This is the top-level output directory for all WA results (by default, this will be "wa_output" in the directory in which WA was invoked.

context.output_directory This is the output directory for the current iteration. This will an iteration-specific subdirectory under the main results location. If there is no current iteration (e.g. when processing overall run results) this will point to the same location as `root_output_directory`.

context.host_working_directory This an addition location that may be used by extensions to store non-iteration specific intermediate files (e.g. configuration).

Additionally, the global `wlauto.settings` object exposes on other location:

settings.dependency_directory this is the root directory for all extension dependencies (e.g. media files, assets etc) that are not included within the extension itself.

As per Python best practice, it is recommended that methods and values in `os.path` standard library module are used for host path manipulation.

On the device

Workloads and instruments have a `device` attribute, which is an interface to the device used by WA. It defines the following location:

device.working_directory This is the directory for all WA-related files on the device. All files deployed to the device should be pushed to somewhere under this location (the only exception being executables installed with `device.install` method).

Since there could be a mismatch between path notation used by the host and the device, the `os.path` modules should *not* be used for on-device path manipulation. Instead device has an equipment module exposed through `device.path` attribute. This has all the same attributes and behaves the same way as `os.path`, but is guaranteed to produce valid paths for the device, irrespective of the host's path notation. For example:

```
result_file = self.device.path.join(self.device.working_directory, "result.txt")
self.command = "{} -a -b -c {}".format(target_binary, result_file)
```

Note: result processors, unlike workloads and instruments, do not have their own device attribute; however they can access the device through the context.

Deploying executables to a device

Some devices may have certain restrictions on where executable binaries may be placed and how they should be invoked. To ensure your extension works with as wide a range of devices as possible, you should use WA APIs for deploying and invoking executables on a device, as outlined below.

As with other resources (see *Dynamic Resource Resolution*) , host-side paths to the exectuable binary to be deployed should be obtained via the resource resolver. A special resource type, `Executable` is used to identify a binary to be deployed. This is simiar to the regular `File` resource, however it takes an additional parameter that specifies the ABI for which executable was compiled.

In order for the binary to be obtained in this way, it must be stored in one of the locations scanned by the resource resolver in a directry structure `<root>/bin/<abi>/<binary>` (where `root` is the base resource location to be searched, e.g. `~/.workload_automation/dependencies/<extension name>`, and `<abi>` is the ABI for which the exectuable has been compiled, as returned by `self.device.abi`).

Once the path to the host-side binary has been obtained, it may be deployed using one of two methods of a `Device` instace – `install` or `install_if_needed`. The latter will check a version of that binary has been perviously deployed by WA and will not try to re-install.

```
from wlauto import Executable

host_binary = context.resolver.get(Executable(self, self.device.abi, 'some_binary'))
target_binary = self.device.install_if_needed(host_binary)
```

Note: Please also note that the check is done based solely on the binary name. For more information please see: `wlauto.common.linux.BaseLinuxDevice.install_if_needed()`

Both of the above methods will return the path to the installed binary on the device. The executable should be invoked *only* via that path; do **not** assume that it will be in `PATH` on the target (or that the executable with the same name in `PATH` is the version deployed by WA.

```
self.command = "{} -a -b -c".format(target_binary)
self.device.execute(self.command)
```

Parameters

All extensions can be parameterized. Parameters are specified using `parameters` class attribute. This should be a list of `wlauto.core.Parameter` instances. The following attributes can be specified on parameter creation:

name This is the only mandatory argument. The name will be used to create a corresponding attribute in the extension instance, so it must be a valid Python identifier.

kind This is the type of the value of the parameter. This could be an callable. Normally this should be a standard Python type, e.g. `int` or `float`, or one the types defined in *wlauto.utils.types*. If not explicitly specified, this will default to `str`.

Note: Irrespective of the `kind` specified, `None` is always a valid value for a parameter. If you don't want to allow `None`, then set `mandatory` (see below) to `True`.

allowed_values A list of the only allowed values for this parameter.

Note: For composite types, such as `list_of_strings` or `list_of_ints` in *wlauto.utils.types*, each element of the value will be checked against `allowed_values` rather than the composite value itself.

default The default value to be used for this parameter if one has not been specified by the user. Defaults to `None`.

mandatory A `bool` indicating whether this parameter is mandatory. Setting this to `True` will make `None` an illegal value for the parameter. Defaults to `False`.

Note: Specifying a `default` will mean that this parameter will, effectively, be ignored (unless the user sets the param to `None`).

Note: Mandatory parameters are *bad*. If at all possible, you should strive to provide a sensible `default` or to make do without the parameter. Only when the param is absolutely necessary, and there really is no sensible default that could be given (e.g. something like login credentials), should you consider making it mandatory.

constraint This is an additional constraint to be enforced on the parameter beyond its type or fixed allowed values set. This should be a predicate (a function that takes a single argument – the user-supplied value – and returns a `bool` indicating whether the constraint has been satisfied).

override A parameter name must be unique not only within an extension but also with that extension's class hierarchy. If you try to declare a parameter with the same name as already exists, you will get an error. If you do want to

override a parameter from further up in the inheritance hierarchy, you can indicate that by setting `override` attribute to `True`.

When overriding, you do not need to specify every other attribute of the parameter, just the ones you what to override. Values for the rest will be taken from the parameter in the base class.

Validation and cross-parameter constraints

An extension will get validated at some point after constructions. When exactly this occurs depends on the extension type, but it *will* be validated before it is used.

You can implement `validate` method in your extension (that takes no arguments beyond the `self`) to perform any additions *internal* validation in your extension. By "internal", I mean that you cannot make assumptions about the surrounding environment (e.g. that the device has been initialized).

The contract for `validate` method is that it should raise an exception (either `wlauto.exceptions.ConfigError` or extension-specific exception type – see further on this page) if some validation condition has not, and cannot, been met. If the method returns without raising an exception, then the extension is in a valid internal state.

Note that `validate` can be used not only to verify, but also to impose a valid internal state. In particular, this where cross-parameter constraints can be resolved. If the `default` or `allowed_values` of one parameter depend on another parameter, there is no way to express that declaratively when specifying the parameters. In that case the dependent attribute should be left unspecified on creation and should instead be set inside `validate`.

Logging

Every extension class has it's own logger that you can access through `self.logger` inside the extension's methods. Generally, a `Device` will log everything it is doing, so you shouldn't need to add much additional logging in your expansion's. But you might what to log additional information, e.g. what settings your extension is using, what it is doing on the host, etc. Operations on the host will not normally be logged, so your extension should definitely log what it is doing on the host. One situation in particular where you should add logging is before doing something that might take a significant amount of time, such as downloading a file.

Documenting

All extensions and their parameter should be documented. For extensions themselves, this is done through `description` class attribute. The convention for an extension description is that the first paragraph should be a short summary description of what the extension does and why one would want to use it (among other things, this will get extracted and used by `wa list` command). Subsequent paragraphs (separated by blank lines) can then provide a more detailed description, including any limitations and setup instructions.

For parameters, the description is passed as an argument on creation. Please note that if `default`, `allowed_values`, or `constraint`, are set in the parameter, they do not need to be explicitly mentioned in the description (wa documentation utilities will automatically pull those). If the `default` is set in `validate` or additional cross-parameter constraints exist, this *should* be documented in the parameter description.

Both extensions and their parameters should be documented using reStructureText markup (standard markup for Python documentation). See:

http://docutils.sourceforge.net/rst.html

Aside from that, it is up to you how you document your extension. You should try to provide enough information so that someone unfamiliar with your extension is able to use it, e.g. you should document all settings and parameters your extension expects (including what the valid value are).

Error Notification

When you detect an error condition, you should raise an appropriate exception to notify the user. The exception would typically be `ConfigError` or (depending the type of the extension) `WorkloadError`/`DeviceError`/`InstrumentError`/`ResultProcessorError`. All these errors are defined in `wlauto.exception` module.

`ConfigError` should be raised where there is a problem in configuration specified by the user (either through the agenda or config files). These errors are meant to be resolvable by simple adjustments to the configuration (and the error message should suggest what adjustments need to be made. For all other errors, such as missing dependencies, mis-configured environment, problems performing operations, etc., the extension type-specific exceptions should be used.

If the extension itself is capable of recovering from the error and carrying on, it may make more sense to log an ERROR or WARNING level message using the extension's logger and to continue operation.

Utils

Workload Automation defines a number of utilities collected under *`wlauto.utils`* subpackage. These utilities were created to help with the implementation of the framework itself, but may be also be useful when implementing extensions.

4.2.2 Adding a Workload

Note: You can use `wa create workload [name]` script to generate a new workload structure for you. This script can also create the boilerplate for UI automation, if your workload needs it. See `wa create -h` for more details.

New workloads can be added by subclassing *`wlauto.core.workload.Workload`*

The Workload class defines the following interface:

```
class Workload(Extension):

    name = None

    def init_resources(self, context):
        pass

    def validate(self):
        pass

    def initialize(self, context):
        pass

    def setup(self, context):
        pass

    def setup(self, context):
        pass

    def run(self, context):
        pass
```

```
    def update_result(self, context):
        pass

    def teardown(self, context):
        pass

    def finalize(self, context):
        pass
```

Note: Please see *Conventions* section for notes on how to interpret this.

The interface should be implemented as follows

name This identifies the workload (e.g. it used to specify it in the agenda.

init_resources This method may be optionally override to implement dynamic resource discovery for the workload. This method executes early on, before the device has been initialized, so it should only be used to initialize resources that do not depend on the device to resolve. This method is executed once per run for each workload instance.

validate This method can be used to validate any assumptions your workload makes about the environment (e.g. that required files are present, environment variables are set, etc) and should raise a *wlauto.exceptions.WorkloadError* if that is not the case. The base class implementation only makes sure sure that the name attribute has been set.

initialize This method will be executed exactly once per run (no matter how many instances of the workload there are). It will run after the device has been initialized, so it may be used to perform device-dependent initialization that does not need to be repeated on each iteration (e.g. as installing executables required by the workload on the device).

setup Everything that needs to be in place for workload execution should be done in this method. This includes copying files to the device, starting up an application, configuring communications channels, etc.

run This method should perform the actual task that is being measured. When this method exits, the task is assumed to be complete.

> **Note:** Instrumentation is kicked off just before calling this method and is disabled right after, so everything in this method is being measured. Therefore this method should contain the least code possible to perform the operations you are interested in measuring. Specifically, things like installing or starting applications, processing results, or copying files to/from the device should be done elsewhere if possible.

update_result This method gets invoked after the task execution has finished and should be used to extract metrics and add them to the result (see below).

teardown This could be used to perform any cleanup you may wish to do, e.g. Uninstalling applications, deleting file on the device, etc.

finalize This is the complement to `initialize`. This will be executed exactly once at the end of the run. This should be used to perform any final clean up (e.g. uninstalling binaries installed in the `initialize`).

Workload methods (except for `validate`) take a single argument that is a *wlauto.core.execution.ExecutionContext* instance. This object keeps track of the current execution state (such as the current work-

load, iteration number, etc), and contains, among other things, a `wlauto.core.workload.WorkloadResult` instance that should be populated from the `update_result` method with the results of the execution.

```
# ...

def update_result(self, context):
   # ...
   context.result.add_metric('energy', 23.6, 'Joules', lower_is_better=True)

# ...
```

Example

This example shows a simple workload that times how long it takes to compress a file of a particular size on the device.

Note: This is intended as an example of how to implement the Workload interface. The methodology used to perform the actual measurement is not necessarily sound, and this Workload should not be used to collect real measurements.

```
import os
from wlauto import Workload, Parameter

class ZiptestWorkload(Workload):

    name = 'ziptest'
    description = '''
                  Times how long it takes to gzip a file of a particular size on a␣
↪device.

                  This workload was created for illustration purposes only. It should␣
↪not be
                  used to collect actual measurements.

                  '''

    parameters = [
            Parameter('file_size', kind=int, default=2000000,
                      description='Size of the file (in bytes) to be gzipped.')
    ]

    def setup(self, context):
            # Generate a file of the specified size containing random garbage.
            host_infile = os.path.join(context.output_directory, 'infile')
            command = 'openssl rand -base64 {} > {}'.format(self.file_size, host_
↪infile)
            os.system(command)
            # Set up on-device paths
            devpath = self.device.path  # os.path equivalent for the device
            self.device_infile = devpath.join(self.device.working_directory, 'infile')
            self.device_outfile = devpath.join(self.device.working_directory, 'outfile
↪')
            # Push the file to the device
            self.device.push_file(host_infile, self.device_infile)

    def run(self, context):
            self.device.execute('cd {} && (time gzip {}) &>> {}'.format(self.device.
↪working_directory,
```

```
                                                                    self.device_
↪infile,
                                                                    self.device_
↪outfile))

    def update_result(self, context):
            # Pull the results file to the host
            host_outfile = os.path.join(context.output_directory, 'outfile')
            self.device.pull_file(self.device_outfile, host_outfile)
            # Extract metrics form the file's contents and update the result
            # with them.
            content = iter(open(host_outfile).read().strip().split())
            for value, metric in zip(content, content):
            mins, secs = map(float, value[:-1].split('m'))
            context.result.add_metric(metric, secs + 60 * mins)

    def teardown(self, context):
            # Clean up on-device file.
            self.device.delete_file(self.device_infile)
            self.device.delete_file(self.device_outfile)
```

Adding revent-dependent Workload:

`wlauto.common.game.GameWorkload` is the base class for all the workloads that depend on *revent* files. It implements all the methods needed to push the files to the device and run them. New GameWorkload can be added by subclassing `wlauto.common.game.GameWorkload`:

The GameWorkload class defines the following interface:

```
class GameWorkload(Workload):

    name = None
    package = None
    activity = None
```

The interface should be implemented as follows

name This identifies the workload (e.g. it used to specify it in the agenda.

package This is the name of the '.apk' package without its file extension.

activity The name of the main activity that runs the package.

Example:

This example shows a simple GameWorkload that plays a game.

```
from wlauto.common.game import GameWorkload

class MyGame(GameWorkload):

    name = 'mygame'
    package = 'com.mylogo.mygame'
    activity = 'myActivity.myGame'
```

Convention for Naming revent Files for `wlauto.common.game.GameWorkload`

There is a convention for naming revent files which you should follow if you want to record your own revent files. Each revent file must start with the device name(case sensitive) then followed by a dot '.' then the stage name then '.revent'. All your custom revent files should reside at '~/.workload_automation/dependencies/WORKLOAD NAME/'. These are the current supported stages:

setup This stage is where the game is loaded. It is a good place to record revent here to modify the game settings and get it ready to start.

run This stage is where the game actually starts. This will allow for more accurate results if the revent file for this stage only records the game being played.

For instance, to add a custom revent files for a device named mydevice and a workload name mygame, you create a new directory called mygame in '~/.workload_automation/dependencies/'. Then you add the revent files for the stages you want in ~/.workload_automation/dependencies/mygame/:

```
mydevice.setup.revent
mydevice.run.revent
```

Any revent file in the dependencies will always overwrite the revent file in the workload directory. So it is possible for example to just provide one revent for setup in the dependencies and use the run.revent that is in the workload directory.

4.2.3 Adding an Instrument

Instruments can be used to collect additional measurements during workload execution (e.g. collect power readings). An instrument can hook into almost any stage of workload execution. A typical instrument would implement a subset of the following interface:

```
class Instrument(Extension):

    name = None
    description = None

    parameters = [
    ]

    def initialize(self, context):
        pass

    def setup(self, context):
        pass

    def start(self, context):
        pass

    def stop(self, context):
        pass

    def update_result(self, context):
        pass

    def teardown(self, context):
        pass
```

```
    def finalize(self, context):
        pass
```

This is similar to a Workload, except all methods are optional. In addition to the workload-like methods, instruments can define a number of other methods that will get invoked at various points during run execution. The most useful of which is perhaps `initialize` that gets invoked after the device has been initialised for the first time, and can be used to perform one-time setup (e.g. copying files to the device – there is no point in doing that for each iteration). The full list of available methods can be found in *Signals Documentation*.

Prioritization

Callbacks (e.g. `setup()` methods) for all instrumentation get executed at the same point during workload execution, one after another. The order in which the callbacks get invoked should be considered arbitrary and should not be relied on (e.g. you cannot expect that just because instrument A is listed before instrument B in the config, instrument A's callbacks will run first).

In some cases (e.g. in `start()` and `stop()` methods), it is important to ensure that a particular instrument's callbacks run a closely as possible to the workload's invocations in order to maintain accuracy of readings; or, conversely, that a callback is executed after the others, because it takes a long time and may throw off the accuracy of other instrumentation. You can do this by prepending `fast_` or `slow_` to your callbacks' names. For example:

```
class PreciseInstrument(Instument):

    # ...

    def fast_start(self, context):
        pass

    def fast_stop(self, context):
        pass

    # ...
```

`PreciseInstrument` will be started after all other instrumentation (i.e. *just* before the workload runs), and it will stopped before all other instrumentation (i.e. *just* after the workload runs). It is also possible to use `very_fast_` and `very_slow_` prefixes when you want to be really sure that your callback will be the last/first to run.

If more than one active instrument have specified fast (or slow) callbacks, then their execution order with respect to each other is not guaranteed. In general, having a lot of instrumentation enabled is going to necessarily affect the readings. The best way to ensure accuracy of measurements is to minimize the number of active instruments (perhaps doing several identical runs with different instruments enabled).

Example

Below is a simple instrument that measures the execution time of a workload:

```
class ExecutionTimeInstrument(Instrument):
    """
    Measure how long it took to execute the run() methods of a Workload.

    """

    name = 'execution_time'

    def initialize(self, context):
```

```
        self.start_time = None
        self.end_time = None

    def fast_start(self, context):
        self.start_time = time.time()

    def fast_stop(self, context):
        self.end_time = time.time()

    def update_result(self, context):
        execution_time = self.end_time - self.start_time
        context.result.add_metric('execution_time', execution_time, 'seconds')
```

4.2.4 Adding a Result Processor

A result processor is responsible for processing the results. This may involve formatting and writing them to a file, uploading them to a database, generating plots, etc. WA comes with a few result processors that output results in a few common formats (such as csv or JSON).

You can add your own result processors by creating a Python file in `~/.workload_automation/result_processors` with a class that derives from *wlauto.core.result.ResultProcessor*, which has the following interface:

```
class ResultProcessor(Extension):

    name = None
    description = None

    parameters = [
    ]

    def initialize(self, context):
            pass

    def process_iteration_result(self, result, context):
            pass

    def export_iteration_result(self, result, context):
            pass

    def process_run_result(self, result, context):
            pass

    def export_run_result(self, result, context):
            pass

    def finalize(self, context):
            pass
```

The method names should be fairly self-explanatory. The difference between "process" and "export" methods is that export methods will be invoke after process methods for all result processors have been generated. Process methods may generated additional artifacts (metrics, files, etc), while export methods should not – the should only handle existing results (upload them to a database, archive on a filer, etc).

The result object passed to iteration methods is an instance of *wlauto.core.result.IterationResult*, the result object passed to run methods is an instance of *wlauto.core.result.RunResult*. Please refer to their

API documentation for details.

Example

Here is an example result processor that formats the results as a column-aligned table:

```
import os
from wlauto import ResultProcessor
from wlauto.utils.misc import write_table


class Table(ResultProcessor):

    name = 'table'
    description = 'Gerates a text file containing a column-aligned table with run␣
↪results.'

    def process_run_result(self, result, context):
        rows = []
        for iteration_result in result.iteration_results:
            for metric in iteration_result.metrics:
                rows.append([metric.name, str(metric.value), metric.units or '',
                             metric.lower_is_better  and '-' or '+'])

        outfile =  os.path.join(context.output_directory, 'table.txt')
        with open(outfile, 'w') as wfh:
            write_table(rows, wfh)
```

4.2.5 Adding a Resource Getter

A resource getter is a new extension type added in version 2.1.3. A resource getter implement a method of acquiring resources of a particular type (such as APK files or additional workload assets). Resource getters are invoked in priority order until one returns the desired resource.

If you want WA to look for resources somewhere it doesn't by default (e.g. you have a repository of APK files), you can implement a getter for the resource and register it with a higher priority than the standard WA getters, so that it gets invoked first.

Instances of a resource getter should implement the following interface:

```
class ResourceGetter(Extension):

    name = None
    resource_type = None
    priority = GetterPriority.environment

    def get(self, resource, **kwargs):
        raise NotImplementedError()
```

The getter should define a name (as with all extensions), a resource type, which should be a string, e.g. `'jar'`, and a priority (see *Getter Prioritization* below). In addition, `get` method should be implemented. The first argument is an instance of *`wlauto.core.resource.Resource`* representing the resource that should be obtained. Additional keyword arguments may be used by the invoker to provide additional information about the resource. This method should return an instance of the resource that has been discovered (what "instance" means depends on the resource, e.g. it could be a file path), or `None` if this getter was unable to discover that resource.

Getter Prioritization

A priority is an integer with higher numeric values indicating a higher priority. The following standard priority aliases are defined for getters:

cached The cached version of the resource. Look here first. This priority also implies that the resource at this location is a "cache" and is not the only version of the resource, so it may be cleared without losing access to the resource.

preferred Take this resource in favour of the environment resource.

environment Found somewhere under ~/.workload_automation/ or equivalent, or from environment variables, external configuration files, etc. These will override resource supplied with the package.

package Resource provided with the package.

remote Resource will be downloaded from a remote location (such as an HTTP server or a samba share). Try this only if no other getter was successful.

These priorities are defined as class members of *wlauto.core.resource.GetterPriority*, e.g. `GetterPriority.cached`.

Most getters in WA will be registered with either `environment` or `package` priorities. So if you want your getter to override the default, it should typically be registered as `preferred`.

You don't have to stick to standard priority levels (though you should, unless there is a good reason). Any integer is a valid priority. The standard priorities range from -20 to 20 in increments of 10.

Example

The following is an implementation of a getter for a workload APK file that looks for the file under `~/.workload_automation/dependencies/<workload_name>`:

```
import os
import glob

from wlauto import ResourceGetter, GetterPriority, settings
from wlauto.exceptions import ResourceError


class EnvironmentApkGetter(ResourceGetter):

    name =  'environment_apk'
    resource_type = 'apk'
    priority = GetterPriority.environment

    def get(self, resource):
        resource_dir = _d(os.path.join(settings.dependency_directory, resource.owner.
↪name))
        version = kwargs.get('version')
        found_files = glob.glob(os.path.join(resource_dir, '*.apk'))
        if version:
            found_files = [ff for ff in found_files if version.lower() in ff.lower()]
        if len(found_files) == 1:
            return found_files[0]
        elif not found_files:
            return None
        else:
```

```
            raise ResourceError('More than one .apk found in {} for {}.'.
↪format(resource_dir,
                                                                        ␣
↪resource.owner.name))
```

4.2.6 Adding a Device

At the moment, only Android devices are supported. Most of the functionality for interacting with a device is implemented in `wlauto.common.AndroidDevice` and is exposed through `generic_android` device interface, which should suffice for most purposes. The most common area where custom functionality may need to be implemented is during device initialization. Usually, once the device gets to the Android home screen, it's just like any other Android device (modulo things like differences between Android versions).

If your device doesn't not work with `generic_device` interface and you need to write a custom interface to handle it, you would do that by subclassing `AndroidDevice` and then just overriding the methods you need. Typically you will want to override one or more of the following:

reset Trigger a device reboot. The default implementation just sends `adb reboot` to the device. If this command does not work, an alternative implementation may need to be provided.

hard_reset This is a harsher reset that involves cutting the power to a device (e.g. holding down power button or removing battery from a phone). The default implementation is a no-op that just sets some internal flags. If you're dealing with unreliable prototype hardware that can crash and become unresponsive, you may want to implement this in order for WA to be able to recover automatically.

connect When this method returns, adb connection to the device has been established. This gets invoked after a reset. The default implementation just waits for the device to appear in the adb list of connected devices. If this is not enough (e.g. your device is connected via Ethernet and requires an explicit `adb connect` call), you may wish to override this to perform the necessary actions before invoking the `AndroidDevices` version.

init This gets called once at the beginning of the run once the connection to the device has been established. There is no default implementation. It's there to allow whatever custom initialisation may need to be performed for the device (setting properties, configuring services, etc).

Please refer to the API documentation for `wlauto.common.AndroidDevice` for the full list of its methods and their functionality.

4.2.7 Other Extension Types

In addition to extension types covered above, there are few other, more specialized ones. They will not be covered in as much detail. Most of them expose relatively simple interfaces with only a couple of methods and it is expected that if the need arises to extend them, the API-level documentation that accompanies them, in addition to what has been outlined here, should provide enough guidance.

commands This allows extending WA with additional sub-commands (to supplement exiting ones outlined in the *invocation* section).

modules Modules are "extensions for extensions". They can be loaded by other extensions to expand their functionality (for example, a flashing module maybe loaded by a device in order to support flashing).

4.2.8 Packaging Your Extensions

If your have written a bunch of extensions, and you want to make it easy to deploy them to new systems and/or to update them on existing systems, you can wrap them in a Python package. You can use `wa create package`

command to generate appropriate boiler plate. This will create a `setup.py` and a directory for your package that you can place your extensions into.

For example, if you have a workload inside `my_workload.py` and a result processor in `my_result_processor.py`, and you want to package them as `my_wa_exts` package, first run the create command

```
wa create package my_wa_exts
```

This will create a `my_wa_exts` directory which contains a `my_wa_exts/setup.py` and a subdirectory `my_wa_exts/my_wa_exts` which is the package directory for your extensions (you can rename the top-level `my_wa_exts` directory to anything you like – it's just a "container" for the setup.py and the package directory). Once you have that, you can then copy your extensions into the package directory, creating `my_wa_exts/my_wa_exts/my_workload.py` and `my_wa_exts/my_wa_exts/my_result_processor.py`. If you have a lot of extensions, you might want to organize them into subpackages, but only the top-level package directory is created by default, and it is OK to have everything in there.

Note: When discovering extensions thorugh this mechanism, WA traveries the Python module/submodule tree, not the directory strucuter, therefore, if you are going to create subdirectories under the top level dictory created for you, it is important that your make sure they are valid Python packages; i.e. each subdirectory must contain a __init__.py (even if blank) in order for the code in that directory and its subdirectories to be discoverable.

At this stage, you may want to edit `params` structure near the bottom of the `setup.py` to add correct author, license and contact information (see "Writing the Setup Script" section in standard Python documentation for details). You may also want to add a README and/or a COPYING file at the same level as the setup.py. Once you have the contents of your package sorted, you can generate the package by running

```
cd my_wa_exts
python setup.py sdist
```

This will generate `my_wa_exts/dist/my_wa_exts-0.0.1.tar.gz` package which can then be deployed on the target system with standard Python package management tools, e.g.

```
sudo pip  install my_wa_exts-0.0.1.tar.gz
```

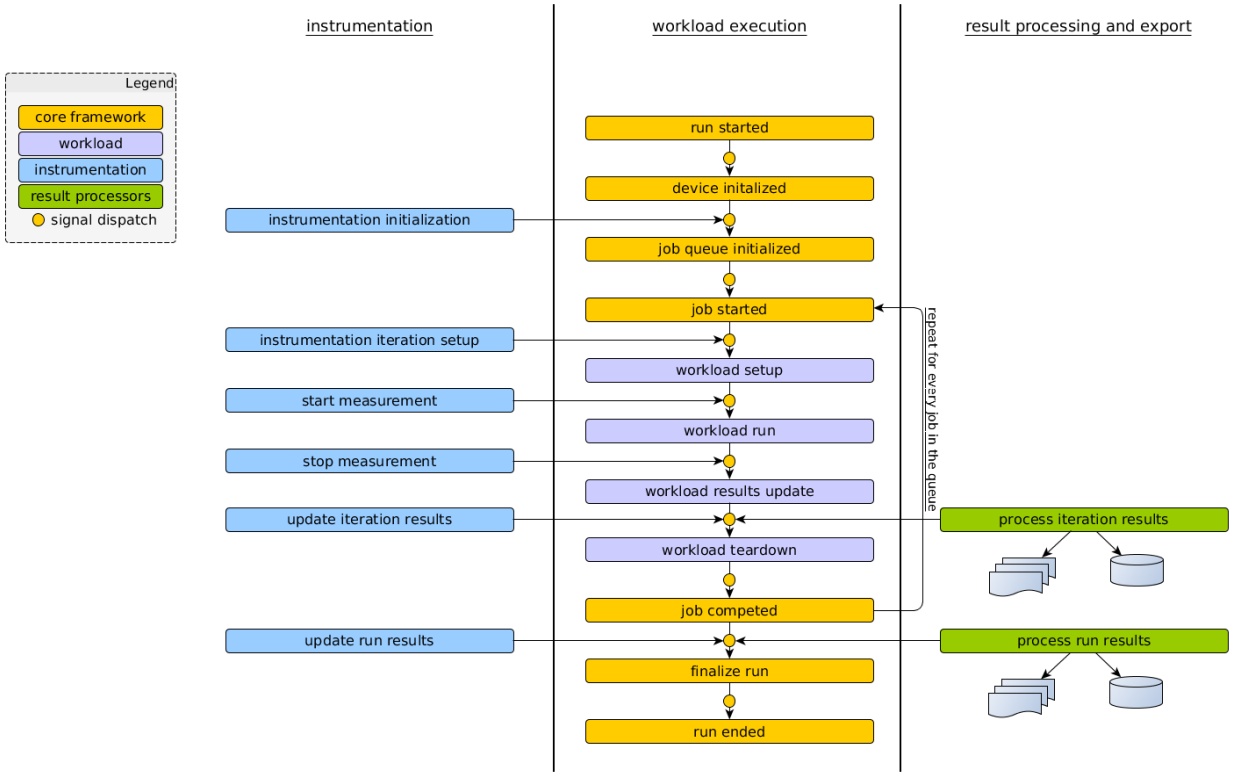
As part of the installation process, the setup.py in the package, will write the package's name into `~/.workoad_automoation/packages`. This will tell WA that the package contains extension and it will load them next time it runs.

Note: There are no unistall hooks in `setuputils`, so if you ever uninstall your WA extensions package, you will have to manually remove it from `~/.workload_automation/packages` otherwise WA will complain abou a missing package next time you try to run it.

4.3 Framework Overview

4.3.1 Execution Model

At the high level, the execution model looks as follows:



After some initial setup, the framework initializes the device, loads and initialized instrumentation and begins executing jobs defined by the workload specs in the agenda. Each job executes in four basic stages:

setup Initial setup for the workload is performed. E.g. required assets are deployed to the devices, required services or applications are launched, etc. Run time configuration of the device for the workload is also performed at this time.

run This is when the workload actually runs. This is defined as the part of the workload that is to be measured. Exactly what happens at this stage depends entirely on the workload.

result processing Results generated during the execution of the workload, if there are any, are collected, parsed and extracted metrics are passed up to the core framework.

teardown Final clean up is performed, e.g. applications may closed, files generated during execution deleted, etc.

Signals are dispatched (see *signal_dispatch* below) at each stage of workload execution, which installed instrumentation can hook into in order to collect measurements, alter workload execution, etc. Instrumentation implementation usually mirrors that of workloads, defining setup, teardown and result processing stages for a particular instrument. Instead of a `run`, instruments usually implement a `start` and a `stop` which get triggered just before and just after a workload run. However, the signal dispatch mechanism give a high degree of flexibility to instruments allowing them to hook into almost any stage of a WA run (apart from the very early initialization).

Metrics and artifacts generated by workloads and instrumentation are accumulated by the framework and are then passed to active result processors. This happens after each individual workload execution and at the end of the run. A result process may chose to act at either or both of these points.

4.3.2 Control Flow

This section goes into more detail explaining the relationship between the major components of the framework and how control passes between them during a run. It will only go through the major transition and interactions and will not attempt to describe very single thing that happens.

Note: This is the control flow for the `wa run` command which is the main functionality of WA. Other commands are much simpler and most of what is described below does not apply to them.

1. `wlauto.core.entry_point` parses the command form the arguments and executes the run command (`wlauto.commands.run.RunCommand`).
2. Run command initializes the output directory and creates a `wlauto.core.agenda.Agenda` based on the command line arguments. Finally, it instantiates a `wlauto.core.execution.Executor` and passes it the Agenda.
3. The Executor uses the Agenda to create a `wlauto.core.configuraiton.RunConfiguration` fully defines the configuration for the run (it will be serialised into `__meta` subdirectory under the output directory.
4. The Executor proceeds to instantiate and install instrumentation, result processors and the device interface, based on the RunConfiguration. The executor also initialise a `wlauto.core.execution.ExecutionContext` which is used to track the current state of the run execution and also serves as a means of communication between the core framework and the extensions.
5. Finally, the Executor instantiates a `wlauto.core.execution.Runner`, initializes its job queue with workload specs from the RunConfiguraiton, and kicks it off.
6. The Runner performs the run time initialization of the device and goes through the workload specs (in the order defined by `execution_order` setting), running each spec according to the execution model described in the previous section. The Runner sends signals (see below) at appropriate points during execution.
7. At the end of the run, the control is briefly passed back to the Executor, which outputs a summary for the run.

4.3.3 Signal Dispatch

WA uses the louie (formerly, pydispatcher) library for signal dispatch. Callbacks can be registered for signals emitted during the run. WA uses a version of louie that has been modified to introduce priority to registered callbacks (so that callbacks that are know to be slow can be registered with a lower priority so that they do not interfere with other callbacks).

This mechanism is abstracted for instrumentation. Methods of an `wlauto.core.Instrument` subclass automatically get hooked to appropriate signals based on their names when the instrument is "installed" for the run. Priority can be specified by adding `very_fast_`, `fast_` , `slow_` or `very_slow_` prefixes to method names.

The full list of method names and the signals they map to may be viewed *here*.

Signal dispatching mechanism may also be used directly, for example to dynamically register callbacks at runtime or allow extensions other than `Instruments` to access stages of the run they are normally not aware of.

The sending of signals is the responsibility of the Runner. Signals gets sent during transitions between execution stages and when special evens, such as errors or device reboots, occur.

See Also

Instrumentation Signal-Method Mapping

Instrument methods get automatically hooked up to signals based on their names. Mostly, the method name correponds to the name of the signal, however there are a few convienience aliases defined (listed first) to make easier to relate instrumenation code to the workload execution model.

method name	signal
initialize	run-init-signal
setup	successful-workload-setup-signal
start	before-workload-execution-signal
stop	after-workload-execution-signal
process_workload_result	successful-iteration-result-update-signal
update_result	after-iteration-result-update-signal
teardown	after-workload-teardown-signal
finalize	run-fin-signal
on_run_start	start-signal
on_run_end	end-signal
on_workload_spec_start	workload-spec-start-signal
on_workload_spec_end	workload-spec-end-signal
on_iteration_start	iteration-start-signal
on_iteration_end	iteration-end-signal
before_initial_boot	before-initial-boot-signal
on_successful_initial_boot	successful-initial-boot-signal
after_initial_boot	after-initial-boot-signal
before_first_iteration_boot	before-first-iteration-boot-signal
on_successful_first_iteration_boot	successful-first-iteration-boot-signal
after_first_iteration_boot	after-first-iteration-boot-signal
before_boot	before-boot-signal
on_successful_boot	successful-boot-signal
after_boot	after-boot-signal
on_spec_init	spec-init-signal
on_run_init	run-init-signal
on_iteration_init	iteration-init-signal
before_workload_setup	before-workload-setup-signal
on_successful_workload_setup	successful-workload-setup-signal
after_workload_setup	after-workload-setup-signal
before_workload_execution	before-workload-execution-signal
on_successful_workload_execution	successful-workload-execution-signal
after_workload_execution	after-workload-execution-signal
before_workload_result_update	before-iteration-result-update-signal
on_successful_workload_result_update	successful-iteration-result-update-signal
after_workload_result_update	after-iteration-result-update-signal
before_workload_teardown	before-workload-teardown-signal
on_successful_workload_teardown	successful-workload-teardown-signal
after_workload_teardown	after-workload-teardown-signal
before_overall_results_processing	before-overall-results-process-signal
on_successful_overall_results_processing	successful-overall-results-process-signal
after_overall_results_processing	after-overall-results-process-signal
on_error	error_logged
on_warning	warning_logged

The names above may be prefixed with one of pre-defined prefixes to set the priority of the Instrument method realive to other callbacks registered for the signal (within the same priority level, callbacks are invoked in the order they were registered). The table below shows the mapping of the prifix to the corresponding priority:

prefix	priority
very_fast_	20
fast_	10
normal_	0
slow_	-10
very_slow_	-20

4.4 Dynamic Resource Resolution

Introduced in version 2.1.3.

The idea is to decouple resource identification from resource discovery. Workloads/instruments/devices/etc state *what* resources they need, and not *where* to look for them – this instead is left to the resource resolver that is now part of the execution context. The actual discovery of resources is performed by resource getters that are registered with the resolver.

A resource type is defined by a subclass of *wlauto.core.resource.Resource*. An instance of this class describes a resource that is to be obtained. At minimum, a `Resource` instance has an owner (which is typically the object that is looking for the resource), but specific resource types may define other parameters that describe an instance of that resource (such as file names, URLs, etc).

An object looking for a resource invokes a resource resolver with an instance of `Resource` describing the resource it is after. The resolver goes through the getters registered for that resource type in priority order attempting to obtain the resource; once the resource is obtained, it is returned to the calling object. If none of the registered getters could find the resource, `None` is returned instead.

The most common kind of object looking for resources is a `Workload`, and since v2.1.3, `Workload` class defines *wlauto.core.workload.Workload.init_resources()* method that may be overridden by subclasses to perform resource resolution. For example, a workload looking for an APK file would do so like this:

```
from wlauto import Workload
from wlauto.common.resources import ApkFile

class AndroidBenchmark(Workload):

    # ...

    def init_resources(self, context):
            self.apk_file = context.resource.get(ApkFile(self))

    # ...
```

Currently available resource types are defined in *wlauto.common.resources*.

4.5 Additional Topics

4.5.1 Modules

Modules are essentially plug-ins for Extensions. They provide a way of defining common and reusable functionality. An Extension can load zero or more modules during its creation. Loaded modules will then add their capabilities (see *Capabilities*) to those of the Extension. When calling code tries to access an attribute of an Extension the Extension doesn't have, it will try to find the attribute among its loaded modules and will return that instead.

Note: Modules are themselves extensions, and can therefore load their own modules. *Do not* abuse this.

For example, calling code may wish to reboot an unresponsive device by calling `device.hard_reset()`, but the `Device` in question does not have a `hard_reset` method; however the `Device` has loaded `netio_switch` module which allows to disable power supply over a network (say this device is in a rack and is powered through such a switch). The module has `reset_power` capability (see *Capabilities* below) and so implements `hard_reset`. This will get invoked when `device.hard_rest()` is called.

Note: Modules can only extend Extensions with new attributes; they cannot override existing functionality. In the example above, if the `Device` has implemented `hard_reset()` itself, then *that* will get invoked irrespective of which modules it has loaded.

If two loaded modules have the same capability or implement the same method, then the last module to be loaded "wins" and its method will be invoke, effectively overriding the module that was loaded previously.

Specifying Modules

Modules get loaded when an Extension is instantiated by the extension loader. There are two ways to specify which modules should be loaded for a device.

4.5.2 Capabilities

Capabilities define the functionality that is implemented by an Extension, either within the Extension itself or through loadable modules. A capability is just a label, but there is an implied contract. When an Extension claims to have a particular capability, it promises to expose a particular set of functionality through a predefined interface.

Currently used capabilities are described below.

Note: Since capabilities are basically random strings, the user can always define their own; and it is then up to the user to define, enforce and document the contract associated with their capability. Below, are the "standard" capabilities used in WA.

Note: The method signatures in the descriptions below show the calling signature (i.e. they're omitting the initial self parameter).

active_cooling

Intended to be used by devices and device modules, this capability implies that the device implements a controllable active cooling solution (e.g. a programmable fan). The device/module must implement the following methods:

start_active_cooling() Active cooling is started (e.g. the fan is turned on)

stop_active_cooling() Active cooling is stopped (e.g. the fan is turned off)

reset_power

Intended to be used by devices and device modules, this capability implies that the device is capable of performing a hard reset by toggling power. The device/module must implement the following method:

hard_reset() The device is restarted. This method cannot rely on the device being responsive and must work even if the software on the device has crashed.

flash

Intended to be used by devices and device modules, this capability implies that the device can be flashed with new images. The device/module must implement the following method:

flash(image_bundle=None, images=None) `image_bundle` is a path to a "bundle" (e.g. a tarball) that contains all the images to be flashed. Which images go where must also be defined within the bundle. `images` is a dict mapping image destination (e.g. partition name) to the path to that specific image. Both `image_bundle` and `images` may be specified at the same time. If there is overlap between the two, `images` wins and its contents will be flashed in preference to the `image_bundle`.

4.6 DAQ Server Guide

NI-DAQ, or just "DAQ", is the Data Acquisition device developed by National Instruments:

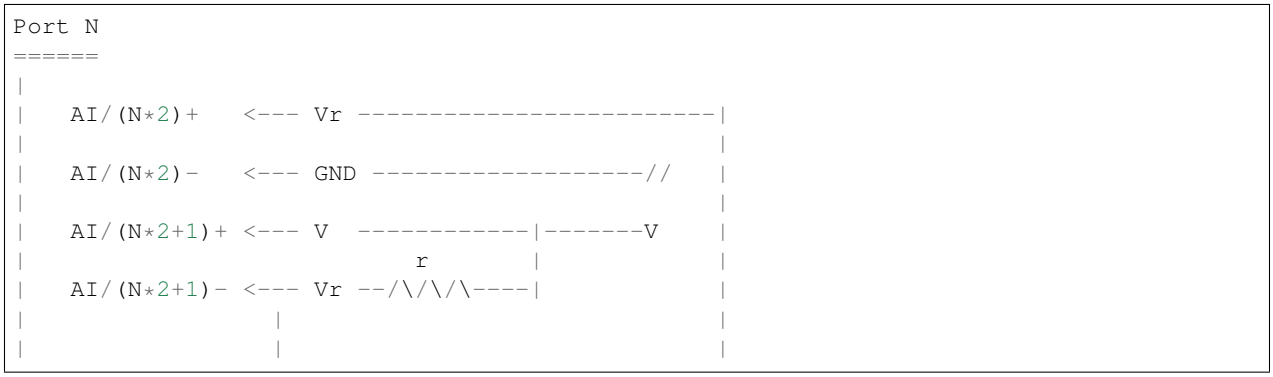
http://www.ni.com/data-acquisition/

WA uses the DAQ to collect power measurements during workload execution. A client/server solution for this is distributed as part of WA, though it is distinct from WA and may be used separately (by invoking the client APIs from a Python script, or used directly from the command line).

This solution is dependent on the NI-DAQmx driver for the DAQ device. At the time of writing, only Windows versions of the driver are supported (there is an old Linux version that works on some versions of RHEL and Centos, but it is unsupported and won't work with recent Linux kernels). Because of this, the server part of the solution will need to be run on a Windows machine (though it should also work on Linux, if the driver becomes available).

4.6.1 DAQ Device Wiring

The server expects the device to be wired in a specific way in order to be able to collect power measurements. Two consecutive Analogue Input (AI) channels on the DAQ are used to form a logical "port" (starting with AI/0 and AI/1 for port 0). Of these, the lower/even channel (e.g. AI/0) is used to measure the voltage on the rail we're interested in; the higher/odd channel (e.g. AI/1) is used to measure the voltage drop across a known very small resistor on the same rail, which is then used to calculate current. The logical wiring diagram looks like this:



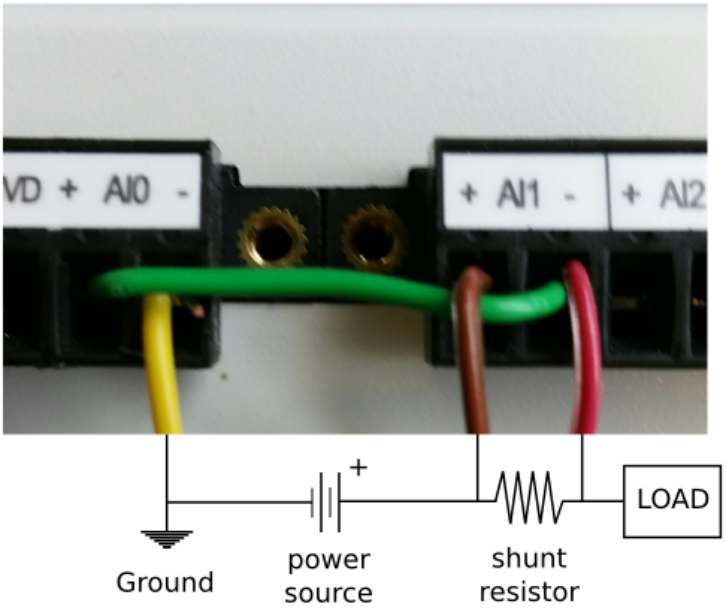
```
|                  |-----------------------------|
======

Where:
        V: Voltage going into the resistor
        Vr: Voltage between resistor and the SOC
        GND: Ground
        r: The resistor across the rail with a known
           small value.
```

The physical wiring will depend on the specific DAQ device, as channel layout varies between models.

Note: Current solution supports variable number of ports, however it assumes that the ports are sequential and start at zero. E.g. if you want to measure power on three rails, you will need to wire ports 0-2 (AI/0 to AI/5 channels on the DAQ) to do it. It is not currently possible to use any other configuration (e.g. ports 1, 2 and 5).

As an example, the following illustration shows the wiring of PORT0 (using AI/0 and AI/1 channels) on a DAQ USB-6210



4.6.2 Setting up NI-DAQmx driver on a Windows Machine

- The NI-DAQmx driver is pretty big in size, 1.5 GB. The driver name is 'NI-DAQmx' and its version '9.7.0f0' which you can obtain it from National Instruments website by downloading NI Measurement & Automation Explorer (Ni MAX) from: http://joule.ni.com/nidu/cds/view/p/id/3811/lang/en

  Note: During the installation process, you might be prompted to install .NET framework 4.

- The installation process is quite long, 7-15 minutes.
- Once installed, open NI MAX, which should be in your desktop, if not type its name in the start->search.

- Connect the NI-DAQ device to your machine. You should see it appear under 'Devices and Interfaces'. If not, press 'F5' to refresh the list.
- Complete the device wiring as described in the *DAQ Device Wiring* section.
- Quit NI MAX.

4.6.3 Setting up DAQ server

The DAQ power measurement solution is implemented in daqpower Python library, the package for which can be found in WA's install location under `wlauto/external/daq_server/daqpower-1.0.0.tar.gz` (the version number in your installation may be different).

- Install NI-DAQmx driver, as described in the previous section.
- Install Python 2.7.
- Download and install `pip`, `numpy` and `twisted` Python packages. These packages have C extensions, an so you will need a native compiler set up if you want to install them from PyPI. As an easier alternative, you can find pre-built Windows installers for these packages here (the versions are likely to be older than what's on PyPI though).
- Install the daqpower package using pip:

```
pip install C:\Python27\Lib\site-packages\wlauto\external\daq_server\daqpower-1.0.
↪0.tar.gz
```

  This should automatically download and install `PyDAQmx` package as well (the Python bindings for the NI-DAQmx driver).

4.6.4 Running DAQ server

Once you have installed the `daqpower` package and the required dependencies as described above, you can start the server by executing `run-daq-server` from the command line. The server will start listening on the default port, 45677.

Note: There is a chance that pip will not add `run-daq-server` into your path. In that case, you can run daq server as such: `python C:\path to python\Scripts\run-daq-server`

You can optionally specify flags to control the behaviour or the server:

```
usage: run-daq-server [-h] [-d DIR] [-p PORT] [--debug] [--verbose]

optional arguments:
-h, --help            show this help message and exit
-d DIR, --directory DIR
                        Working directory
-p PORT, --port PORT  port the server will listen on.
--debug               Run in debug mode (no DAQ connected).
--verbose             Produce verobose output.
```

Note: The server will use a working directory (by default, the directory the run-daq-server command was executed in, or the location specified with -d flag) to store power traces before they are collected by the client. This directory must be read/write-able by the user running the server.

4.6.5 Collecting Power with WA

Note: You do *not* need to install the `daqpower` package on the machine running WA, as it is already included in the WA install structure. However, you do need to make sure that `twisted` package is installed.

You can enable `daq` instrument your agenda/config.py in order to get WA to collect power measurements. At minimum, you will also need to specify the resistor values for each port in your configuration, e.g.:

```
resistor_values = [0.005, 0.005]  # in Ohms
```

This also specifies the number of logical ports (measurement sites) you want to use, and, implicitly, the port numbers (ports 0 to N-1 will be used).

Note: "ports" here refers to the logical ports wired on the DAQ (see *DAQ Device Wiring*, not to be confused with the TCP port the server is listening on.

Unless you're running the DAQ server and WA on the same machine (unlikely considering that WA is officially supported only on Linux and recent NI-DAQmx drivers are only available on Windows), you will also need to specify the IP address of the server:

```
daq_server =  127.0.0.1
```

There are a number of other settings that can optionally be specified in the configuration (e.g. the labels to be used for DAQ ports). Please refer to the *`wlauto.instrumentation.daq.Daq`* documentation for details.

4.6.6 Collecting Power from the Command Line

`daqpower` package also comes with a client that may be used from the command line. Unlike when collecting power with WA, you *will* need to install the `daqpower` package. Once installed, you will be able to interract with a running DAQ server by invoking `send-daq-command`. The invocation syntax is

```
send-daq-command --host HOST [--port PORT] COMMAND [OPTIONS]
```

Options are command-specific. COMMAND may be one of the following (and they should generally be inoked in that order):

configure Set up a new session, specifying the configuration values to be used. If there is already a configured session, it will be terminated. OPTIONS for this this command are the DAQ configuration parameters listed in the DAQ instrument documentation with all _ replaced by – and prefixed with `--`, e.g. `--resistor-values`.

start Start collecting power measurments.

stop Stop collecting power measurments.

get_data Pull files containg power measurements from the server. There is one option for this command: `--output-directory` which specifies where the files will be pulled to; if this is not specified, the will be in the current directory.

close Close the currently configured server session. This will get rid of the data files and configuration on the server, so it would no longer be possible to use "start" or "get_data" commands before a new session is configured.

A typical command line session would go like this:

```
send-daq-command --host 127.0.0.1 configure --resistor-values 0.005 0.005
# set up and kick off the use case you want to measure
send-daq-command --host 127.0.0.1 start
# wait for the use case to complete
send-daq-command --host 127.0.0.1 stop
send-daq-command --host 127.0.0.1 get_data
# files called PORT_0.csv and PORT_1.csv will appear in the current directory
# containing measurements collected during use case execution
send-daq-command --host 127.0.0.1 close
# the session is terminated and the csv files on the server have been
# deleted. A new session may now be configured.
```

In addtion to these "standard workflow" commands, the following commands are also available:

list_devices Returns a list of DAQ devices detected by the NI-DAQmx driver. In case mutiple devices are connected to the server host, you can specify the device you want to use with `--device-id` option when configuring a session.

list_ports Returns a list of ports tha have been configured for the current session, e.g. `['PORT_0', 'PORT_1']`.

list_port_files Returns a list of data files that have been geneted (unless something went wrong, there should be one for each port).

4.6.7 Collecting Power from another Python Script

You can invoke the above commands from a Python script using `daqpower.client.execute_command()` function, passing in `daqpower.config.ServerConfiguration` and, in case of the configure command, `daqpower.config.DeviceConfigruation`. Please see the implementation of the `daq` WA instrument for examples of how these APIs can be used.

4.7 revent

4.7.1 Overview and Usage

revent utility can be used to record and later play back a sequence of user input events, such as key presses and touch screen taps. This is an alternative to Android UI Automator for providing automation for workloads.

```
usage:
        revent [record time file|replay file|info] [verbose]
                record: stops after either return on stdin
                        or time (in seconds)
                        and stores in file
                replay: replays eventlog from file
                info:shows info about each event char device
                any additional parameters make it verbose
```

Recording

WA features a `record` command that will automatically deploy and start revent on the target device:

```
wa record
INFO     Connecting to device...
INFO     Press Enter when you are ready to record...
[Pressed Enter]
INFO     Press Enter when you have finished recording...
[Pressed Enter]
INFO     Pulling files from device
```

Once started, you will need to get the target device ready to record (e.g. unlock screen, navigate menus and launch an app) then press `ENTER`. The recording has now started and button presses, taps, etc you perform on the device will go into the .revent file. To stop the recording simply press `ENTER` again.

Once you have finished recording the revent file will be pulled from the device to the current directory. It will be named `{device_model}.revent`. When recording revent files for a `GameWorkload` you can use the `-s` option to add `run` or `setup` suffixes.

From version 2.6 of WA onwards, a "gamepad" recording mode is also supported. This mode requires a gamepad to be connected to the device when recoridng, but the recordings produced in this mode should be portable across devices.

For more information run please read *record*

Replaying

To replay a recorded file, run `wa replay`, giving it the file you want to replay:

```
wa replay my_recording.revent
```

For more information run please read *replay*

Using revent With Workloads

Some workloads (pretty much all games) rely on recorded revents for their execution. `wlauto.common.GameWorkload`-derived workloads expect two revent files – one for performing the initial setup (navigating menus, selecting game modes, etc), and one for the actual execution of the game. Because revents are very device-specific*[0], these two files would need to be recorded for each device.

The files must be called `<device name>.(setup|run).revent`, where `<device name>` is the name of your device (as defined by the `name` attribute of your device's class). WA will look for these files in two places: `<install dir>/wlauto/workloads/<workload name>/revent_files` and `~/.workload_automation/dependencies/<workload name>`. The first location is primarily intended for revent files that come with WA (and if you did a system-wide install, you'll need sudo to add files there), so it's probably easier to use the second location for the files you record. Also, if revent files for a workload exist in both locations, the files under `~/.workload_automation/dependencies` will be used in favor of those installed with WA.

For example, if you wanted to run angrybirds workload on "Acme" device, you would record the setup and run revent files using the method outlined in the section above and then pull them for the devices into the following locations:

```
~/workload_automation/dependencies/angrybirds/Acme.setup.revent
~/workload_automation/dependencies/angrybirds/Acme.run.revent
```

(you may need to create the intermediate directories if they don't already exist).

[0] It's not just about screen resolution – the event codes may be different even if devices use the same screen.

revent vs. UiAutomator

In general, Android UI Automator is the preferred way of automating user input for workloads because, unlike revent, UI Automator does not depend on a particular screen resolution, and so is more portable across different devices. It also gives better control and can potentially be faster for ling UI manipulations, as input events are scripted based on the available UI elements, rather than generated by human input.

On the other hand, revent can be used to manipulate pretty much any workload, where as UI Automator only works for Android UI elements (such as text boxes or radio buttons), which makes the latter useless for things like games. Recording revent sequence is also faster than writing automation code (on the other hand, one would need maintain a different revent log for each screen resolution).

4.7.2 Using state detection with revent

State detection can be used to verify that a workload is executing as expected. This utility, if enabled, and if state definitions are available for the particular workload, takes a screenshot after the setup and the run revent sequence, matches the screenshot to a state and compares with the expected state. A WorkloadError is raised if an unexpected state is encountered.

To enable state detection, make sure a valid state definition file and templates exist for your workload and set the check_states parameter to True.

State definition directory

State and phase definitions should be placed in a directory of the following structure inside the dependencies directory of each workload (along with revent files etc):

```
dependencies/
   <workload_name>/
      state_definitions/
         definition.yaml
         templates/
            <oneTemplate>.png
            <anotherTemplate>.png
            ...
```

definition.yaml file

This defines each state of the workload and lists which templates are expected to be found and how many are required to be detected for a conclusive match. It also defines the expected state in each workload phase where a state detection is run (currently those are setup_complete and run_complete).

Templates are picture elements to be matched in a screenshot. Each template mentioned in the definition file should be placed as a file with the same name and a .png extension inside the templates folder. Creating template png files is as simple as taking a screenshot of the workload in a given state, cropping out the relevant templates (eg. a button, label or other unique element that is present in that state) and storing them in PNG format.

Please see the definition file for Angry Birds below as an example to understand the format. Note that more than just two states (for the afterSetup and afterRun phase) can be defined and this helps track the cause of errors in case an unexpected state is encountered.

```
workload_name: angrybirds

workload_states:
```

```
  - state_name: titleScreen
    templates:
      - play_button
      - logo
    matches: 2
  - state_name: worldSelection
    templates:
      - first_world_thumb
      - second_world_thumb
      - third_world_thumb
      - fourth_world_thumb
    matches: 3
  - state_name: level_selection
    templates:
      - locked_level
      - first_level
    matches: 2
  - state_name: gameplay
    templates:
      - pause_button
      - score_label_text
    matches: 2
  - state_name: pause_screen
    templates:
      - replay_button
      - menu_button
      - resume_button
      - help_button
    matches: 4
  - state_name: level_cleared_screen
    templates:
      - level_cleared_text
      - menu_button
      - replay_button
      - fast_forward_button
    matches: 4

workload_phases:
  - phase_name: setup_complete
    expected_state: gameplay
  - phase_name: run_complete
    expected_state: level_cleared_screen
```

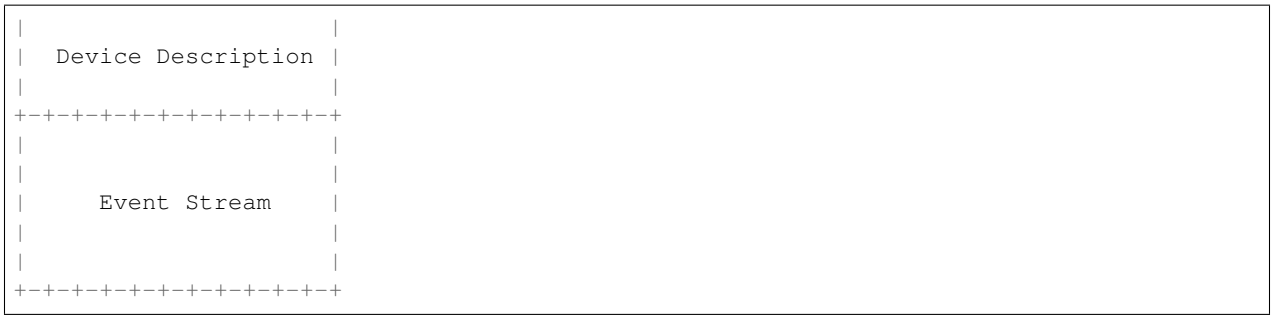
4.7.3 File format of revent recordings

You do not need to understand recording format in order to use revent. This section is intended for those looking to extend revent in some way, or to utilize revent recordings for other purposes.

Format Overview

Recordings are stored in a binary format. A recording consists of three sections:

```
+-+-+-+-+-+-+-+-+-+-+-+-+
|       Header          |
+-+-+-+-+-+-+-+-+-+-+-+-+
```



The header contains metadata describing the recording. The device description contains information about input devices involved in this recording. Finally, the event stream contains the recorded input events.

All fields are either fixed size or prefixed with their length or the number of (fixed-sized) elements.

Note: All values below are little endian

Recording Header

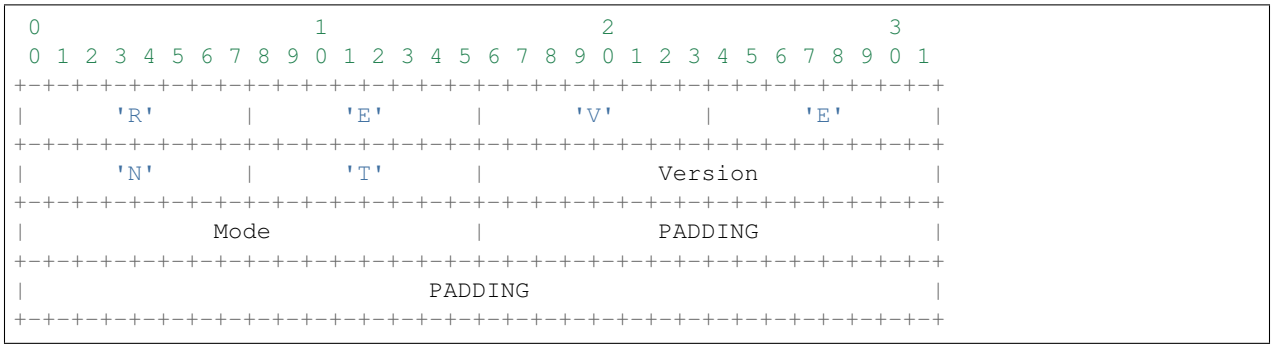
An revent recoding header has the following structure

- It starts with the "magic" string `REVENT` to indicate that this is an revent recording.
- The magic is followed by a 16 bit version number. This indicates the format version of the recording that follows. Current version is `2`.
- The next 16 bits indicate the type of the recording. This dictates the structure of the Device Description section. Valid values are:

  0 This is a general input event recording. The device description contains a list of paths from which the events where recorded.

  1 This a gamepad recording. The device description contains the description of the gamepad used to create the recording.

- The header is zero-padded to 128 bits.



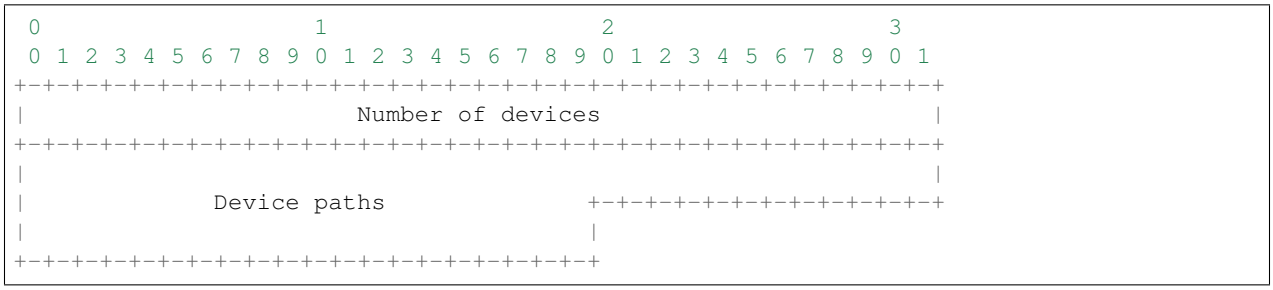
Device Description

This section describes the input devices used in the recording. Its structure is determined by the value of `Mode` field in the header.

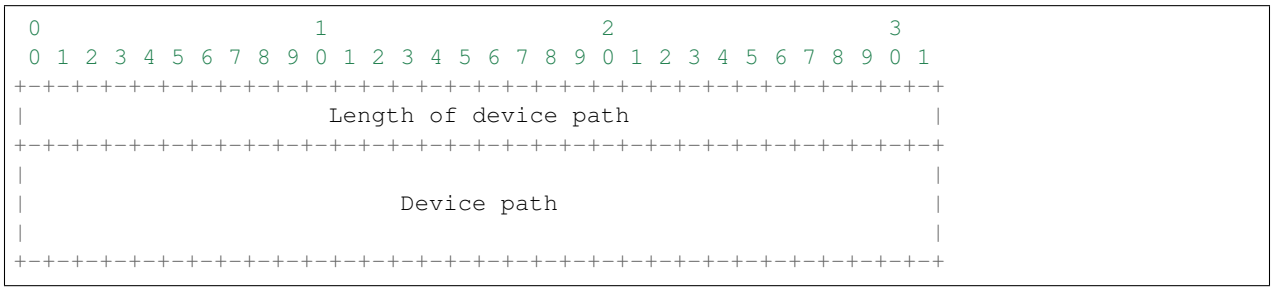
general recording

Note: This is the only format supported prior to version 2.

The recording has been made from all available input devices. This section contains the list of /dev/input paths for the devices, prefixed with total number of the devices recorded.

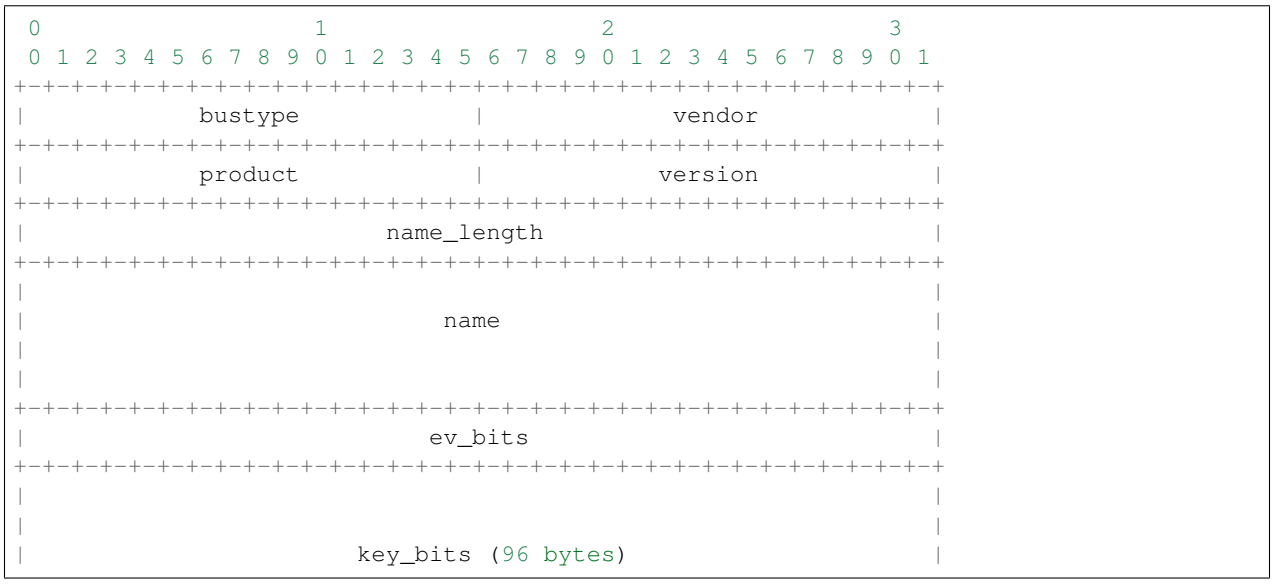


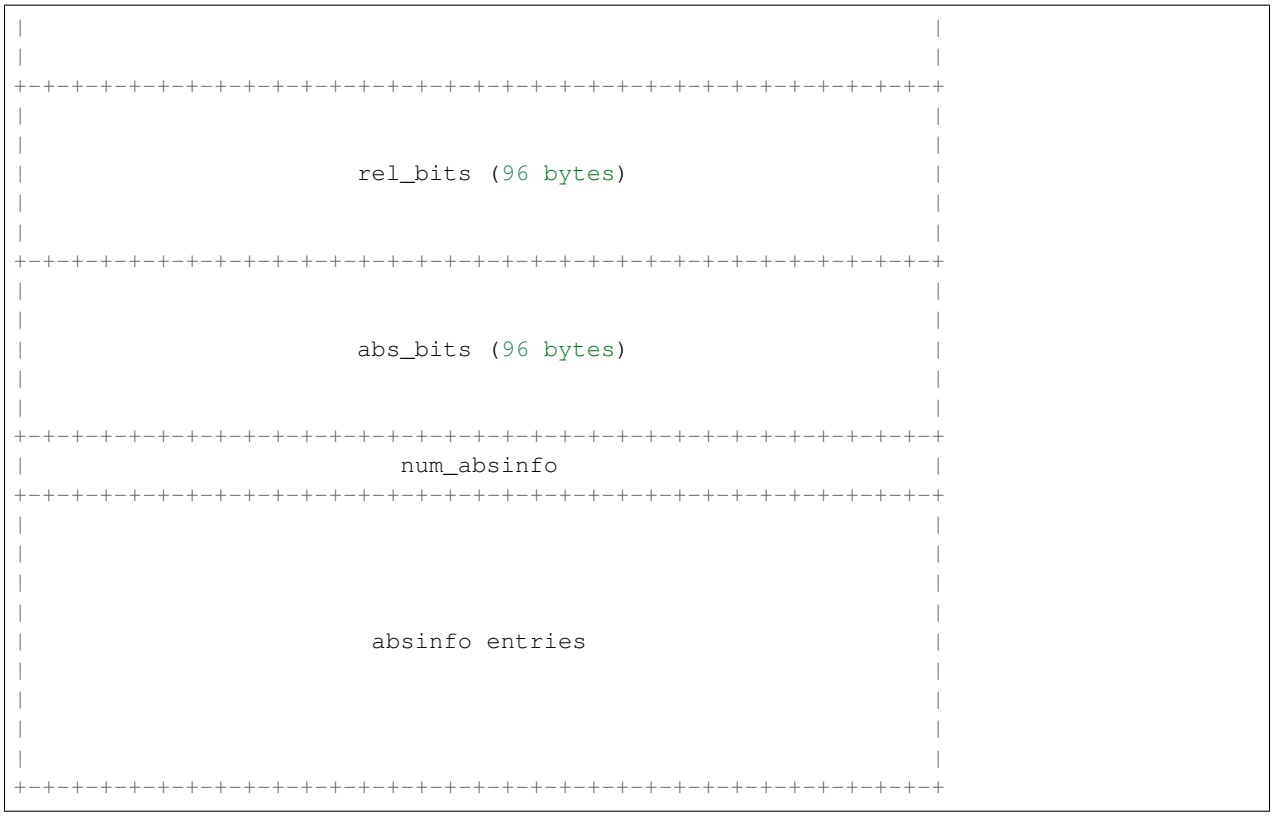
Similarly, each device path is a length-prefixed string. Unlike C strings, the path is *not* NULL-terminated.



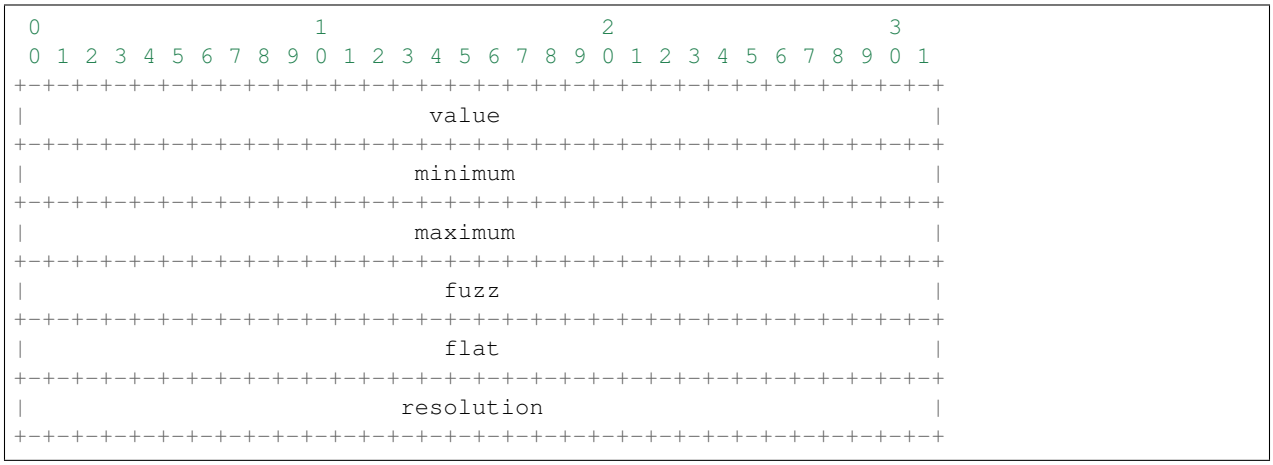
gamepad recording

The recording has been made from a specific gamepad. All events in the stream will be for that device only. The section describes the device properties that will be used to create a virtual input device using /dev/uinput. Please see linux/input.h header in the Linux kernel source for more information about the fields in this section.





Each `absinfo` entry consists of six 32 bit values. The number of entries is determined by the `abs_bits` field.



Event structure

The majority of an revent recording will be made up of the input events that were recorded. The event stream is prefixed with the number of events in the stream.

Each event entry structured as follows:

- An unsigned integer representing which device from the list of device paths this event is for (zero indexed). E.g. Device ID = 3 would be the 4th device in the list of device paths.
- A signed integer representing the number of seconds since "epoch" when the event was recorded.

- A signed integer representing the microseconds part of the timestamp.
- An unsigned integer representing the event type
- An unsigned integer representing the event code
- An unsigned integer representing the event value

For more information about the event type, code and value please read: https://www.kernel.org/doc/Documentation/input/event-codes.txt

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Device ID            |         Timestamp Seconds       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Timestamp Seconds (cont.)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Timestamp Seconds (cont.)     |         stamp Micoseconds       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Timestamp Micoseconds (cont.)                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Timestamp Micoseconds (cont.) |           Event Type            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Event Code            |           Event Value           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Event Value (cont.)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Parser

WA has a parser for revent recordings. This can be used to work with revent recordings in scripts. Here is an example:

```
from wlauto.utils.revent import ReventRecording

with ReventRecording('/path/to/recording.revent') as recording:
    print "Recording: {}".format(recording.filepath)
    print "There are {} input events".format(recording.num_events)
    print "Over a total of {} seconds".format(recording.duration)
```

4.8 APK Workloads

4.8.1 APK resolution

WA has various resource getters that can be configured to locate APK files but for most people APK files should be kept in the `$WA_USER_DIRECTORY/dependencies/SOME_WORKLOAD/` directory. (by default `~/.workload_automation/dependencies/SOME_WORKLOAD/`). The `WA_USER_DIRECTORY` enviroment variable can be used to chnage the location of this folder. The APK files need to be put into the corresponding directories for the workload they belong to. The name of the file can be anything but as explained below may need to contain certain peices of information.

All ApkWorkloads have parameters that affect the way in which APK files are resolved, `exact_abi`, `force_install` and `check_apk`. Their exact behaviours are outlined below.

`exact_abi`
: If this setting is enabled WA's resource resolvers will look for the devices ABI with any native code present in

the apk. By default this setting is disabled since most apks will work across all devices. You may wish to enable this feature when working with devices that support multiple ABI's (like 64-bit devices that can run 32-bit APK files) and are specifically trying to test one or the other.

force_install
: If this setting is enabled WA will *always* use the APK file on the host, and re-install it on every iteration. If there is no APK on the host that is a suitable version and/or ABI for the workload WA will error when `force_install` is enabled.

check_apk
: This parameter is used to specify a preference over host or target versions of the app. When set to `True` WA will prefer the host side version of the APK. It will check if the host has the APK and if the host APK meets the version requirements of the workload. If does and the target already has same version nothing will be done, other wise it will overwrite the targets app with the host version. If the hosts is missing the APK or it does not meet version requirements WA will fall back to the app on the target if it has the app and it is of a suitable version. When this parameter is set to `false` WA will prefer to use the version already on the target if it meets the workloads version requirements. If it does not it will fall back to search the host for the correct version. In both modes if neither the host nor target have a suitable version, WA will error and not run the workload.

Some workloads will also feature the follow parameters which will alter the way their APK files are resolved.

version
: This parameter is used to specify which version of uiautomation for the workload is used. In some workloads e.g. `geekbench` multiple versions with drastically different UI's are supported. When a workload uses a version it is required for the APK file to contain the uiautomation version in the file name. In the case of antutu the file names could be: `geekbench_2.apk` or `geekbench_3.apk`.

variant_name
: Some workloads use variants of APK files, this is usually the case with web browser APK files, these work in exactly the same way as the version, the variant of the apk

4.9 Contributing Code

We welcome code contributions via GitHub pull requests.To help with maintainability of the code line we ask that the code uses a coding style consistent with the rest of WA code. Briefly, it is

- PEP8 with line length and block comment rules relaxed (the wrapper for PEP8 checker inside `dev_scripts` will run it with appropriate configuration).
- Four-space indentation (*no tabs!*).
- Title-case for class names, underscore-delimited lower case for functions, methods, and variables.
- Use descriptive variable names. Delimit words with `'_'` for readability. Avoid shortening words, skipping vowels, etc (common abbreviations such as "stats" for "statistics", "config" for "configuration", etc are OK). Do *not* use Hungarian notation (so prefer `birth_date` over `dtBirth`).

New extensions should also follow implementation guidelines specified in *Writing Extensions* section of the documentation.

We ask that the following checks are performed on the modified code prior to submitting a pull request:

Note: You will need pylint and pep8 static checkers installed:

```
pip install pep8
pip install pylint
```

It is recommened that you install via pip rather than through your distribution's package mananger because the latter is likely to contain out-of-date version of these tools.

- `./dev_scripts/pylint` should be run without arguments and should produce no output (any output should be addressed by making appropriate changes in the code or adding a pylint ignore directive, if there is a good reason for keeping the code as is).
- `./dev_scripts/pep8` should be run without arguments and should produce no output (any output should be addressed by making appropriate changes in the code).
- If the modifications touch core framework (anything under `wlauto/core`), unit tests should be run using `nosetests`, and they should all pass.
  - If significant additions have been made to the framework, unit tests should be added to cover the new functionality.
- If modifications have been made to documentation (this includes description attributes for Parameters and Extensions), documentation should be built to make sure no errors or warning during build process, and a visual inspection of new/updated sections in resulting HTML should be performed to ensure everything renders as expected.

Once you have your contribution is ready, please follow instructions in GitHub documentation to create a pull request.

CHAPTER 5

API Reference

5.1 wlauto

5.1.1 wlauto package

Subpackages

wlauto.commands package

Submodules

wlauto.commands.create module

wlauto.commands.create.**create_workload**(*name*, *kind='basic'*, *where='local'*, *check_name=True*, ***kwargs*)

wlauto.commands.get_assets module

class wlauto.commands.get_assets.**GetAssetsCommand**(*subparsers*)

Bases: *wlauto.core.command.Command*

aliases = AC([])

artifacts = AC([])

assets_url = 'https://github.com/ARM-software/workload-automation-assets/raw/master/dep

core_modules = []

description = '\n This command downloads external extension dependencies used by Workl

execute(*args*)

exit_with_error(*message*, *code=1*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'get-assets'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.commands.get_assets.**NamedExtension**(*name*, ***kwargs*)

Bases: *wlauto.core.extension.Extension*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.commands.get_assets.**not_empty**(*val*)

wlauto.commands.list module

class wlauto.commands.list.**ListCommand**(*subparsers*)

Bases: *wlauto.core.command.Command*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'List available WA extensions with a short description of each.'

execute(*args*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'list'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.commands.list.**check_platform**(*extension*, *platform*)

wlauto.commands.record module

class wlauto.commands.record.**LightContext**(*config*)
Bases: object

class wlauto.commands.record.**RecordCommand**(*subparsers*)
Bases: *wlauto.commands.record.ReventCommand*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "Performs a revent recording\n\n This command helps create revent record

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'record'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*args*)

validate(**args*, ***kwargs*)

class wlauto.commands.record.**ReplayCommand**(*subparsers*)
Bases: *wlauto.commands.record.ReventCommand*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'Replay a revent recording\n\n Revent allows you to record raw inputs su

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'replay'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*args*)

validate(**args*, ***kwargs*)

class wlauto.commands.record.**ReventCommand**(*subparsers*)
Bases: *wlauto.core.command.Command*

aliases = AC([])

artifacts = AC([])

core_modules = []

execute(*args*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*args*)

validate(**args*, ***kwargs*)

validate_args(*args*)

wlauto.commands.run module

class wlauto.commands.run.**RunCommand**(*subparsers*)

Bases: *wlauto.core.command.Command*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'Execute automated workloads on a remote device and process the resultin

execute(*args*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'run'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

set_up_output_directory(*args*)

validate(**args*, ***kwargs*)

wlauto.commands.show module

class wlauto.commands.show.**ShowCommand**(*subparsers*)

Bases: *wlauto.core.command.Command*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Display documentation for the specified extension (workload, instrum

execute(*args*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'command'

name = 'show'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.commands.show.**format_extension**(*extension*, *out*, *width*)

wlauto.commands.show.**format_extension_description**(*extension*, *out*, *width*)

wlauto.commands.show.**format_extension_name**(*extension*, *out*)

wlauto.commands.show.**format_extension_parameters**(*extension*, *out*, *width*, *shift=4*)

wlauto.commands.show.**format_extension_summary**(*extension*, *out*, *width*)

wlauto.commands.show.**format_supported_platforms**(*extension*, *out*, *width*)

Module contents

wlauto.common package

Subpackages

wlauto.common.android package

Submodules

wlauto.common.android.device module

class wlauto.common.android.device.**AndroidDevice**(***kwargs*)

Bases: *wlauto.common.linux.device.BaseLinuxDevice*

Device running Android OS.

abi

aliases = AC([])

artifacts = AC([])

boot(*hard=False*, ***kwargs*)

broadcast_media_mounted(*dirpath*, *as_root=False*)

Force a re-index of the mediaserver cache for the specified directory.

broadcast_media_scan_file(*filepath*)

Force a re-index of the mediaserver cache for the specified file.

capture_screen(*filepath*)

Caputers the current device screen into the specified file in a PNG format.

capture_ui_hierarchy(*filepath*)

Captures the current view hierarchy into the specified file in a XML format.

clear_logcat()

Clear (flush) logcat log.

connect()

core_modules = []

default_timeout = 30

delay = 2

delete_file(*filepath*, *as_root=False*)

deploy_sqlite3(*context*)

disable_screen_lock()
Attempts to disable he screen lock on the device.

Note: This does not always work...

Added inversion 2.1.4

disable_selinux()

disconnect()

dump_logcat(*outfile*, *filter_spec=None*)
Dump the contents of logcat, for the specified filter spec to the specified output file. See http://developer.android.com/tools/help/logcat.html

Parameters

- **outfile** – Output file on the host into which the contents of the log will be written.
- **filter_spec** – Logcat filter specification. see http://developer.android.com/tools/debugging/debugging-log.html#filteringOutput

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

ensure_screen_is_on()

executable_is_installed(*executable_name*)

execute(*command*, *timeout=30*, *check_exit_code=True*, *background=False*, *as_root=False*, *busybox=False*, ***kwargs*)
Execute the specified command on the device using adb.

Parameters:

param command The command to be executed. It should appear exactly as if you were typing it into a shell.

param timeout Time, in seconds, to wait for adb to return before aborting and raising an error. Defaults to `AndroidDevice.default_timeout`.

param check_exit_code If `True`, the return code of the command on the Device will be check and exception will be raised if it is not 0. Defaults to `True`.

param background If `True`, will execute adb in a subprocess, and will return immediately, not waiting for adb to return. Defaults to `False`

param busybox If `True`, will use busybox to execute the command. Defaults to `False`.

Added in version 2.1.3

Note: The device must be rooted to be able to use some busybox features.

param as_root If `True`, will attempt to execute command in privileged mode. The device must be rooted, otherwise an error will be raised. Defaults to `False`.

Added in version 2.1.3

Returns If `background` parameter is set to `True`, the subprocess object will be returned; otherwise, the contents of STDOUT from the device will be returned.

Raises DeviceError if adb timed out or if the command returned non-zero exit code on the device, or if attempting to execute a command in privileged mode on an unrooted device.

file_exists(*filepath*)

finalize(**args*, ***kwargs*)

forward_port(*from_port*, *to_port*)
Forward a port on the device to a port on localhost.

Parameters

- **from_port** – Port on the device which to forward.
- **to_port** – Port on the localhost to which the device port will be forwarded.

Ports should be specified using adb spec. See the "adb forward" section in "adb help".

get_android_id()
Get the device's ANDROID_ID. Which is

"A 64-bit number (as a hex string) that is randomly generated when the user first sets up the device and should remain constant for the lifetime of the user's device."

Note: This will get reset on userdata erasure.

---

get_android_version()

get_device_model()

get_installed_package_abi(*package*)
Returns the primary abi of the specified package if it is installed on the device, or `None` otherwise.

get_installed_package_version(*package*)
Returns the version (versionName) of the specified package if it is installed on the device, or `None` otherwise.

Added in version 2.1.4

get_pids_of(*process_name*)
Returns a list of PIDs of all processes with the specified name.

get_properties(*context*)
Captures and saves the information from /system/build.prop and /proc/version

get_screen_size()

get_sdk_version()

getprop(*prop=None*)
Returns parsed output of Android getprop command. If a property is specified, only the value for that property will be returned (with `None` returned if the property doesn't exist. Otherwise, `wlauto.utils.android.AndroidProperties` will be returned, which is a dict-like object.

hard_reset()

initialize(**args*, ***kwargs*)

install(*filepath*, *timeout=30*, *with_name=None*, *replace=False*)

install_apk(*filepath*, *timeout=300*, *replace=False*, *allow_downgrade=False*)

install_executable(*filepath*, *with_name=None*)
Installs a binary executable on device. Returns the path to the installed binary, or `None` if the installation has failed. Optionally, `with_name` parameter may be used to specify a different name under which the executable will be installed.

Added in version 2.1.3. Updated in version 2.1.5 with `with_name` parameter.

is_installed(*name*)

is_rooted

is_screen_on()
Returns `True` if the device screen is currently on, `False` otherwise.

kick_off(*command*, *as_root=None*)
Like execute but closes adb session and returns immediately, leaving the command running on the device (this is different from execute(background=True) which keeps adb connection open and returns a subprocess object).

Added in version 2.1.4

list_packages()
List packages installed on the device.

Added in version 2.1.4

listdir(*path*, *as_root=False*, ***kwargs*)

long_delay = 6

package_is_installed(*package_name*)
Returns `True` the if a package with the specified name is installed on the device, and `False` otherwise.

Added in version 2.1.4

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

perform_unlock_swipe()

ping()

platform = 'android'

ps(***kwargs*)
Returns the list of running processes on the device. Keyword arguments may be used to specify simple filters for columns.

Added in version 2.1.4

pull_file(*source*, *dest*, *as_root=False*, *timeout=30*)
Modified in version 2.1.4: added `as_root` parameter.

push_file(*source*, *dest*, *as_root=False*, *timeout=30*)
Modified in version 2.1.4: added `as_root` parameter.

ready_timeout = 60

refresh_device_files(*file_list*)
Depending on the devices android version and root status, determine the appropriate method of forcing a re-index of the mediaserver cache for a given list of files.

reset()

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

start()

stop()

supported_abi

uninstall(*package*)

uninstall_executable(*executable_name*)
Added in version 2.1.3.

validate(**args*, ***kwargs*)

class wlauto.common.android.device.**BigLittleDevice**(***kwargs*)
Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.common.android.resources module

class wlauto.common.android.resources.**ApkFile**(*owner*, *platform=None*, *uiauto=False*, *package=None*)
Bases: *wlauto.common.resources.FileResource*

name = 'apk'

class wlauto.common.android.resources.**JarFile**(*owner*)
Bases: *wlauto.common.resources.FileResource*

name = 'jar'

class wlauto.common.android.resources.**ReventFile**(*owner*, *stage*)
Bases: *wlauto.common.resources.FileResource*

name = 'revent'

wlauto.common.android.workload module

wlauto.common.android.workload.**AndroidBenchmark**
alias of *ApkWorkload*

class wlauto.common.android.workload.**AndroidUiAutoBenchmark**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.UiAutomatorWorkload*, *wlauto.common.android.workload.ApkWorkload*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.common.android.workload.**AndroidUxPerfWorkload**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

aliases = AC([])

artifacts = AC([])

core_modules = []

delete_assets()

deployable_assets = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

push_assets(*context*)

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

class wlauto.common.android.workload.**AndroidUxPerfWorkloadMeta**

Bases: *wlauto.core.extension.ExtensionMeta*

to_propagate = [('parameters', <class 'wlauto.core.extension.Param'>, <class 'wlauto.c

class wlauto.common.android.workload.**ApkWorkload**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

A workload based on an APK file.

Defines the following attributes:

Package The package name of the app. This is usually a Java-style name of the form `com.companyname.appname`.

Activity This is the initial activity of the app. This will be used to launch the app during the setup. Many applications do not specify a launch activity so this may be left blank if necessary.

View The class of the main view pane of the app. This needs to be defined in order to collect SurfaceFlinger-derived statistics (such as FPS) for the app, but may otherwise be left as `None`.

Launch_main If `False`, the default activity will not be launched (during setup), allowing workloads to start the app with an intent of their choice in the run step. This is useful for apps without a launchable default/main activity or those where it cannot be launched without intent data (which is provided at the run phase).

Install_timeout Timeout for the installation of the APK. This may vary wildly based on the size and nature of a specific APK, and so should be defined on per-workload basis.

Note: To a lesser extent, this will also vary based on the the device and the nature of adb connection (USB vs Ethernet), so, as with all timeouts, so leeway must be included in the specified value.

Min_apk_version The minimum supported apk version for this workload. May be `None`.

Max_apk_version The maximum supported apk version for this workload. May be `None`.

Note: Both package and activity for a workload may be obtained from the APK using the `aapt` tool that comes with the ADT (Android Developemnt Tools) bundle.

---

activity = None

aliases = AC([])

artifacts = AC([])

check_host_version()

core_modules = []

do_post_install(*context*)
May be overwritten by derived classes.

finalize(**args*, ***kwargs*)

force_install_apk(*context*, *host_version*)

initialize(**args*, ***kwargs*)

install_apk(*context*, *replace=False*)

kill_background()

launch_application()

launch_main = True

launch_package()

max_apk_version = None

min_apk_version = None

package = None

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

prefer_host_apk(*context*, *host_version*, *target_version*)

prefer_target_apk(*context*, *host_version*, *target_version*)

reset(*context*)

run(*context*)

setup(*context*)

setup_workload_apk(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

validate_version(*version*)

verify_apk_version(*target_version*, *target_abi*, *host_version*)

view = None

class wlauto.common.android.workload.**GameWorkload**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.ApkWorkload*, *wlauto.common.linux.workload.ReventWorkload*

GameWorkload is the base class for all the workload that use revent files to run.

For more in depth details on how to record revent files, please see *revent*. To subclass this class, please refer to *Adding revent-dependent Workload:*.

Additionally, this class defines the following attributes:

Asset_file A tarball containing additional assets for the workload. These are the assets that are not part of the APK but would need to be downloaded by the workload (usually, on first run of the app). Since the presence of a network connection cannot be assumed on some devices, this provides an alternative means of obtaining the assets.

Saved_state_file A tarball containing the saved state for a workload. This tarball gets deployed in the same way as the asset file. The only difference being that it is usually much slower and re-deploying the tarball should alone be enough to reset the workload to a known state (without having to reinstall the app or re-deploy the other assets).

Loading_time Time it takes for the workload to load after the initial activity has been started.

aliases = AC([])

artifacts = AC([])

asset_file = None

check_state(*context*, *phase*)

core_modules = []

do_post_install(*context*)

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

loading_time = 10

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

reset(*context*)

run(*context*)

saved_state_file = None

setup(*context*)

setup_required = True

supported_platforms = ['android']

teardown(*context*)

validate(**args*, ***kwargs*)

view = 'SurfaceView'

class wlauto.common.android.workload.**UiAutomatorWorkload**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

Base class for all workloads that rely on a UI Automator APK file.

This class should be subclassed by workloads that rely on android UiAutomator to work. This class handles installing the UI Automator APK to the device and invoking it to run the workload. By default, it will look for the `*.apk` file in the same directory as the .py file for the workload (this can be changed by overriding the `uiauto_file` property in the subclassing workload).

To inintiate UI Automation, the fully-qualified name of the Java class and the corresponding method name are needed. By default, the package part of the class name is derived from the class file, and class and method names are `UiAutomation` and `runUiAutomaton` respectively. If you have generated the boilder plate for the UiAutomatior code using `create_workloads` utility, then everything should be named correctly. If you're creating the Java project manually, you need to make sure the names match what is expected, or you could override `uiauto_package`, `uiauto_class` and `uiauto_method` class attributes with the value that match your Java code.

You can also pass parameters to the APK file. To do this add the parameters to `self.uiauto_params` dict inside your class's `__init__` or `setup` methods.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

run_timeout = 600

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

uiauto_class = 'UiAutomation'

uiauto_method = 'android.support.test.runner.AndroidJUnitRunner'

uiauto_package = ''

uninstall_uiauto_apk = True

update_result(*context*)

validate(**args*, ***kwargs*)

Module contents

wlauto.common.gem5 package

Submodules

wlauto.common.gem5.device module

class wlauto.common.gem5.device.**BaseGem5Device**
Bases: object

Base implementation for a gem5-based device

This class is used as the base class for OS-specific devices such as the G3m5LinuxDevice and the Gem5AndroidDevice. The majority of the gem5-specific functionality is included here.

Note: When inheriting from this class, make sure to inherit from this class prior to inheriting from the OS-specific class, i.e. LinuxDevice, to ensure that the methods are correctly overridden.

capture_screen(*filepath*)

checkpoint_gem5(*end_simulation=False*)
Checkpoint the gem5 simulation, storing all system state

close()

connect()
Connect to the gem5 simulation and wait for Android to boot. Then, create checkpoints, and mount the VirtIO device.

connect_gem5()
Connect to the telnet port of the gem5 simulation.

We connect, and wait for the prompt to be found. We do not use a timeout for this, and wait for the prompt in a while loop as the gem5 simulation can take many hours to reach a prompt when booting the system. We also inject some newlines periodically to try and force gem5 to show a prompt. Once the prompt has been found, we replace it with a unique prompt to ensure that we are able to match it properly. We also disable the echo as this simplifies parsing the output when executing commands on the device.

default_timeout = 3600

delay = 3600

delete_file(*filepath*, ***kwargs*)
Delete a file on the device

deploy_m5(*context*, *force=False*)
Deploys the m5 binary to the device and returns the path to the binary on the device.

Parameters **force** – by default, if the binary is already present on the device, it will not be deployed again. Setting force to `True` overrides that behaviour and ensures that the binary is always copied. Defaults to `False`.

Returns The on-device path to the m5 binary.

disconnect()
Close and disconnect from the gem5 simulation. Additionally, we remove the temporary directory used to pass files into the simulation.

execute(*command*, *timeout=1000*, *check_exit_code=True*, *background=False*, *as_root=False*, *busybox=False*, ***kwargs*)

file_exists(*filepath*)
Check if a file exists

find_prompt()

forward_port(*_*)

gem5_shell(*command*, *as_root=False*, *timeout=None*, *check_exit_code=True*, *sync=True*)
Execute a command in the gem5 shell

This wraps the telnet connection to gem5 and processes the raw output.

This method waits for the shell to return, and then will try and separate the output from the command from the command itself. If this fails, warn, but continue with the potentially wrong output.

The exit code is also checked by default, and non-zero exit codes will raise a DeviceError.

gem5_util(*command*)
Execute a gem5 utility command using the m5 binary on the device

get_directory(*context*, *directory*)
Pull a directory from the device

get_pids_of(*process_name*)
Returns a list of PIDs of all processes with the specified name.

get_properties(*context*)
Get the property files from the device

init_gem5(*_*)
Start gem5, find out the telnet port and connect to the simulation.

We first create the temporary directory used by VirtIO to pass files into the simulation, as well as the gem5 output directory.We then create files for the standard output and error for the gem5 process. The gem5 process then is started.

is_rooted

long_delay = 10800

mount_virtio()
Mount the VirtIO device in the simulated system.

move_to_temp_dir(*source*)
Move a file to the temporary directory on the host for copying to the gem5 device

parameters = [Param({'kind': <type 'str'>, 'mandatory': False, 'name': 'gem5_binary

path_module = 'posixpath'

ping()

platform = None

pull_file(*source*, *dest*, ***kwargs*)
Pull a file from the gem5 device using m5 writefile

The file is copied to the local directory within the guest as the m5 writefile command assumes that the file is local. The file is then written out to the host system using writefile, prior to being moved to the destination on the host.

push_file(*source*, *dest*, ***kwargs*)
Push a file to the gem5 device using VirtIO

The file to push to the device is copied to the temporary directory on the host, before being copied within the simulation to the destination. Checks, in the form of 'ls' with error code checking, are performed to ensure that the file is copied to the destination.

ready_timeout = 10800

reset()

resize_shell()
Resize the shell to avoid line wrapping issues.

start_gem5()
Starts the gem5 simulator, and parses the output to get the telnet port.

sync_gem5_shell()
Synchronise with the gem5 shell.

Write some unique text to the gem5 device to allow us to synchronise with the shell output. We actually get two prompts so we need to match both of these.

validate()

wait_for_boot()

Module contents

wlauto.common.linux package

Submodules

wlauto.common.linux.device module

class wlauto.common.linux.device.**BaseLinuxDevice**(***kwargs*)
Bases: *wlauto.core.device.Device*

abi

aliases = AC([])

artifacts = AC([])

core_modules = []

cpuinfo

deploy_busybox(*context*, *force=False*)
Deploys the busybox binary to the specified device and returns the path to the binary on the device.

Parameters

- **context** – an instance of ExecutionContext
- **force** – by default, if the binary is already present on the device, it will not be deployed again. Setting force to `True` overrides that behavior and ensures that the binary is always copied. Defaults to `False`.

Returns The on-device path to the busybox binary.

disable_cpu(*cpu*)
Disable the specified core.

Parameters **cpu** – CPU core to disable. This must be the full name as it appears in sysfs, e.g. "cpu0".

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

enable_cpu(*cpu*)
Enable the specified core.

Parameters **cpu** – CPU core to enable. This must be the full name as it appears in sysfs, e.g. "cpu0".

file_transfer_cache

finalize(**args*, ***kwargs*)

get_binary_path(*name*, *search_system_binaries=True*)
Searches the devices `binary_directory` for the given binary, if it cant find it there it tries using which to find it.

Parameters

- **name** – The name of the binary
- **search_system_binaries** – By default this function will try using which to find the binary if it isn't in `binary_directory`. When this is set to `False` it will not try this.

Returns The on-device path to the binary.

get_device_model()

get_number_of_online_cores(*core*)

get_online_cpus(*c*)

get_pids_of(*process_name*)

get_properties(*context*)

get_sysfile_value(*sysfile*, *kind=None*, *binary=False*)
Get the contents of the specified sysfile.

Parameters

- **sysfile** – The file who's contents will be returned.
- **kind** – The type of value to be expected in the sysfile. This can be any Python callable that takes a single str argument. If not specified or is None, the contents will be returned as a string.
- **binary** – Whether the value should be encoded into base64 for reading to deal with binary format.

get_sysfile_values()
Returns a dict mapping paths of sysfiles that were previously set to their current values.

has_gpu = True

hotplug_cpu(*cpu*, *online*)
Hotplug the specified CPU either on or off. See https://www.kernel.org/doc/Documentation/cpu-hotplug.txt

Parameters

- **cpu** – The CPU for which the governor is to be set. This must be the full name as it appears in sysfs, e.g. "cpu0".
- **online** – CPU will be enabled if this value bool()'s to True, and will be disabled otherwise.

initialize(**args*, ***kwargs*)

install_if_needed(*host_path*, *search_system_binaries=True*)
Similar to get_binary_path but will install the binary if not found.

Parameters

- **host_path** – The path to the binary on the host
- **search_system_binaries** – By default this function will try using which to find the binary if it isn't in `binary_directory`. When this is set to `False` it will not try this.

Returns The on-device path to the binary.

invoke(*binary*, *args=None*, *in_directory=None*, *on_cpus=None*, *background=False*, *as_root=False*, *timeout=30*)
Executes the specified binary under the specified conditions.

Binary binary to execute. Must be present and executable on the device.

Args arguments to be passed to the binary. The can be either a list or a string.

In_directory execute the binary in the specified directory. This must be an absolute path.

On_cpus taskset the binary to these CPUs. This may be a single `int` (in which case, it will be interpreted as the mask), a list of `int`s, in which case this will be interpreted as the list of cpus, or string, which will be interpreted as a comma-separated list of cpu ranges, e.g. `"0,4-7"`.

Background If `True`, a `subprocess.Popen` object will be returned straight away. If `False` (the default), this will wait for the command to terminate and return the STDOUT output

As_root Specify whether the command should be run as root

Timeout If the invocation does not terminate within this number of seconds, a `TimeoutError` exception will be raised. Set to `None` if the invocation should not timeout.

is_directory(*filepath*)

is_file(*filepath*)

is_installed(*name*)

is_network_connected()
Checks for internet connectivity on the device by pinging IP address provided.

Parameters **ip_address** – IP address to ping. Default is Google's public DNS server (8.8.8.8)

Returns `True` if internet is available, `False` otherwise.

kill(*pid*, *signal=None*, *as_root=False*)
Kill the specified process.

param pid PID of the process to kill.

param signal Specify which singal to send to the process. This must be a valid value for -s option of kill. Defaults to `None`.

Modified in version 2.1.4: added `signal` parameter.

killall(*process_name*, *signal=None*, *as_root=None*)
Kill all processes with the specified name.

param process_name The name of the process(es) to kill.

param signal Specify which singal to send to the process. This must be a valid value for -s option of kill. Defaults to `None`.

Modified in version 2.1.5: added `as_root` parameter.

list_file_systems()

number_of_cores
Added in version 2.1.4.

online_cpus

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

path_module = 'posixpath'

ps(***kwargs*)

resource_cache

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

set_number_of_online_cores(*core*, *number*)

set_sysfile_value(*sysfile*, *value*, *verify=True*, *binary=False*)
Set the value of the specified sysfile. By default, the value will be checked afterwards. Can be overridden by setting `verify` parameter to `False`. By default binary values will not be written correctly this can be changed by setting the `binary` parameter to `True`.

set_sysfile_values(*params*)
The plural version of `set_sysfile_value`. Takes a single parameter which is a mapping of file paths to values to be set. By default, every value written will be verified. This can be disabled for individual paths by appending `'!'` to them. To enable values being written as binary data, a `'^'` can be prefixed to the path.

supported_abi

validate(**args*, ***kwargs*)

class wlauto.common.linux.device.**FstabEntry**(*device*, *mount_point*, *fs_type*, *options*, *dump_freq*, *pass_num*)
Bases: `tuple`

device
Alias for field number 0

dump_freq
Alias for field number 4

fs_type
Alias for field number 2

mount_point
Alias for field number 1

options
Alias for field number 3

pass_num
Alias for field number 5

class wlauto.common.linux.device.**LinuxDevice**(**args*, ***kwargs*)

Bases: *wlauto.common.linux.device.BaseLinuxDevice*

aliases = AC([])

artifacts = AC([])

boot(*hard=False*, ***kwargs*)

capture_screen(*filepath*)

connect()

core_modules = []

default_timeout = 30

delay = 2

delete_file(*filepath*, *as_root=False*)

disconnect()

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

ensure_screen_is_on()

execute(*command*, *timeout=30*, *check_exit_code=True*, *background=False*, *as_root=False*, *strip_colors=True*, ***kwargs*)

Execute the specified command on the device using adb.

Parameters:

param command The command to be executed. It should appear exactly as if you were typing it into a shell.

param timeout Time, in seconds, to wait for adb to return before aborting and raising an error. Defaults to `AndroidDevice.default_timeout`.

param check_exit_code If `True`, the return code of the command on the Device will be check and exception will be raised if it is not 0. Defaults to `True`.

param background If `True`, will execute create a new ssh shell rather than using the default session and will return it immediately. If this is `True`, `timeout`, `strip_colors` and (obvisously) `check_exit_code` will be ignored; also, with this, `as_root=True` is only valid if `username` for the device was set to `root`.

param as_root If `True`, will attempt to execute command in privileged mode. The device must be rooted, otherwise an error will be raised. Defaults to `False`.

Added in version 2.1.3

Returns If `background` parameter is set to `True`, the subprocess object will be returned; otherwise, the contents of STDOUT from the device will be returned.

file_exists(*filepath*)

finalize(**args*, ***kwargs*)

get_pids_of(*process_name*)

Returns a list of PIDs of all processes with the specified name.

hard_reset()

initialize(**args*, ***kwargs*)

insmod(*path*)
Install a kernel module located on the host on the target device.

install(*filepath*, *timeout=30*, *with_name=None*)

install_executable(*filepath*, *timeout=30*, *with_name=None*)

is_rooted

is_screen_on()

kick_off(*command*, *as_root=None*)
Like execute but closes ssh session and returns immediately, leaving the command running on the device (this is different from execute(background=True) which keeps ssh connection open and returns a subprocess object).

listdir(*path*, *as_root=False*, ***kwargs*)

long_delay = 6

lsmod()
List loaded kernel modules.

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

ping()

platform = 'linux'

ps(***kwargs*)

pull_file(*source*, *dest*, *as_root=False*, *timeout=30*)

push_file(*source*, *dest*, *as_root=False*, *timeout=30*)

ready_timeout = 60

reset()

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

uninstall(*executable_name*)

uninstall_executable(*executable_name*)

validate(**args*, ***kwargs*)

class wlauto.common.linux.device.**LsmodEntry**(*name*, *size*, *use_count*, *used_by*)
Bases: tuple

name
Alias for field number 0

size
Alias for field number 1

use_count
Alias for field number 2

used_by
Alias for field number 3

class wlauto.common.linux.device.**PsEntry**(*user*, *pid*, *ppid*, *vsize*, *rss*, *wchan*, *pc*, *state*, *name*)
Bases: tuple

name
Alias for field number 8

pc
Alias for field number 6

pid
Alias for field number 1

ppid
Alias for field number 2

rss
Alias for field number 4

state
Alias for field number 7

user
Alias for field number 0

vsize
Alias for field number 3

wchan
Alias for field number 5

wlauto.common.linux.workload module

class wlauto.common.linux.workload.**ReventWorkload**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n A workload for playing back revent recordings. You can supply three

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

setup_required = False

teardown(*context*)

teardown_required = False

update_result(*context*)

validate(**args*, ***kwargs*)

Module contents

Submodules

wlauto.common.resources module

class wlauto.common.resources.**Executable**(*owner*, *platform*, *filename*)
Bases: *wlauto.common.resources.FileResource*

name = 'executable'

class wlauto.common.resources.**ExtensionAsset**(*owner*, *path*)
Bases: *wlauto.common.resources.File*

name = 'extension_asset'

class wlauto.common.resources.**File**(*owner*, *path*, *url=None*)
Bases: *wlauto.common.resources.FileResource*

name = 'file'

class wlauto.common.resources.**FileResource**(*owner*)
Bases: *wlauto.core.resource.Resource*

Base class for all resources that are a regular file in the file system.

delete(*instance*)

Module contents

wlauto.core package

Submodules

wlauto.core.agenda module

class wlauto.core.agenda.**Agenda**(*source=None*)
Bases: object

add_workload_entry(*w*)

get_workload_entry(*w*)

class wlauto.core.agenda.**AgendaEntry**
Bases: object

to_dict()

class wlauto.core.agenda.**AgendaGlobalEntry**(***kwargs*)
Bases: *wlauto.core.agenda.AgendaEntry*

Workload configuration global to all workloads.

class wlauto.core.agenda.**AgendaSectionEntry**(*agenda*, ***kwargs*)
Bases: *wlauto.core.agenda.AgendaEntry*

Specifies execution of a workload, including things like the number of iterations, device runtime_parameters configuration, etc.

to_dict()

class wlauto.core.agenda.**AgendaWorkloadEntry**(***kwargs*)
Bases: *wlauto.core.agenda.AgendaEntry*

Specifies execution of a workload, including things like the number of iterations, device runtime_parameters configuration, etc.

wlauto.core.agenda.**dict_constructor**(*loader*, *node*)

wlauto.core.agenda.**dict_representer**(*dumper*, *data*)

wlauto.core.agenda.**get_aliased_param**(*d*, *aliases*, *default=None*, *pop=True*)

wlauto.core.bootstrap module

class wlauto.core.bootstrap.**ConfigLoader**
Bases: object

This class is responsible for loading and validating config files.

get_config_paths()

log_file

meta_directory

update(*source*)

update_from_dict(*source*)

update_from_file(*source*)

wlauto.core.bootstrap.**init_environment**(*env_root*, *dep_dir*, *extension_paths*, *overwrite_existing=False*)
Initialise a fresh user environment creating the workload automation

wlauto.core.command module

class wlauto.core.command.**Command**(*subparsers*)
Bases: *wlauto.core.extension.Extension*

Defines a Workload Automation command. This will be executed from the command line as `wa <command> [args ...]`. This defines the name to be used when invoking wa, the code that will actually be executed on invocation and the argument parser to be used to parse the reset of the command line arguments.

aliases = AC([])

artifacts = AC([])

core_modules = []

description = None

epilog = None

execute(*args*)
Execute this command.

Args An `argparse.Namespace` containing command line arguments (as returned by `argparse.ArgumentParser.parse_args()`. This would usually be the result of invoking `self.parser`.

finalize(**args*, ***kwargs*)

formatter_class = None

help = None

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

usage = None

validate(**args*, ***kwargs*)

wlauto.core.configuration module

class wlauto.core.configuration.**ConfigurationJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*obj*)

class wlauto.core.configuration.**RebootPolicy**(*policy*)

Bases: object

Represents the reboot policy for the execution – at what points the device should be rebooted. This, in turn, is controlled by the policy value that is passed in on construction and would typically be read from the user's settings. Valid policy values are:

Never The device will never be rebooted.

As_needed Only reboot the device if it becomes unresponsive, or needs to be flashed, etc.

Initial The device will be rebooted when the execution first starts, just before executing the first workload spec.

Each_spec The device will be rebooted before running a new workload spec.

Each_iteration The device will be rebooted before each new iteration.

can_reboot

perform_initial_boot

reboot_on_each_iteration

reboot_on_each_spec

valid_policies = ['never', 'as_needed', 'initial', 'each_spec', 'each_iteration']

class wlauto.core.configuration.**RunConfiguration**(*ext_loader*)

Bases: object

Loads and maintains the unified configuration for this run. This includes configuration for WA execution as a whole, and parameters for specific specs.

WA configuration mechanism aims to be flexible and easy to use, while at the same time providing storing validation and early failure on error. To meet these requirements, the implementation gets rather complicated. This is going to be a quick overview of the underlying mechanics.

Note: You don't need to know this to use WA, or to write extensions for it. From the point of view of extension writers, configuration from various sources "magically" appears as attributes of their classes. This explanation peels back the curtain and is intended for those who, for one reason or another, need to understand how the magic works.

---

terminology

run

A single execution of a WA agenda.

run config(uration) (object)

An instance of this class. There is one per run.

config(uration) item

A single configuration entry or "setting", e.g. the device interface to use. These can be for the run as a whole, or for a specific extension.

(workload) spec

A specification of a single workload execution. This combines workload configuration with things like the number of iterations to run, which instruments to enable, etc. More concretely, this is an instance of *WorkloadRunSpec*.

overview

There are three types of WA configuration:

1. "Meta" configuration that determines how the rest of the configuration is processed (e.g. where extensions get loaded from). Since this does not pertain to *run* configuration, it will not be covered further.
2. Global run configuration, e.g. which workloads, result processors and instruments will be enabled for a run.
3. Per-workload specification configuration, that determines how a particular workload instance will get executed (e.g. what workload parameters will be used, how many iterations.

run configuration

Run configuration may appear in a config file (usually `~/.workload_automation/config.py`), or in the `config` section of an agenda. Configuration is specified as a nested structure of dictionaries (associative arrays, or maps) and lists in the syntax following the format implied by the file extension (currently, YAML and Python are supported). If the same configuration item appears in more than one source, they are merged with conflicting entries taking the value from the last source that specified them.

In addition to a fixed set of global configuration items, configuration for any WA Extension (instrument, result processor, etc) may also be specified, namespaced under the extension's name (i.e. the extensions name is a key in the global config with value being a dict of parameters and their values). Some Extension parameters also specify a "global alias" that may appear at the top-level of the config rather than under the Extension's name. It is *not* an error to specify configuration for an Extension that has not been enabled for a particular run; such configuration will be ignored.

per-workload configuration

Per-workload configuration can be specified in three places in the agenda: the workload entry in the `workloads` list, the `global` entry (configuration there will be applied to every workload entry), and in a

section entry in `sections` list (configuration in every section will be applied to every workload entry separately, creating a "cross-product" of section and workload configurations; additionally, sections may specify their own workload lists).

If they same configuration item appears in more than one of the above places, they will be merged in the following order: `global`, `section`, `workload`, with conflicting scalar values in the later overriding those from previous locations.

Global parameter aliases

As mentioned above, an Extension's parameter may define a global alias, which will be specified and picked up from the top-level config, rather than config for that specific extension. It is an error to specify the value for a parameter both through a global alias and through extension config dict in the same configuration file. It is, however, possible to use a global alias in one file, and specify extension configuration for the same parameter in another file, in which case, the usual merging rules would apply.

Loading and validation of configuration

Validation of user-specified configuration happens at several stages of run initialisation, to ensure that appropriate context for that particular type of validation is available and that meaningful errors can be reported, as early as is feasible.

- Syntactic validation is performed when configuration is first loaded. This is done by the loading mechanism (e.g. YAML parser), rather than WA itself. WA propagates any errors encountered as `ConfigErrors`.
- Once a config file is loaded into a Python structure, it scanned to extract settings. Static configuration is validated and added to the config. Extension configuration is collected into a collection of "raw" config, and merged as appropriate, but is not processed further at this stage.
- Once all configuration sources have been processed, the configuration as a whole is validated (to make sure there are no missing settings, etc).
- Extensions are loaded through the run config object, which instantiates them with appropriate parameters based on the "raw" config collected earlier. When an Extension is instantiated in such a way, its config is "officially" added to run configuration tracked by the run config object. Raw config is discarded at the end of the run, so that any config that wasn't loaded in this way is not recoded (as it was not actually used).
- Extension parameters a validated individually (for type, value ranges, etc) as they are loaded in the Extension's __init__.
- An extension's `validate()` method is invoked before it is used (exactly when this happens depends on the extension's type) to perform any final validation *that does not rely on the target being present* (i.e. this would happen before WA connects to the target). This can be used perform inter-parameter validation for an extension (e.g. when valid range for one parameter depends on another), and more general WA state assumptions (e.g. a result processor can check that an instrument it depends on has been installed).
- Finally, it is the responsibility of individual extensions to validate any assumptions they make about the target device (usually as part of their `setup()`).

Handling of Extension aliases.

WA extensions can have zero or more aliases (not to be confused with global aliases for extension *parameters*). An extension allows associating an alternative name for the extension with a set of parameter values. In other words aliases associate common configurations for an extension with a name, providing a shorthand for it. For example, "t-rex_offscreen" is an alias for "glbenchmark" workload that specifies that "use_case" should be "t-rex" and "variant" should be "offscreen".

special loading rules

Note that as a consequence of being able to specify configuration for *any* Extension namespaced under the Extension's name in the top-level config, two distinct mechanisms exist form configuring devices and workloads.

This is valid, however due to their nature, they are handled in a special way. This may be counter intuitive, so configuration of devices and workloads creating entries for their names in the config is discouraged in favour of using the "normal" mechanisms of configuring them (`device_config` for devices and workload specs in the agenda for workloads).

In both cases (devices and workloads), "normal" config will always override named extension config *irrespective of which file it was specified in*. So a `adb_name` name specified in `device_config` inside `~/.workload_automation/config.py` will override `adb_name` specified for `juno` in the agenda (even when device is set to "juno").

Again, this ignores normal loading rules, so the use of named extension configuration for devices and workloads is discouraged. There maybe some situations where this behaviour is useful however (e.g. maintaining configuration for different devices in one config file).

all_instrumentation

default_execution_order = 'by_iteration'

default_reboot_policy = 'as_needed'

finalize()
This must be invoked once all configuration sources have been loaded. This will do the final processing, setting instrumentation and result processor configuration for the run And making sure that all the mandatory config has been specified.

general_config = [<wlauto.core.configuration.RunConfigurationItem object>, <wlauto.cor

get_extension(*ext_name*, **args*)

get_reboot_policy()

ignore_names = ['logging', 'remote_assets_mount_point']

load_config(*source*)
Load configuration from the specified source. The source must be either a path to a valid config file or a dict-like object. Currently, config files can be either python modules (.py extension) or YAML documents (.yaml extension).

reboot_policy

serialize(*wfh*)

set_agenda(*agenda*, *selectors=None*)
Set the agenda for this run; Unlike with config files, there can only be one agenda.

set_reboot_policy(*value*)

to_dict()

workload_config = [<wlauto.core.configuration.RunConfigurationItem object>, <wlauto.co

class wlauto.core.configuration.**RunConfigurationItem**(*name*, *category*, *method*)
Bases: `object`

This represents a predetermined "configuration point" (an individual setting) and describes how it must be handled when encountered.

combine(**args*)
Combine the provided values according to the method for this configuration item. Order matters – values are assumed to be in the order they were specified by the user. The resulting value is also checked to patch the specified type.

valid_categories = {'dict': {}, 'scalar': None, 'list': []}

valid_methods = ['keep', 'replace', 'merge']

class wlauto.core.configuration.**SharedConfiguration**
Bases: object

class wlauto.core.configuration.**WorkloadRunSpec**(*id=None*, *number_of_iterations=None*, *workload_name=None*, *boot_parameters=None*, *label=None*, *section_id=None*, *workload_parameters=None*, *runtime_parameters=None*, *instrumentation=None*, *flash=None*, *classifiers=None*)
Bases: object

Specifies execution of a workload, including things like the number of iterations, device runtime_parameters configuration, etc.

copy()

framework_mandatory_parameters = ['id', 'number_of_iterations']

load(*device*, *ext_loader*)
Loads the workload for the specified device using the specified loader. This must be done before attempting to execute the spec.

mandatory_parameters = ['workload_name']

match_selectors(*selectors*)
Returns True if this spec matches the specified selectors, and False otherwise. selectors must be a dict-like object with attribute names mapping onto selector values. At the moment, only equality selection is supported; i.e. the value of the attribute of the spec must match exactly the corresponding value specified in the selectors dict.

secition

set(*param*, *value*)

to_dict()

validate()

workload

class wlauto.core.configuration.**status_list**
Bases: list

append(*item*)

wlauto.core.device module

Base classes for device interfaces.

Device The base class for all devices. This defines the interface that must be implemented by all devices and therefore any workload and instrumentation can always rely on.

AndroidDevice Implements most of the *Device* interface, and extends it with a number of Android-specific methods.

BigLittleDevice Subclasses AndroidDevice to implement big.LITTLE-specific runtime parameters.

SimpleMulticoreDevice Subclasses `AndroidDevice` to implement homogeneous cores device runtime parameters.

class wlauto.core.device.**RuntimeParameter**(*name*, *getter*, *setter*, *getter_args=None*, *setter_args=None*, *value_name='value'*, *override=False*)

Bases: `object`

A runtime parameter which has its getter and setter methods associated it with it.

class wlauto.core.device.**CoreParameter**(*name*, *getter*, *setter*, *getter_args=None*, *setter_args=None*, *value_name='value'*, *override=False*)

Bases: *wlauto.core.device.RuntimeParameter*

A runtime parameter that will get expanded into a RuntimeParameter for each core type.

get_runtime_parameters(*core_names*)

class wlauto.core.device.**Device**(***kwargs*)

Bases: *wlauto.core.extension.Extension*

Base class for all devices supported by Workload Automation. Defines the interface the rest of WA uses to interact with devices.

name Unique name used to identify the device.

platform The name of the device's platform (e.g. `Android`) this may be used by workloads and instrumentation to assess whether they can run on the device.

working_directory a string of the directory which is going to be used by the workloads on the device.

binaries_directory a string of the binary directory for the device.

has_gpu Should be `True` if the device as a separate GPU, and `False` if graphics processing is done on a CPU.

Note: Pretty much all devices currently on the market have GPUs, however this may not be the case for some development boards.

---

path_module The name of one of the modules implementing the os.path interface, e.g. `posixpath` or `ntpath`. You can provide your own implementation rather than relying on one of the standard library modules, in which case you need to specify the *full* path to you module. e.g. '/home/joebloggs/mypathimp.py'

parameters A list of RuntimeParameter objects. The order of the objects is very important as the setters and getters will be called in the order the RuntimeParameter objects inserted.

active_cores This should be a list of all the currently active cpus in the device in `'/sys/devices/system/cpu/online'`. The returned list should be read from the device at the time of read request.

active_cores = None

aliases = AC([])

artifacts = AC([])

boot(**args*, ***kwargs*)

Perform the seteps necessary to boot the device to the point where it is ready to accept other commands.

Changed in version 2.1.3: no longer expected to wait until boot completes.

capture_screen(*filepath*)
Captures the current device screen into the specified file in a PNG format.

connect(**args*, ***kwargs*)
Establish a connection to the device that will be used for subsequent commands.

Added in version 2.1.3.

core_modules = []

default_working_directory = None

delete_file(*filepath*)
Delete the specified file on the device.

disconnect()
Close the established connection to the device.

dynamic_modules = AC([])

execute(*command*, *timeout=None*, ***kwargs*)
Execute the specified command command on the device and return the output.

Parameters

- **command** – Command to be executed on the device.
- **timeout** – If the command does not return after the specified time, execute() will abort with an error. If there is no timeout for the command, this should be set to 0 or None.

Other device-specific keyword arguments may also be specified.

Returns The stdout output from the command.

file_exists(*filepath*)
Check if the specified file or directory exist on the device.

finalize(**args*, ***kwargs*)

get_pids_of(*process_name*)
Returns a list of PIDs of the specified process name.

get_properties(*output_path*)
Captures and saves the device configuration properties version and any other relevant information. Return them in a dict

get_runtime_parameter_names()

get_runtime_parameters()
returns the runtime parameters that have been set.

get_sysfile_value(*sysfile*, *kind=None*, *binary=False*)
Get the contents of the specified sysfile.

Parameters

- **sysfile** – The file who's contents will be returned.
- **kind** – The type of value to be expected in the sysfile. This can be any Python callable that takes a single str argument. If not specified or is None, the contents will be returned as a string.
- **binary** – Whether the value should be encoded into base64 for reading to deal with binary format.

has_gpu = None

initialize(**args*, ***kwargs*)

install(*filepath*, ***kwargs*)
Install the specified file on the device. What "install" means is device-specific and may possibly also depend on the type of file.

is_network_connected()
Checks if the device is connected to the internet

kill(*pid*, *as_root=False*)
Kill the process with the specified PID.

killall(*process_name*, *as_root=False*)
Kill all running processes with the specified name.

listdir(*path*, ***kwargs*)
List the contents of the specified directory.

name = None

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

path_module = None

ping()
This must return successfully if the device is able to receive commands, or must raise `wlauto.exceptions.DeviceUnresponsiveError` if the device cannot respond.

platform = None

pull_file(*source*, *dest*)
Pull a file from device system onto the host file system.

push_file(*source*, *dest*)
Push a file from the host file system onto the device.

reset()
Initiate rebooting of the device.

Added in version 2.1.3.

runtime_parameters = AC([])

set_runtime_parameters(*params*)
The parameters are taken from the keyword arguments and are specific to a particular device. See the device documentation.

set_sysfile_value(*filepath*, *value*, *verify=True*, *binary=False*)
Write the specified value to the specified file on the device and verify that the value has actually been written.

Parameters

- **file** – The file to be modified.
- **value** – The value to be written to the file. Must be an int or a string convertable to an int.
- **verify** – Specifies whether the value should be verified, once written.
- **binary** – Specifies whether the value should be written as binary data.

Should raise DeviceError if could write value.

sleep(*seconds*)
Sleep for the specified time on the target device.

Parameters **seconds** – Time in seconds to sleep on the device

The sleep is executed on the device using self.execute(). We set the timeout for this command to be 10 seconds longer than the sleep itself to make sure the command has time to complete before we timeout.

start()
This gets invoked before an iteration is started and is endented to help the device manange any internal supporting functions.

stop()
This gets invoked after iteration execution has completed and is endented to help the device manange any internal supporting functions.

uninstall(*filepath*)
Uninstall the specified file on the device. What "uninstall" means is device-specific and may possibly also depend on the type of file.

validate(**args*, ***kwargs*)

class wlauto.core.device.**DeviceMeta**
Bases: *wlauto.core.extension.ExtensionMeta*

to_propagate = [('parameters', <class 'wlauto.core.extension.Param'>, <class 'wlauto.c

wlauto.core.entry_point module

wlauto.core.entry_point.**convert_TERM_into_INT_handler**(*signal*, *frame*)

wlauto.core.entry_point.**load_commands**(*subparsers*)

wlauto.core.entry_point.**main**()

wlauto.core.execution module

This module contains the execution logic for Workload Automation. It defines the following actors:

WorkloadSpec: Identifies the workload to be run and defines parameters under which it should be executed.

Executor: Responsible for the overall execution process. It instantiates and/or intialises the other actors, does any necessary vaidation and kicks off the whole process.

Execution Context: Provides information about the current state of run execution to instrumentation.

RunInfo: Information about the current run.

Runner: This executes workload specs that are passed to it. It goes through stages of execution, emitting an appropriate signal at each step to allow instrumentation to do its stuff.

class wlauto.core.execution.**ByIterationRunner**(*device*, *context*, *result_manager*)
Bases: *wlauto.core.execution.Runner*

Runs the first iteration for all benchmarks first, before proceeding to the next iteration, i.e. A1, B1, C1, A2, B2, C2... instead of A1, A1, B1, B2, C1, C2...

If multiple sections where specified in the agenda, this will run all sections for the first global spec first, followed by all sections for the second spec, etc.

e.g. given sections X and Y, and global specs A and B, with 2 iterations, this will run

X.A1, Y.A1, X.B1, Y.B1, X.A2, Y.A2, X.B2, Y.B2

init_queue(*specs*)

class wlauto.core.execution.**BySectionRunner**(*device*, *context*, *result_manager*)
Bases: *wlauto.core.execution.Runner*

Runs the first iteration for all benchmarks first, before proceeding to the next iteration, i.e. A1, B1, C1, A2, B2, C2... instead of A1, A1, B1, B2, C1, C2...

If multiple sections where specified in the agenda, this will run all specs for the first section followed by all specs for the seciod section, etc.

e.g. given sections X and Y, and global specs A and B, with 2 iterations, this will run

X.A1, X.B1, Y.A1, Y.B1, X.A2, X.B2, Y.A2, Y.B2

init_queue(*specs*)

class wlauto.core.execution.**BySpecRunner**(*device*, *context*, *result_manager*)
Bases: *wlauto.core.execution.Runner*

This is that "classic" implementation that executes all iterations of a workload spec before proceeding onto the next spec.

init_queue(*specs*)

class wlauto.core.execution.**ExecutionContext**(*device*, *config*)
Bases: object

Provides a context for instrumentation. Keeps track of things like current workload and iteration.

This class also provides two status members that can be used by workloads and instrumentation to keep track of arbitrary state. result is reset on each new iteration of a workload; run_status is maintained throughout a Workload Automation run.

add_artifact(*name*, *path*, *kind*, **args*, ***kwargs*)

add_classifiers(***kwargs*)

add_iteration_artifact(*name*, *path*, *kind*, **args*, ***kwargs*)

add_metric(**args*, ***kwargs*)

add_run_artifact(*name*, *path*, *kind*, **args*, ***kwargs*)

current_iteration

default_run_artifacts = [<wlauto.core.extension.Artifact object>]

end_job()

get_artifact(*name*)

initialize()

job_status

next_job(*job*)
Invoked by the runner when starting a new iteration of workload execution.

result

spec

workload

class wlauto.core.execution.**Executor**
Bases: object

The Executor's job is to set up the execution context and pass to a Runner along with a loaded run specification. Once the Runner has done its thing, the Executor performs some final reporint before returning.

The initial context set up involves combining configuration from various sources, loading of requided workloads, loading and installation of instruments and result processors, etc. Static validation of the combined configuration is also performed.

execute(*agenda*, *selectors=None*)
Execute the run specified by an agenda. Optionally, selectors may be used to only selecute a subset of the specified agenda.

Params:

```
:agenda: an ``Agenda`` instance to be executed.
:selectors: A dict mapping selector name to the coresponding values.
```

Selectors

Currently, the following seectors are supported:

ids The value must be a sequence of workload specfication IDs to be executed. Note that if sections are specified inthe agenda, the workload specifacation ID will be a combination of the section and workload IDs.

execute_postamble()
This happens after the run has completed. The overall results of the run are summarised to the user.

class wlauto.core.execution.**RandomRunner**(*device*, *context*, *result_manager*)
Bases: *wlauto.core.execution.Runner*

This will run specs in a random order.

init_queue(*specs*)

class wlauto.core.execution.**RunInfo**(*config*)
Bases: object

Information about the current run, such as its unique ID, run time, etc.

to_dict()

class wlauto.core.execution.**Runner**(*device*, *context*, *result_manager*)
Bases: object

This class is responsible for actually performing a workload automation run. The main responsibility of this class is to emit appropriate signals at the various stages of the run to allow things like traces an other instrumentation to hook into the process.

This is an abstract base class that defines each step of the run, but not the order in which those steps are executed, which is left to the concrete derived classes.

config

current_job

init_queue(*specs*)

next_job

previous_job

run()

spec_changed

spec_will_change

class wlauto.core.execution.**RunnerJob**(*spec*, *retry=0*)
Bases: object

Represents a single execution of a RunnerJobDescription. There will be one created for each iteration specified by RunnerJobDescription.number_of_iterations.

wlauto.core.extension module

class wlauto.core.extension.**Alias**(*name*, ***kwargs*)
Bases: object

This represents a configuration alias for an extension, mapping an alternative name to a set of parameter values, effectively providing an alternative set of default values.

validate(*ext*)

class wlauto.core.extension.**AliasCollection**
Bases: *wlauto.core.extension.AttributeCollection*

class wlauto.core.extension.**Artifact**(*name*, *path*, *kind*, *level='run'*, *mandatory=False*, *description=None*)
Bases: object

This is an artifact generated during execution/post-processing of a workload. Unlike metrics, this represents an actual artifact, such as a file, generated. This may be "result", such as trace, or it could be "meta data" such as logs. These are distinguished using the kind attribute, which also helps WA decide how it should be handled. Currently supported kinds are:

log A log file. Not part of "results" as such but contains information about the run/workload execution that be useful for diagnostics/meta analysis.

meta A file containing metadata. This is not part of "results", but contains information that may be necessary to reproduce the results (contrast with log artifacts which are *not* necessary).

data This file contains new data, not available otherwise and should be considered part of the "results" generated by WA. Most traces would fall into this category.

export Exported version of results or some other artifact. This signifies that this artifact does not contain any new data that is not available elsewhere and that it may be safely discarded without losing information.

raw Signifies that this is a raw dump/log that is normally processed to extract useful information and is then discarded. In a sense, it is the opposite of export, but in general may also be discarded.

Note: whether a file is marked as log/data or raw depends on how important it is to preserve this file, e.g. when archiving, vs how much space it takes up. Unlike export artifacts which are (almost) always ignored by other exporters as that would never result in data loss, raw files *may* be processed by exporters if they decided that the risk of losing potentially (though unlikely) useful data is greater than the time/space cost of handling the artifact (e.g. a database uploader may choose to ignore raw artifacts, where as a network filer archiver may choose to archive them).

ITERATION = 'iteration'

RUN = 'run'

exists(*context*)
Returns `True` if artifact exists within the specified context, and `False` otherwise.

to_dict()

valid_kinds = ['log', 'meta', 'data', 'export', 'raw']

class wlauto.core.extension.**AttributeCollection**(*attrcls*, *owner*)
Bases: `object`

Accumulator for extension attribute objects (such as Parameters or Artifacts). This will replace any class member list accumulating such attributes through the magic of metaprogramming*[0].

add(*p*)

append(*p*)

values

class wlauto.core.extension.**Extension**(***kwargs*)
Bases: `object`

Base class for all WA extensions. An extension is basically a plug-in. It extends the functionality of WA in some way. Extensions are discovered and loaded dynamically by the extension loader upon invocation of WA scripts. Adding an extension is a matter of placing a class that implements an appropriate interface somewhere it would be discovered by the loader. That "somewhere" is typically one of the extension subdirectories under `~/.workload_automation/`.

aliases = AC([])

artifacts = AC([])

can(*capability*)
Check if this extension has the specified capability. The alternative method `can` is identical to this. Which to use is up to the caller depending on what makes semantic sense in the context of the capability, e.g. `can('hard_reset')` vs `has('active_cooling')`.

check_artifacts(*context*, *level*)
Make sure that all mandatory artifacts have been generated.

core_modules = []

dependencies_directory

finalize(**args*, ***kwargs*)

get_config()
Returns current configuration (i.e. parameter values) of this extension.

classmethod get_default_config()

has(*capability*)
Check if this extension has the specified capability. The alternative method `can` is identical to this. Which to use is up to the caller depending on what makes semantic sense in the context of the capability, e.g. `can('hard_reset')` vs `has('active_cooling')`.

initialize(**args*, ***kwargs*)

kind = None

[0] which is totally safe and not going backfire in any way...

load_modules(*loader*)

Load the modules specified by the "modules" Parameter using the provided loader. A loader can be any object that has an atribute called "get_module" that implements the following signature:

```
get_module(name, owner, **kwargs)
```

and returns an instance of *wlauto.core.extension.Module*. If the module with the specified name is not found, the loader must raise an appropriate exception.

name = None

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.core.extension.**ExtensionMeta**

Bases: type

This basically adds some magic to extensions to make implementing new extensions, such as workloads less complicated.

It ensures that certain class attributes (specified by the to_propagate attribute of the metaclass) get propagated down the inheritance hierarchy. The assumption is that the values of the attributes specified in the class are iterable; if that is not met, Bad Things (tm) will happen.

This also provides virtual method implementation, similar to those in C-derived OO languages, and alias specifications.

global_virtuals = ['initialize', 'finalize']

to_propagate = [('parameters', <class 'wlauto.core.extension.Param'>, <class 'wlauto.c

virtual_methods = ['validate', 'initialize', 'finalize']

class wlauto.core.extension.**ListCollection**(*attrcls*, *owner*)

Bases: list

class wlauto.core.extension.**Module**(*owner*, ***kwargs*)

Bases: *wlauto.core.extension.Extension*

This is a "plugin" for an extension this is intended to capture functionality that may be optional for an extension, and so may or may not be present in a particular setup; or, conversely, functionality that may be reusable between multiple devices, even if they are not with the same inheritance hierarchy.

In other words, a Module is roughly equivalent to a kernel module and its primary purpose is to implement WA "drivers" for various peripherals that may or may not be present in a particular setup.

Note: A mudule is itself an Extension and can therefore have its own modules.

---

aliases = AC([])

artifacts = AC([])

capabilities = []

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

root_owner

validate(**args*, ***kwargs*)

class wlauto.core.extension.**Param**(*name*, *kind=None*, *mandatory=None*, *default=None*, *override=False*, *allowed_values=None*, *description=None*, *constraint=None*, *global_alias=None*, *convert_types=True*)

Bases: object

This is a generic parameter for an extension. Extensions instantiate this to declare which parameters are supported.

get_type_name()

kind_map = {<type 'bool'>: <function boolean>, <type 'int'>: <function integer>}

set_value(*obj*, *value=None*)

wlauto.core.extension.**Parameter**
alias of *Param*

wlauto.core.extension_loader module

class wlauto.core.extension_loader.**ExtensionLoader**(*packages=None*, *paths=None*, *ignore_paths=None*, *keep_going=False*, *load_defaults=True*)

Bases: object

Discovers, enumerates and loads available devices, configs, etc. The loader will attempt to discover things on construction by looking in predetermined set of locations defined by default_paths. Optionally, additional locations may specified through paths parameter that must be a list of additional Python module paths (i.e. dot-delimited).

clear()
Clear all discovered items.

get_default_config(*ext_name*)
Returns the default configuration for the specified extension name. The name may be an alias, in which case, the returned config will be augmented with appropriate alias overrides.

get_extension(*name*, **args*, ***kwargs*)
Return extension of the specified kind with the specified name. Any additional parameters will be passed to the extension's __init__.

get_extension_class(*name*, *kind=None*)
Return the class for the specified extension if found or raises ValueError.

get_load_defaults()

has_extension(*name*, *kind=None*)
Returns True if an extensions with the specified name has been discovered by the loader. If kind was specified, only returns True if the extension has been found, *and* it is of the specified kind.

list_extensions(*kind=None*)
List discovered extension classes. Optionally, only list extensions of a particular type.

load_defaults

reload()
Clear all discovered items and re-run the discovery.

resolve_alias(*alias_name*)
Try to resolve the specified name as an extension alias. Returns a two-tuple, the first value of which is actual extension name, and the second is a dict of parameter values for this alias. If the name passed is already an extension name, then the result is `(alias_name, {})`.

set_load_defaults(*value*)

update(*packages=None*, *paths=None*, *ignore_paths=None*)
Load extensions from the specified paths/packages without clearing or reloading existing extension.

class wlauto.core.extension_loader.**ExtensionLoaderItem**(*ext_tuple*)
Bases: `object`

class wlauto.core.extension_loader.**GlobalParameterAlias**(*name*)
Bases: `object`

Represents a "global alias" for an extension parameter. A global alias is specified at the top-level of config rather namespaced under an extension name.

Multiple extensions may have parameters with the same global_alias if they are part of the same inheritance hierarchy and one parameter is an override of the other. This class keeps track of all such cases in its extensions dict.

get_param(*ext*)

iteritems()

update(*other_ext*)

wlauto.core.exttype module

wlauto.core.exttype.**get_extension_type**(*ext*)
Given an instance of `wlauto.core.Extension`, return a string representing the type of the extension (e.g. `'workload'` for a Workload subclass instance).

wlauto.core.instrumentation module

Adding New Instrument

Any new instrument should be a subclass of Instrument and it must have a name. When a new instrument is added to Workload Automation, the methods of the new instrument will be found automatically and hooked up to the supported signals. Once a signal is broadcasted, the corresponding registered method is invoked.

Each method in Instrument must take two arguments, which are self and context. Supported signals can be found in [... link to signals ...] To make implementations easier and common, the basic steps to add new instrument is similar to the steps to add new workload.

Hence, the following methods are sufficient to implement to add new instrument:

- **setup: This method is invoked after the workload is setup. All the** necessary setups should go inside this method. Setup, includes operations like, pushing the files to the target device, install them, clear logs, etc.
- **start: It is invoked just before the workload start execution. Here is** where instrument measures start being registered/taken.
- **stop: It is invoked just after the workload execution stops. The measures** should stop being taken/registered.

- **update_result: It is invoked after the workload updated its result.** update_result is where the taken measures are added to the result so it can be processed by Workload Automation.
- **teardown is invoked after the workload is teared down. It is a good place** to clean any logs generated by the instrument.

For example, to add an instrument which will trace device errors, we subclass Instrument and overwrite the variable name.:

```
#BINARY_FILE = os.path.join(os.path.dirname(__file__), 'trace')
class TraceErrorsInstrument(Instrument):

    name = 'trace-errors'

    def __init__(self, device):
        super(TraceErrorsInstrument, self).__init__(device)
        self.trace_on_device = os.path.join(self.device.working_directory, 'trace')
```

We then declare and implement the aforementioned methods. For the setup method, we want to push the file to the target device and then change the file mode to 755

```
def setup(self, context):
    self.device.push_file(BINARY_FILE, self.device.working_directory)
    self.device.execute('chmod 755 {}'.format(self.trace_on_device))
```

Then we implemented the start method, which will simply run the file to start tracing.

```
def start(self, context):
    self.device.execute('{} start'.format(self.trace_on_device))
```

Lastly, we need to stop tracing once the workload stops and this happens in the stop method:

```
def stop(self, context):
    self.device.execute('{} stop'.format(self.trace_on_device))
```

The generated result can be updated inside update_result, or if it is trace, we just pull the file to the host device. context has a result variable which has add_metric method. It can be used to add the instrumentation results metrics to the final result for the workload. The method can be passed 4 params, which are metric key, value, unit and lower_is_better, which is a boolean.

```
def update_result(self, context):
    # pull the trace file to the device
    result = os.path.join(self.device.working_directory, 'trace.txt')
    self.device.pull_file(result, context.working_directory)

    # parse the file if needs to be parsed, or add result to
    # context.result
```

At the end, we might want to delete any files generated by the instrumentation and the code to clear these file goes in teardown method.

```
def teardown(self, context):
    self.device.delete_file(os.path.join(self.device.working_directory, 'trace.txt'))
```

class wlauto.core.instrumentation.**Instrument**(*device*, ***kwargs*)

Bases: *wlauto.core.extension.Extension*

Base class for instrumentation implementations.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.core.instrumentation.**ManagedCallback**(*instrument*, *callback*)
Bases: object

This wraps instruments' callbacks to ensure that errors do interfer with run execution.

wlauto.core.instrumentation.**check_failures**()

wlauto.core.instrumentation.**disable**(*to_disable*)

wlauto.core.instrumentation.**disable_all**()

wlauto.core.instrumentation.**enable**(*to_enable*)

wlauto.core.instrumentation.**enable_all**()

wlauto.core.instrumentation.**get_disabled**()

wlauto.core.instrumentation.**get_enabled**()

wlauto.core.instrumentation.**get_instrument**(*inst*)

wlauto.core.instrumentation.**install**(*instrument*)
This will look for methods (or any callable members) with specific names in the instrument and hook them up to the corresponding signals.

Parameters **instrument** – Instrument instance to install.

wlauto.core.instrumentation.**is_enabled**(*instrument*)

wlauto.core.instrumentation.**is_installed**(*instrument*)

wlauto.core.instrumentation.**reset_failures**()

wlauto.core.instrumentation.**uninstall**(*instrument*)

wlauto.core.instrumentation.**validate**()

wlauto.core.resolver module

Defines infrastructure for resource resolution. This is used to find various dependencies/assets/etc that WA objects rely on in a flexible way.

class wlauto.core.resolver.**ResourceResolver**(*config*)
Bases: object

Discovers and registers getters, and then handles requests for resources using registered getters.

get(*resource*, *strict=True*, **args*, ***kwargs*)
Uses registered getters to attempt to discover a resource of the specified kind and matching the specified criteria. Returns path to the resource that has been discovered. If a resource has not been discovered, this will raise a ResourceError or, if strict has been set to False, will return None.

load()
Discover getters under the specified source. The source could be either a python package/module or a path.

register(*getter*, *kind*, *priority=0*)
Register the specified resource getter as being able to discover a resource of the specified kind with the specified priority.

This method would typically be invoked by a getter inside its __init__. The idea being that getters register themselves for resources they know they can discover.

priorities

getters that are registered with the highest priority will be invoked first. If multiple getters are registered under the same priority, they will be invoked in the order they were registered (i.e. in the order they were discovered). This is essentially non-deterministic.

Generally getters that are more likely to find a resource, or would find a "better" version of the resource should register with higher (positive) priorities. Fall-back getters that should only be invoked if a resource is not found by usual means should register with lower (negative) priorities.

unregister(*getter*, *kind*)
Unregister a getter that has been registered earlier.

wlauto.core.resource module

class wlauto.core.resource.**GetterPriority**
Bases: object

Enumerates standard ResourceGetter priorities. In general, getters should register under one of these, rather than specifying other priority values.

Cached The cached version of the resource. Look here first. This priority also implies that the resource at this location is a "cache" and is not the only version of the resource, so it may be cleared without losing access to the resource.

Preferred Take this resource in favour of the environment resource.

Environment Found somewhere under ~/.workload_automation/ or equivalent, or from environment variables, external configuration files, etc. These will override resource supplied with the package.

External_package Resource provided by another package.

Package Resource provided with the package.

Remote Resource will be downloaded from a remote location (such as an HTTP server or a samba share). Try this only if no other getter was successful.

cached = 20

environment = 0

external_package = -5

package = -10

preferred = 10

remote = -4

class wlauto.core.resource.**Resource**(*owner*)
Bases: object

Represents a resource that needs to be resolved. This can be pretty much anything: a file, environment variable, a Python object, etc. The only thing a resource *has* to have is an owner (which would normally be the Workload/Instrument/Device/etc object that needs the resource). In addition, a resource have any number of attributes to identify, but all of them are resource type specific.

delete(*instance*)
Delete an instance of this resource type. This must be implemented by the concrete subclasses based on what the resource looks like, e.g. deleting a file or a directory tree, or removing an entry from a database.

Note Implementation should *not* contain any logic for deciding whether or not a resource should be deleted, only the actual deletion. The assumption is that if this method is invoked, then the decision has already been made.

name = None

class wlauto.core.resource.**ResourceGetter**(*resolver*, ***kwargs*)
Bases: *wlauto.core.extension.Extension*

Base class for implementing resolvers. Defines resolver interface. Resolvers are responsible for discovering resources (such as particular kinds of files) they know about based on the parameters that are passed to them. Each resolver also has a dict of attributes that describe its operation, and may be used to determine which get invoked. There is no pre-defined set of attributes and resolvers may define their own.

Class attributes:

Name Name that uniquely identifies this getter. Must be set by any concrete subclass.

Resource_type Identifies resource type(s) that this getter can handle. This must be either a string (for a single type) or a list of strings for multiple resource types. This must be set by any concrete subclass.

Priority Priority with which this getter will be invoked. This should be one of the standard priorities specified in `GetterPriority` enumeration. If not set, this will default to `GetterPriority.environment`.

aliases = AC([])

artifacts = AC([])

core_modules = []

delete(*resource*, **args*, ***kwargs*)
Delete the resource if it is discovered. All arguments are passed to a call to‘‘self.get()‘‘. If that call returns a resource, it is deleted.

Returns `True` if the specified resource has been discovered and deleted, and `False` otherwise.

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)
This will get invoked by the resolver when attempting to resolve a resource, passing in the resource to be resolved as the first parameter. Any additional parameters would be specific to a particular resource type.

This method will only be invoked for resource types that the getter has registered for.

Parameters **resource** – an instance of *wlauto.core.resource.Resource*.

Returns Implementations of this method must return either the discovered resource or `None` if the resource could not be discovered.

initialize(**args*, ***kwargs*)

name = None

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = 0

register()
Registers with a resource resolver. Concrete implementations must override this to invoke `self.resolver.register()` method to register `self` for specific resource types.

resource_type = None

unregister()
Unregister from a resource resolver.

validate(**args*, ***kwargs*)

wlauto.core.result module

This module defines the classes used to handle result processing inside Workload Automation. There will be a `wlauto.core.workload.WorkloadResult` object generated for every workload iteration executed. This object will have a list of `wlauto.core.workload.WorkloadMetric` objects. This list will be populated by the workload itself and may also be updated by instrumentation (e.g. to add power measurements). Once the result object has been fully populated, it will be passed into the `process_iteration_result` method of *ResultProcessor*. Once the entire run has completed, a list containing result objects from all iterations will be passed into `process_results` method of :class'ResultProcessor'.

Which result processors will be active is defined by the `result_processors` list in the `~/.workload_automation/config.py`. Only the result_processors who's names appear in this list will be used.

A `ResultsManager` keeps track of active results processors.

class wlauto.core.result.**IterationResult**(*spec*)
Bases: `object`

Contains the result of running a single iteration of a workload. It is the responsibility of a workload to instantiate a IterationResult, populate it, and return it form its get_result() method.

Status explanations:

NOT_STARTED This iteration has not yet started.

RUNNING This iteration is currently running and no errors have been detected.

OK This iteration has completed and no errors have been detected

PARTIAL One or more instruments have failed (the iteration may still be running).

FAILED The workload itself has failed.

ABORTED The user interupted the workload

SKIPPED The iteration was skipped due to a previous failure

ABORTED = 'ABORTED'

FAILED = 'FAILED'

NONCRITICAL = 'NONCRITICAL'

NOT_STARTED = 'NOT_STARTED'

OK = 'OK'

PARTIAL = 'PARTIAL'

RUNNING = 'RUNNING'

SKIPPED = 'SKIPPED'

add_event(*message*)

add_metric(*name*, *value*, *units=None*, *lower_is_better=False*, *classifiers=None*)

has_metric(*name*)

to_dict()

values = ['NOT_STARTED', 'RUNNING', 'OK', 'NONCRITICAL', 'PARTIAL', 'FAILED', 'ABORTED

class wlauto.core.result.**Metric**(*name*, *value*, *units=None*, *lower_is_better=False*, *classifiers=None*)

Bases: object

This is a single metric collected from executing a workload.

Parameters

- **name** – the name of the metric. Uniquely identifies the metric within the results.
- **value** – The numerical value of the metric for this execution of a workload. This can be either an int or a float.
- **units** – Units for the collected value. Can be None if the value has no units (e.g. it's a count or a standardised score).
- **lower_is_better** – Boolean flag indicating where lower values are better than higher ones. Defaults to False.
- **classifiers** – A set of key-value pairs to further classify this metric beyond current iteration (e.g. this can be used to identify sub-tests).

to_dict()

class wlauto.core.result.**ResultManager**

Bases: object

Keeps track of result processors and passes on the results onto the individual processors.

add_result(*result*, *context*)

finalize(*context*)

initialize(*context*)

install(*processor*)

process_run_result(*result*, *context*)

uninstall(*processor*)

validate()

class wlauto.core.result.**ResultProcessor**(***kwargs*)

Bases: *wlauto.core.extension.Extension*

Base class for result processors. Defines an interface that should be implemented by the subclasses. A result processor can be used to do any kind of post-processing of the results, from writing them out to a file, to uploading them to a database, performing calculations, generating plots, etc.

aliases = AC([])

artifacts = AC([])

core_modules = []

export_iteration_result(*result*, *context*)

export_run_result(*result*, *context*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.core.result.**RunEvent**(*message*)
Bases: object

An event that occured during a run.

to_dict()

class wlauto.core.result.**RunResult**(*run_info*, *output_directory=None*)
Bases: object

Contains overall results for a run.

FAILED = 'FAILED'

OK = 'OK'

OKISH = 'OKISH'

PARTIAL = 'PARTIAL'

UNKNOWN = 'UNKNOWN'

status

values = ['OK', 'OKISH', 'PARTIAL', 'FAILED', 'UNKNOWN']

wlauto.core.signal module

This module wraps louie signalling mechanism. It relies on modified version of loiue that has prioritization added to handler invocation.

class wlauto.core.signal.**Signal**(*name*, *invert_priority=False*)
Bases: object

This class implements the signals to be used for notifiying callbacks registered to respond to different states and stages of the execution of workload automation.

wlauto.core.signal.**connect**(*handler*, *signal*, *sender=<class 'louie.sender.Any'>*, *priority=0*)
Connects a callback to a signal, so that the callback will be automatically invoked when that signal is sent.

Parameters:

handler This can be any callable that that takes the right arguments for the signal. For most siginals this means a single argument that will be an ExecutionContext instance. But please see documentaion for individual signals in the *signals reference*.

signal The signal to which the hanlder will be subscribed. Please see *signals reference* for the list of standard WA signals.

> **Note:** There is nothing that prevents instrumentation from sending their own signals that are not part of the standard set. However the signal must always be an *wlauto.core.signal.Signal* instance.

sender The handler will be invoked only for the signals emitted by this sender. By default, this is set to `louie.dispatcher.Any`, so the handler will be invoked for signals from any sentder.

priority An integer (positive or negative) the specifies the priority of the handler. Handlers with higher priority will be called before handlers with lower priority. The call order of handlers with the same priority is not specified. Defaults to 0.

> **Note:** Priorities for some signals are inverted (so highest priority handlers get executed last). Please see *signals reference* for details.

`wlauto.core.signal.`**`disconnect`**(*handler*, *signal*, *sender=<class 'louie.sender.Any'>*)

Disconnect a previously connected handler form the specified signal, optionally, only for the specified sender.

Parameters:

handler The callback to be disconnected.

signal The signal the handler is to be disconnected form. It will be an *wlauto.core.signal.Signal* instance.

sender If specified, the handler will only be disconnected from the signal sent by this sender.

`wlauto.core.signal.`**`send`**(*signal*, *sender*, **args*, ***kwargs*)

Sends a signal, causing connected handlers to be invoked.

Paramters:

signal Signal to be sent. This must be an instance of *wlauto.core.signal.Signal* or its subclasses.

sender The sender of the signal (typically, this would be `self`). Some handlers may only be subscribed to signals from a particular sender.

The rest of the parameters will be passed on as aruments to the handler.

wlauto.core.version module

`wlauto.core.version.`**`VersionTuple`**

alias of `Version`

`wlauto.core.version.`**`get_wa_version`**()

wlauto.core.workload module

A workload is the unit of execution. It represents a set of activities are are performed and measured together, as well as the necessary setup and teardown procedures. A single execution of a workload produces one `wlauto.core.result.WorkloadResult` that is populated with zero or more `wlauto.core.result.WorkloadMetrics` and/or `wlauto.core.result.Artifacts` by the workload and active instrumentation.

class wlauto.core.workload.**Workload**(*device*, ***kwargs*)
Bases: *wlauto.core.extension.Extension*

This is the base class for the workloads executed by the framework. Each of the methods throwing NotImplementedError *must* be implemented by the derived classes.

aliases = AC([])

artifacts = AC([])

check_network_connected()

core_modules = []

finalize(**args*, ***kwargs*)

init_resources(*context*)
This method may be used to perform early resource discovery and initialization. This is invoked during the initial loading stage and before the device is ready, so cannot be used for any device-dependent initialization. This method is invoked before the workload instance is validated.

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = False

run(*context*)
Execute the workload. This is the method that performs the actual "work" of the

setup(*context*)
Perform the setup necessary to run the workload, such as copying the necessary files to the device, configuring the environments, etc.

This is also the place to perform any on-device checks prior to attempting to execute the workload.

summary_metrics = []

supported_devices = []

supported_platforms = []

teardown(*context*)
Perform any final clean up for the Workload.

update_result(*context*)
Update the result within the specified execution context with the metrics form this workload iteration.

validate(**args*, ***kwargs*)

Module contents

wlauto.devices package

Subpackages

wlauto.devices.android package

Subpackages

wlauto.devices.android.gem5 package

Module contents

class wlauto.devices.android.gem5.**Gem5AndroidDevice**(***kwargs*)

Bases: *wlauto.common.gem5.device.BaseGem5Device*, *wlauto.common.android.device.AndroidDevice*

Implements gem5 Android device.

This class allows a user to connect WA to a simulation using gem5. The connection to the device is made using the telnet connection of the simulator, and is used for all commands. The simulator does not have ADB support, and therefore we need to fall back to using standard shell commands.

Files are copied into the simulation using a VirtIO 9P device in gem5. Files are copied out of the simulated environment using the m5 writefile command within the simulated system.

When starting the workload run, the simulator is automatically started by Workload Automation, and a connection to the simulator is established. WA will then wait for Android to boot on the simulated system (which can take hours), prior to executing any other commands on the device. It is also possible to resume from a checkpoint when starting the simulation. To do this, please append the relevant checkpoint commands from the gem5 simulation script to the gem5_discription argument in the agenda.

Host system requirements:

- VirtIO support. We rely on diod on the host system. This can be installed on ubuntu using the following command:

  sudo apt-get install diod

Guest requirements:

- VirtIO support. We rely on VirtIO to move files into the simulation. Please make sure that the following are set in the kernel configuration:

  CONFIG_NET_9P=y

  CONFIG_NET_9P_VIRTIO=y

  CONFIG_9P_FS=y

  CONFIG_9P_FS_POSIX_ACL=y

  CONFIG_9P_FS_SECURITY=y

  CONFIG_VIRTIO_BLK=y

- m5 binary. Please make sure that the m5 binary is on the device and can by found in the path.

aliases = AC([])

artifacts = AC([])

capture_screen(*filepath*)

clear_logcat()
Clear (flush) logcat log.

core_modules = []

disable_selinux()
Disable SELinux. Overridden as parent implementation uses ADB

dump_logcat(*outfile*, *filter_spec=None*)
Extract logcat from simulation

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

get_properties(*context*)
Get the property files from the device

initialize(**args*, ***kwargs*)

install(*filepath*, *timeout=10800*)
Install an APK or a normal executable

install_apk(*filepath*, *timeout=10800*)
Install an APK on the gem5 device

The APK is pushed to the device. Then the file and folder permissions are changed to ensure that the APK can be correctly installed. The APK is then installed on the device using 'pm'.

install_executable(*filepath*, *with_name=None*)
Install an executable

kind = 'device'

login_to_device()

name = 'gem5_android'

parameters = AC(["Param({'kind': <type 'str'>, 'mandatory': False, 'name': 'gem5_bi

platform = 'android'

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

uninstall(*package*)

validate(**args*, ***kwargs*)

wait_for_boot()
Wait for the system to boot

We monitor the sys.boot_completed and service.bootanim.exit system properties to determine when the system has finished booting. In the event that we cannot coerce the result of service.bootanim.exit to an integer, we assume that the boot animation was disabled and do not wait for it to finish.

wlauto.devices.android.generic package

Module contents

class wlauto.devices.android.generic.**GenericDevice**(***kwargs*)
Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

default_working_directory = '/storage/sdcard0/working'

description = '\n A generic Android device interface. Use this if you do not have an i

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'generic_android'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.juno package

Module contents

class wlauto.devices.android.juno.**Juno**(***kwargs*)
Bases: *wlauto.common.android.device.BigLittleDevice*

aliases = AC([])

artifacts = AC([])

boot(*hard=False*, ***kwargs*)

capabilities = ['reset_power']

connect()

core_modules = ['vexpress']

description = '\n ARM Juno next generation big.LITTLE development platform.\n '

disconnect()

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

firmware_prompt = 'Cmd>'

get_android_id()

hard_reset()

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'juno'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

reset()

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

short_delay = 1

validate(**args*, ***kwargs*)

wait_for_microsd_mount_point(*target*, *timeout=100*)

wlauto.devices.android.meizumx6 package

Module contents

class wlauto.devices.android.meizumx6.**MeizuMX6**(***kwargs*)

Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

is_rooted

kind = 'device'

name = 'meizumx6'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.nexus10 package

Module contents

class wlauto.devices.android.nexus10.**Nexus10Device**(***kwargs*)

Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

default_working_directory = '/sdcard/working'

description = '\n Nexus10 is a 10 inch tablet device, which has dual-core A15.\n\n To

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

max_cores = 2

name = 'Nexus10'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.nexus5 package

Module contents

class wlauto.devices.android.nexus5.**Nexus5Device**(***kwargs*)

Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

default_working_directory = '/storage/sdcard0/working'

description = '\n Adapter for Nexus 5.\n\n To be able to use Nexus5 in WA, the followi

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

max_cores = 4

name = 'Nexus5'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.note3 package

Module contents

class wlauto.devices.android.note3.**Note3Device**(***kwargs*)
Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

connect()

core_modules = []

description = '\n Adapter for Galaxy Note 3.\n\n To be able to use Note3 in WA, the fo

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

hard_reset()

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'Note3'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

reset()

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.odroidxu3 package

Module contents

class wlauto.devices.android.odroidxu3.**OdroidXU3**(***kwargs*)
Bases: *wlauto.common.android.device.AndroidDevice*

aliases = AC([])

artifacts = AC([])

core_modules = ['odroidxu3-fan']

description = 'HardKernel Odroid XU3 development board.'

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'odroidxu3'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.android.tc2 package

Module contents

class wlauto.devices.android.tc2.**TC2Device**(***kwargs*)

Bases: *wlauto.common.android.device.BigLittleDevice*

a15_governor_tunables

a15_only_modes = ['mp_a15_only', 'iks_a15']

a7_governor_tunables

a7_only_modes = ['mp_a7_only', 'iks_a7', 'iks_cpu']

aliases = AC([])

artifacts = AC([])

boot(***kwargs*)

connect()

core_clusters

core_modules = []

core_names

cpu_cores

description = "\n TC2 is a development board, which has three A7 cores and two A15 cor

disable_idle_states()

Disable idle states on TC2. See http://wiki.arm.com/Research/TC2SetupAndUsage ("Enabling Idle Modes" section) and http://wiki.arm.com/ASD/ControllingPowerManagementInLinaroKernels

disconnect()

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

enable_idle_states()

Fully enables idle states on TC2. See http://wiki.arm.com/Research/TC2SetupAndUsage ("Enabling Idle Modes" section) and http://wiki.arm.com/ASD/ControllingPowerManagementInLinaroKernels

finalize(**args*, ***kwargs*)

get_cpuidle()

get_mode()

has_gpu = False

initialize(**args*, ***kwargs*)

kind = 'device'

max_a15_cores

max_a7_cores

mode

name = 'TC2'

not_configurable_modes = ['iks_a7', 'iks_cpu', 'iks_a15']

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

set_irq_affinity(*cluster*)

Set's IRQ affinity to the specified cluster.

This method will only work if the device mode is mp_a7_bootcluster or mp_a15_bootcluster. This operation does not make sense if there is only one cluster active (all IRQs will obviously go to that), and it will not work for IKS kernel because clusters are not exposed to sysfs.

Parameters **cluster** – must be either 'a15' or 'a7'.

set_mode(*mode*)

validate(**args*, ***kwargs*)

Module contents

wlauto.devices.linux package

Subpackages

wlauto.devices.linux.XE503C12 package

Module contents

class wlauto.devices.linux.XE503C12.**Xe503c12Chormebook**(**args*, ***kwargs*)

Bases: *wlauto.common.linux.device.LinuxDevice*

abi = 'armeabi'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'A developer-unlocked Samsung XE503C12 running sshd.'

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'XE503C12'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

platform = 'chromeos'

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.linux.chromeos_test_image package

Module contents

class wlauto.devices.linux.chromeos_test_image.**ChromeOsDevice**(***kwargs*)

Bases: *wlauto.common.linux.device.LinuxDevice*

abi = 'armeabi'

aliases = AC([])

artifacts = AC([])

core_modules = []

default_timeout = 100

description = '\n Chrome OS test image device. Use this if you are working on a Chrome

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

get_ui_status()

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'chromeos_test_image'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

platform = 'chromeos'

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

set_ui_status(*status*)

stop()

validate(**args*, ***kwargs*)

wlauto.devices.linux.gem5 package

Module contents

class wlauto.devices.linux.gem5.**Gem5LinuxDevice**(***kwargs*)

Bases: *wlauto.common.gem5.device.BaseGem5Device*, *wlauto.common.linux.device.LinuxDevice*

Implements gem5 Linux device.

This class allows a user to connect WA to a simulation using gem5. The connection to the device is made using the telnet connection of the simulator, and is used for all commands. The simulator does not have ADB support, and therefore we need to fall back to using standard shell commands.

Files are copied into the simulation using a VirtIO 9P device in gem5. Files are copied out of the simulated environment using the m5 writefile command within the simulated system.

When starting the workload run, the simulator is automatically started by Workload Automation, and a connection to the simulator is established. WA will then wait for Android to boot on the simulated system (which can take hours), prior to executing any other commands on the device. It is also possible to resume from a checkpoint when starting the simulation. To do this, please append the relevant checkpoint commands from the gem5 simulation script to the gem5_discription argument in the agenda.

Host system requirements:

- VirtIO support. We rely on diod on the host system. This can be installed on ubuntu using the following command:

  sudo apt-get install diod

Guest requirements:

- VirtIO support. We rely on VirtIO to move files into the simulation. Please make sure that the following are set in the kernel configuration:

  CONFIG_NET_9P=y

  CONFIG_NET_9P_VIRTIO=y

  CONFIG_9P_FS=y

  CONFIG_9P_FS_POSIX_ACL=y

  CONFIG_9P_FS_SECURITY=y

  CONFIG_VIRTIO_BLK=y

- m5 binary. Please make sure that the m5 binary is on the device and can by found in the path.

aliases = AC([])

artifacts = AC([])

capture_screen(*filepath*)

core_modules = []

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'device'

login_to_device()

name = 'gem5_linux'

parameters = AC(["Param({'kind': <type 'str'>, 'mandatory': False, 'name': 'gem5_bi

platform = 'linux'

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.linux.generic package

Module contents

class wlauto.devices.linux.generic.**GenericDevice**(**args*, ***kwargs*)

Bases: *wlauto.common.linux.device.LinuxDevice*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n A generic Linux device interface. Use this if you do not have an int

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

has_gpu = True

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'generic_linux'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

wlauto.devices.linux.odroidxu3_linux package

Module contents

class wlauto.devices.linux.odroidxu3_linux.**OdroidXU3LinuxDevice**(**args*, ***kwargs*)

Bases: *wlauto.common.linux.device.LinuxDevice*

abi = 'armeabi'

aliases = AC([])

artifacts = AC([])

core_modules = ['odroidxu3-fan']

description = 'HardKernel Odroid XU3 development board (Ubuntu image).'

dynamic_modules = AC(["{'devcpufreq': {}}", "{'cpuidle': {}}"])

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'device'

name = 'odroidxu3_linux'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

runtime_parameters = AC(['sysfile_values', '${core}_cores', '${core}_min_frequency', '

validate(**args*, ***kwargs*)

Module contents

Module contents

wlauto.instrumentation package

Subpackages

wlauto.instrumentation.acmecape package

Module contents

class wlauto.instrumentation.acmecape.**AcmeCapeInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Instrumetnation for the BayLibre ACME cape for power/energy measurme

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'acmecape'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

very_fast_start(*context*)

very_fast_stop(*context*)

wlauto.instrumentation.coreutil package

Module contents

class wlauto.instrumentation.coreutil.**Calculator**(*cores*, *threshold*, *context*)

Bases: object

Read /proc/stat and dump data into `proc.txt` which is parsed to generate `coreutil.csv` Sample output from 'proc.txt'

```
------------------------------------------------------------------
cpu  9853753 51448 3248855 12403398 4241 111 14996 0 0 0
cpu0 1585220 7756 1103883 4977224 552 97 10505 0 0 0
cpu1 2141168 7243 564347 972273 504 4 1442 0 0 0
cpu2 1940681 7994 651946 1005534 657 3 1424 0 0 0
cpu3 1918013 8833 667782 1012249 643 3 1326 0 0 0
```

```
cpu4 165429 5363 50289 1118910 474 0 148 0 0 0
cpu5 1661299 4910 126654 1104018 480 0 53 0 0 0
cpu6 333642 4657 48296 1102531 482 2 55 0 0 0
cpu7 108299 4691 35656 1110658 448 0 41 0 0 0
------------------------------------------------------------------------
Description:

1st column  : cpu_id( cpu0, cpu1, cpu2,......)
Next all column represents the amount of time, measured in units of USER_HZ
2nd column  : Time spent in user mode
3rd column  : Time spent in user mode with low priority
4th column  : Time spent in system mode
5th column  : Time spent in idle task
6th column  : Time waiting for i/o to compelete
7th column  : Time servicing interrupts
8th column  : Time servicing softirqs
9th column  : Stolen time is the time spent in other operating systems
10th column : Time spent running a virtual CPU
11th column : Time spent running a niced guest

------------------------------------------------------------------------
```

Procedure to calculate instantaneous CPU utilization:

1. Subtract two consecutive samples for every column(except 1st)
2. Sum all the values except “Time spent in idle task”
3. CPU utilization(%) = (value obtained in 2)/sum of all the values)*100

calculate()

calculate_core_utilization()
Calculates CPU utilization

calculate_total_active()
Read proc.txt file and calculate ‘self.active’ and ‘self.total’

generate_csv(*context*)
generates `coreutil.csv`

idle_time_index = 3

class wlauto.instrumentation.coreutil.**CoreUtilization**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Measures CPU core activity during workload execution in terms of the

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'coreutil'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)
Calls ProcCollect class

start(*context*)
Starts collecting data once the workload starts

stop(*context*)
Stops collecting data once the workload stops

update_result(*context*)
updates result into coreutil.csv

validate(**args*, ***kwargs*)

class wlauto.instrumentation.coreutil.**ProcCollect**(*device*, *logger*, *out_dir*)
Bases: threading.Thread

Dumps data into proc.txt

run()

stop()
Executed once the workload stops

wlauto.instrumentation.daq package

Module contents

class wlauto.instrumentation.daq.**Daq**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

before_overall_results_processing(*context*)

core_modules = []

description = "\n DAQ instrument obtains the power consumption of the target device's

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

insert_start_marker(*context*)

insert_stop_marker(*context*)

kind = 'instrument'

name = 'daq'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

slow_start(*context*)

slow_stop(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.daq.**dict_or_bool**(*value*)
Ensures that either a dictionary or a boolean is used as a parameter.

wlauto.instrumentation.delay package

Module contents

class wlauto.instrumentation.delay.**DelayInstrument**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n This instrument introduces a delay before executing either an iterat

do_wait_for_temperature(*temperature*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'delay'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

very_slow_on_iteration_start(*context*)

very_slow_on_spec_start(*context*)

very_slow_start(*context*)

wait_for_temperature(*temperature*)

wlauto.instrumentation.dmesg package

Module contents

class wlauto.instrumentation.dmesg.**DmesgInstrument**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

Collected dmesg output before and during the run.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

loglevel_file = '/proc/sys/kernel/printk'

name = 'dmesg'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

slow_start(*context*)

slow_stop(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.energy_model package

Module contents

class wlauto.instrumentation.energy_model.**CapPowerState**(*cap*, *power*)

Bases: tuple

cap

Alias for field number 0

power

Alias for field number 1

class wlauto.instrumentation.energy_model.**EnergyModel**

Bases: object

add_cap_entry(*cluster*, *perf*, *clust_pow*, *core_pow*)

add_cluster_idle(*cluster*, *values*)

add_core_idle(*cluster*, *values*)

class wlauto.instrumentation.energy_model.**EnergyModelInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

before_overall_results_processing(*context*)

configure_clusters()

core_modules = []

desicription = '\n Generates a power mode for the device based on specified workload.

disable_thermal_management()

discover_idle_states()

enable_all_cores()

enable_all_idle_states()

fast_start(*context*)

fast_stop(*context*)

finalize(**args*, ***kwargs*)

get_cluster_specs(*old_specs*, *cluster*, *context*)

get_core_name(*cluster*)

get_cpus(*cluster*)

get_device_idle_states(*cluster*)

get_frequencies_param(*cluster*)

initialize(**args*, ***kwargs*)

initialize_job_queue(*context*)

initialize_result_tracking()

kind = 'instrument'

name = 'energy_model'

on_iteration_start(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

perform_runtime_validation()

reset()

reset_cgroups()

setup(*context*)

setup_measurement(*measured*)

slow_update_result(*context*)

thermal_correction(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.energy_model.**IdlePowerState**(*power*)

Bases: tuple

power

Alias for field number 0

class wlauto.instrumentation.energy_model.**PowerPerformanceAnalysis**(*data*)

Bases: object

wlauto.instrumentation.energy_model.**build_energy_model**(*freq_power_table*, *cpus_power*, *idle_power*, *first_cluster_idle_state*)

wlauto.instrumentation.energy_model.**fit_polynomial**(*s*, *n*)

wlauto.instrumentation.energy_model.**generate_em_c_file**(*em*, *big_core*, *little_core*, *em_template_file*, *outfile*)

wlauto.instrumentation.energy_model.**generate_report**(*freq_power_table*, *measured_cpus_table*, *cpus_table*, *idle_power_table*, *report_template_file*, *device_name*, *em_text*, *outfile*)

wlauto.instrumentation.energy_model.**get_cap_power_plot**(*data_single_core*)

wlauto.instrumentation.energy_model.**get_cpus_power_table**(*data*, *index*, *opps*, *leak_factors*)

wlauto.instrumentation.energy_model.**get_figure_data**(*fig*, *fmt='png'*)

wlauto.instrumentation.energy_model.**get_idle_power_plot**(*df*)

wlauto.instrumentation.energy_model.**get_normalized_single_core_data**(*data*)

wlauto.instrumentation.energy_model.**opp_table**(*d*)

wlauto.instrumentation.energy_model.**plot_cpus_table**(*projected*, *ax*, *cluster*)

wlauto.instrumentation.energy_model.**wa_result_to_power_perf_table**(*df*, *performance_metric*, *index*)

wlauto.instrumentation.energy_probe package

Module contents

class wlauto.instrumentation.energy_probe.**EnergyProbe**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

MAX_CHANNELS = 3

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'Collects power traces using the ARM energy probe.\n\n This instrument r

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'energy_probe'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.fps package

Module contents

class wlauto.instrumentation.fps.**FpsInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures Frames Per Second (FPS) and associated metrics for a worklo

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'fps'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

slow_update_result(*context*)

start(*context*)

stop(*context*)

supported_platforms = ['android']

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.fps.**LatencyCollector**(*outfile*, *device*, *activities*, *keep_raw*, *logger*, *dumpsys_period*, *run_command*, *list_command*, *fps_method*)

Bases: threading.Thread

run()

stop()

wlauto.instrumentation.freqsweep package

Module contents

class wlauto.instrumentation.freqsweep.**FreqSweep**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Sweeps workloads through all available frequencies on a device.\n\n

finalize(**args*, ***kwargs*)

get_sweep_workload_specs(*old_specs*, *sweep_spec*, *context*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'freq_sweep'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.instrumentation.hwmon package

Module contents

class wlauto.instrumentation.hwmon.**HwmonInstrument**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Hardware Monitor (hwmon) is a generic Linux kernel subsystem,\n prov

fast_start(*context*)

fast_stop(*context*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'hwmon'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.juno_energy package

Module contents

class wlauto.instrumentation.juno_energy.**JunoEnergy**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Collects internal energy meter measurements from Juno development bo

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'juno_energy'

on_run_init(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

teardown(*conetext*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.misc package

Module contents

Some "standard" instruments to collect additional info about workload execution.

Note: The run() method of a Workload may perform some "boilerplate" as well as the actual execution of the workload (e.g. it may contain UI automation needed to start the workload). This "boilerplate" execution will also be measured by these instruments. As such, they are not suitable for collected precise data about specific operations.

---

class wlauto.instrumentation.misc.**DynamicFrequencyInstrument**(*device*, ***kwargs*)

Bases: *wlauto.instrumentation.misc.FsExtractor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Collects dynamic frequency (DVFS) settings before and after workload

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'cpufreq'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

tarname = 'cpufreq.tar'

validate(**args*, ***kwargs*)

class wlauto.instrumentation.misc.**ExecutionTimeInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measure how long it took to execute the run() methods of a Workload.'

finalize(**args*, ***kwargs*)

get_start_time(*context*)

get_stop_time(*context*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'execution_time'

on_run_start(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = 15

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.misc.**FsExtractor**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

AFTER_PATH = 2

BEFORE_PATH = 1

DEVICE_PATH = 0

DIFF_PATH = 3

aliases = AC([])

artifacts = AC([])

core_modules = []

extract_timeout = 30

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

initialize_tmpfs(*context*)

mount_command = 'mount -t tmpfs -o size={} tmpfs {}'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

slow_start(*context*)

slow_stop(*context*)

tarname = 'sysfs.tar'

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.misc.**InterruptStatsInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Pulls the ``/proc/interrupts`` file before and after workload execut

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'interrupts'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.misc.**SysfsExtractor**(*device*, ***kwargs*)

Bases: *wlauto.instrumentation.misc.FsExtractor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Collects the contest of a set of directories, before and after workl

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'sysfs_extractor'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.instrumentation.netstats package

Module contents

class wlauto.instrumentation.netstats.**NetstatsCollector**(*target*, *apk*, *service='.TrafficMetricsService'*)

Bases: `object`

get_data(*outfile*)

reset(*sites=None*, *period=None*)

setup(*force=False*)

start()

stop()

teardown()

class wlauto.instrumentation.netstats.**NetstatsInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures transmit/receive network traffic on an Android divice on pe

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'netstats'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.netstats.**extract_netstats**(*filepath*, *tag=None*)

wlauto.instrumentation.netstats.**netstats_to_measurements**(*netstats*)

wlauto.instrumentation.netstats.**write_measurements_csv**(*measurements*, *filepath*)

wlauto.instrumentation.perf package

Module contents

class wlauto.instrumentation.perf.**PerfInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Perf is a Linux profiling with performance counters.\n\n Performance

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'perf'

on_run_init(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.pmu_logger package

Module contents

class wlauto.instrumentation.pmu_logger.**CciPmuLogger**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n This instrument allows collecting CCI counter data.\n\n It relies on

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'cci_pmu_logger'

on_run_init(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.poller package

Module contents

class wlauto.instrumentation.poller.**FilePoller**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Polls the given files at a set sample interval. The values are outpu

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'file_poller'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

start(*context*)

stop(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.screenon package

Module contents

class wlauto.instrumentation.screenon.**ScreenMonitor**(*device*, *polling_period*)
Bases: threading.Thread

run()

stop()

class wlauto.instrumentation.screenon.**ScreenOnInstrument**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Ensure screen is on before each iteration on Android devices.\n\n A

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'screenon'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

slow_setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.servo_power_monitors package

Module contents

class wlauto.instrumentation.servo_power_monitors.**PowerPoller**(*host*, *port*, *channels*, *sampling_rate*)
Bases: threading.Thread

run()

stop()

class wlauto.instrumentation.servo_power_monitors.**ServoPowerMonitor**(*device*, ***kwargs*)
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Collects power traces using the Chromium OS Servo Board.\n\n Servo i

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'servo_power'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

servod_delay_between_tries = 0.1

servod_max_tries = 100

setup(*context*)

start(*context*)

stop(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.streamline package

Module contents

class wlauto.instrumentation.streamline.**StreamlineInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

configuration_file_name = 'configuration.xml'

core_modules = []

daemon = 'gatord'

description = '\n Collect Streamline traces from the device.\n\n .. note:: This instr

driver = 'gator.ko'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'streamline'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.instrumentation.streamline.**StreamlineResourceGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

dependencies_directory = '/home/docs/.workload_automation/dependencies/streamline'

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'streamline_resource'

old_dependencies_directory = '/home/docs/.workload_automation/streamline'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = 1

resource_type = 'file'

validate(**args*, ***kwargs*)

wlauto.instrumentation.systrace package

Module contents

class wlauto.instrumentation.systrace.**systrace**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n This instrument uses systrace.py from the android SDK to dump atrace

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'instrument'

name = 'systrace'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

start(*context*)

stop(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.instrumentation.trace_cmd package

Module contents

class wlauto.instrumentation.trace_cmd.**TraceCmdInstrument**(*device*, ***kwargs*)

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n trace-cmd is an instrument which interacts with ftrace Linux kernel

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

insert_end_mark(*context*)

insert_start_mark(*context*)

kind = 'instrument'

name = 'trace-cmd'

on_run_end(*context*)

on_run_init(*context*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

stop(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

very_slow_start(*context*)

Module contents

wlauto.instrumentation.**clear_instrumentation**()

wlauto.instrumentation.**instrument_is_enabled**(*instrument*)

Returns `True` if the specified instrument is installed and is currently enabled, and `False` other wise. The instrument maybe specified either as a name or a subclass (or instance of subclass) of `wlauto.core.Instrument`.

wlauto.instrumentation.**instrument_is_installed**(*instrument*)

Returns `True` if the specified instrument is installed, and `False` other wise. The instrument maybe specified either as a name or a subclass (or instance of subclass) of `wlauto.core.Instrument`.

wlauto.modules package

Submodules

wlauto.modules.active_cooling module

class wlauto.modules.active_cooling.**MbedFanActiveCooling**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['active_cooling']

core_modules = []

description = 'Controls a cooling fan via an mbed connected to a serial port.'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'mbed-fan'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

start_active_cooling()

stop_active_cooling()

timeout = 30

validate(**args*, ***kwargs*)

class wlauto.modules.active_cooling.**OdroidXU3ctiveCooling**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['active_cooling']

core_modules = []

description = '\n Enabled active cooling by controling the fan an Odroid XU3\n\n .. no

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'odroidxu3-fan'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

start_active_cooling()

stop_active_cooling()

validate(**args*, ***kwargs*)

wlauto.modules.cgroups module

class wlauto.modules.cgroups.**CgroupController**(*mount_name*)
Bases: object

kind = 'cpuset'

mount(*device*, *mount_root*)

class wlauto.modules.cgroups.**Cgroups**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['cgroups']

controllers = [<wlauto.modules.cgroups.CpusetController object>]

core_modules = []

description = '\n Adds cgroups query and manuputions APIs to a Device interface.\n\n Cu

finalize(**args*, ***kwargs*)

get_cgroup_controller(*kind*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'cgroups'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.modules.cgroups.**CpusetController**(**args*, ***kwargs*)
Bases: *wlauto.modules.cgroups.CgroupController*

create_group(*name*, *cpus*, *mems*)

mount(*device*, *mount_root*)

move_all_tasks_to(*target_group*)

move_tasks(*source*, *dest*)

class wlauto.modules.cgroups.**CpusetGroup**(*controller*, *name*, *cpus*, *mems*)
Bases: object

add_task(*tid*)

add_tasks(*tasks*)

get()

get_tasks()

set(*cpus*, *mems*)

wlauto.modules.cpufreq module

class wlauto.modules.cpufreq.**CpufreqModule**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['cpufreq']

core_modules = []

description = '\n cpufreq-related functionality module for the device. Query and set f

finalize(**args*, ***kwargs*)

get_cluster_active_cpu(*cluster*)
Returns the first *active* cpu for the cluster. If the entire cluster has been hotplugged, this will raise a `ValueError`.

get_cluster_cur_frequency(*cluster*)

get_cluster_governor(*cluster*)

get_cluster_governor_tunables(*cluster*)

get_cluster_max_frequency(*cluster*)

get_cluster_min_frequency(*cluster*)

get_core_clusters(*core*, *strict=True*)
Returns the list of clusters that contain the specified core. if `strict` is `True`, raises ValueError if no clusters has been found (returns empty list if `strict` is `False`).

get_core_cur_frequency(*core*)

get_core_governor(*core*)

get_core_governor_tunables(*core*)

get_core_max_frequency(*core*)

get_core_min_frequency(*core*)

get_core_online_cpu(*core*)

get_cpu_frequency(*cpu*)
Returns the current frequency currently set for the specified CPU.

Warning, this method does not check if the cpu is online or not. It will try to read the current frequency and the following exception will be raised

```
:raises: DeviceError if for some reason the frequency could not be read.
```

get_cpu_governor(*cpu*)
Returns the governor currently set for the specified CPU.

get_cpu_governor_tunables(*cpu*)

get_cpu_max_frequency(*cpu*)
Returns the max frequency currently set for the specified CPU.

Warning, this method does not check if the cpu is online or not. It will try to read the maximum frequency and the following exception will be raised

```
:raises: DeviceError if for some reason the frequency could not be read.
```

get_cpu_min_frequency(*cpu*)
Returns the min frequency currently set for the specified CPU.

Warning, this method does not check if the cpu is online or not. It will try to read the minimum frequency and the following exception will be raised

```
:raises: DeviceError if for some reason the frequency could not be read.
```

initialize(**args*, ***kwargs*)

kind = 'module'

list_available_cluster_governor_tunables(*cluster*)

list_available_cluster_governors(*cluster*)

list_available_core_frequencies(*core*)

list_available_core_governor_tunables(*core*)

list_available_core_governors(*core*)

list_available_cpu_frequencies(*cpu*)
Returns a list of frequencies supported by the cpu or an empty list if not could be found.

list_available_cpu_governor_tunables(*cpu*)
Returns a list of tunables available for the governor on the specified CPU.

list_available_cpu_governors(*cpu*)
Returns a list of governors supported by the cpu.

name = 'devcpufreq'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

probe(*device*)

set_cluster_cur_frequency(*cluster*, *freq*)

set_cluster_governor(*cluster*, *governor*, ***tunables*)

set_cluster_governor_tunables(*cluster*, *governor*, ***tunables*)

set_cluster_max_frequency(*cluster*, *freq*)

set_cluster_min_frequency(*cluster*, *freq*)

set_core_cur_frequency(*core*, *freq*)

set_core_governor(*core*, *governor*, ***tunables*)

set_core_governor_tunables(*core*, *tunables*)

set_core_max_frequency(*core*, *freq*)

set_core_min_frequency(*core*, *freq*)

set_cpu_frequency(*cpu*, *frequency*, *exact=True*)
Set's the minimum value for CPU frequency. Actual frequency will depend on the Governor used and may vary during execution. The value should be either an int or a string representing an integer.

If `exact` flag is set (the default), the Value must also be supported by the device. The available frequencies can be obtained by calling get_available_frequencies() or examining

/sys/devices/system/cpu/cpuX/cpufreq/scaling_available_frequencies

on the device (if it exists).

Raises ConfigError if the frequency is not supported by the CPU.

Raises DeviceError if, for some reason, frequency could not be set.

set_cpu_governor(*cpu*, *governor*, ***kwargs*)
Set the governor for the specified CPU. See https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

Parameters

- **cpu** – The CPU for which the governor is to be set. This must be the full name as it appears in sysfs, e.g. "cpu0".
- **governor** – The name of the governor to be used. This must be supported by the specific device.

Additional keyword arguments can be used to specify governor tunables for governors that support them.

Note On big.LITTLE all cores in a cluster must be using the same governor. Setting the governor on any core in a cluster will also set it on all other cores in that cluster.

Raises ConfigError if governor is not supported by the CPU.

Raises DeviceError if, for some reason, the governor could not be set.

set_cpu_governor_tunables(*cpu*, *governor*, ***kwargs*)
Set tunables for the specified governor. Tunables should be specified as keyword arguments. Which tunables and values are valid depends on the governor.

Parameters

- **cpu** – The cpu for which the governor will be set. This must be the full cpu name as it appears in sysfs, e.g. `cpu0`.
- **governor** – The name of the governor. Must be all lower case.

The rest should be keyword parameters mapping tunable name onto the value to be set for it.

Raises ConfigError if governor specified is not a valid governor name, or if a tunable specified is not valid for the governor.

Raises DeviceError if could not set tunable.

set_cpu_max_frequency(*cpu*, *frequency*)
Set's the minimum value for CPU frequency. Actual frequency will depend on the Governor used and may vary during execution. The value should be either an int or a string representing an integer. The Value must also be supported by the device. The available frequencies can be obtained by calling get_available_frequencies() or examining

/sys/devices/system/cpu/cpuX/cpufreq/scaling_available_frequencies

on the device.

Raises ConfigError if the frequency is not supported by the CPU.

Raises DeviceError if, for some reason, frequency could not be set.

set_cpu_min_frequency(*cpu*, *frequency*)
Set's the minimum value for CPU frequency. Actual frequency will depend on the Governor used and may vary during execution. The value should be either an int or a string representing an integer. The Value must also be supported by the device. The available frequencies can be obtained by calling get_available_frequencies() or examining

/sys/devices/system/cpu/cpuX/cpufreq/scaling_available_frequencies

on the device.

Raises ConfigError if the frequency is not supported by the CPU.

Raises DeviceError if, for some reason, frequency could not be set.

validate(**args*, ***kwargs*)

wlauto.modules.cpuidle module

class wlauto.modules.cpuidle.**Cpuidle**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['cpuidle']

core_modules = []

description = '\n Adds cpuidle state query and manuputions APIs to a Device interface.

finalize(**args*, ***kwargs*)

get_cpuidle_driver()

get_cpuidle_governor()

get_cpuidle_states(*cpu=0*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'cpuidle'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

probe(*device*)

root_path = '/sys/devices/system/cpu/cpuidle'

validate(**args*, ***kwargs*)

class wlauto.modules.cpuidle.**CpuidleState**(*device*, *index*, *path*)
Bases: object

disable

get(*prop*)

ordinal

set(*prop*, *value*)

time

usage

wlauto.modules.flashing module

class wlauto.modules.flashing.**FastbootFlasher**(*owner*, ***kwargs*)
Bases: *wlauto.modules.flashing.Flasher*

aliases = AC([])

artifacts = AC([])

core_modules = []

delay = 0.5

description = '\n Enables automated flashing of images using the fastboot utility.\n\n

finalize(**args*, ***kwargs*)

flash(*image_bundle=None*, *images=None*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'fastboot'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

partitions_file_name = 'partitions.txt'

serial_timeout = 30

validate(**args*, ***kwargs*)

class wlauto.modules.flashing.**Flasher**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

Implements a mechanism for flashing a device. The images to be flashed can be specified either as a tarball "image bundle" (in which case instructions for flashing are provided as flasher-specific metadata also in the bundle), or as individual image files, in which case instructions for flashing as specified as part of flashing config.

Note: It is important that when resolving configuration, concrete flasher implementations prioritise settings specified in the config over those in the bundle (if they happen to clash).

---

aliases = AC([])

artifacts = AC([])

capabilities = ['flash']

core_modules = []

finalize(**args*, ***kwargs*)

flash(*image_bundle=None*, *images=None*)
Flashes the specified device using the specified config. As a post condition, the device must be ready to run workloads upon returning from this method (e.g. it must be fully-booted into the OS).

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.modules.flashing.**VersatileExpressFlasher**(*owner*, ***kwargs*)
Bases: *wlauto.modules.flashing.Flasher*

aliases = AC([])

artifacts = AC([])

core_modules = []

deploy_image_bundle(*device*, *bundle*)

deploy_images(*device*, *image_bundle=None*, *images=None*)

description = "\n Enables flashing of kernels and firmware to ARM Versatile Express de

finalize(**args*, ***kwargs*)

flash(*image_bundle=None*, *images=None*, *recreate_uefi_entry=True*)

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'vexpress'

overlay_images(*device*, *images*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.modules.flashing.**expand_path**(*original_path*)

wlauto.modules.flashing.**get_mapping**(*base_dir*, *partition_file*)

wlauto.modules.flashing.**validate_image_bundle**(*bundle*)

wlauto.modules.reset module

class wlauto.modules.reset.**NetioSwitchReset**(*owner*, ***kwargs*)

Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['reset_power']

core_modules = []

description = '\n Enables hard reset of devices connected to a Netio ethernet power sw

finalize(**args*, ***kwargs*)

hard_reset()

initialize(**args*, ***kwargs*)

kind = 'module'

name = 'netio_switch'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

Module contents

wlauto.resource_getters package

Submodules

wlauto.resource_getters.standard module

This module contains the standard set of resource getters used by Workload Automation.

class wlauto.resource_getters.standard.**DependencyFileGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Gets resources from the specified mount point. Copies them the local

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'filer'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -4

relative_path = ''

resource_type = 'file'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentApkGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.EnvironmentFileGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Uses the same dependency resolution mechanism as ``EnvironmentFileGe

extension = 'apk'

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'environment_apk'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentCommonDependencyGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'environment_common_dependency'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -1

resource_type = 'file'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentDependencyGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'environment_dependency'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = 0

resource_type = 'file'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentExecutableGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.ExecutableGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'env_exe_getter'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentFileGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Looks for exactly one file with the specified extension in the owner

extension = None

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'environment_file'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

register()

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentJarGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.EnvironmentFileGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

extension = 'jar'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'environment_jar'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**EnvironmentReventGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.ReventGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get_base_location(*resource*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'enviroment_revent'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**ExecutableGetter**(*resolver*, ***kwargs*)
Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'exe_getter'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = 0

resource_type = 'executable'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**ExtensionAssetGetter**(*resolver*, ***kwargs*)
Bases: *wlauto.resource_getters.standard.EnvironmentDependencyGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'extension_asset'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

resource_type = 'extension_asset'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**HttpGetter**(*resolver*, ***kwargs*)
Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Downloads resources from a server based on an index fetched from the

download_asset(*asset*, *owner_name*)

fetch_index()

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

geturl(*url*, *stream=False*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'http_assets'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -4

resolve_resource(*resource*)

resource_type = ['apk', 'file', 'jar', 'revent', 'executable']

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageApkGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.PackageFileGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Uses the same dependency resolution mechanism as ``PackageFileGetter

extension = 'apk'

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'package_apk'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageCommonDependencyGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'packaged_common_dependency'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -11

resource_type = 'file'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageDependencyGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'packaged_dependency'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -10

resource_type = 'file'

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageExecutableGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.ExecutableGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'package_exe_getter'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -10

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageFileGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Looks for exactly one file with the specified extension in the owner

extension = None

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'package_file'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

register()

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageJarGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.PackageFileGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

extension = 'jar'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'package_jar'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**PackageReventGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.resource_getters.standard.ReventGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get_base_location(*resource*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'package_revent'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**RemoteFilerGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Finds resources on a (locally mounted) remote filer and caches them

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

get_from(*resource*, *version*, *location*)

initialize(**args*, ***kwargs*)

kind = 'resource_getter'

name = 'filer_assets'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

priority = -4

resource_type = ['apk', 'file', 'jar', 'revent']

try_get_resource(*resource*, *version*, *remote_path*, *local_path*)

validate(**args*, ***kwargs*)

class wlauto.resource_getters.standard.**ReventGetter**(*resolver*, ***kwargs*)

Bases: *wlauto.core.resource.ResourceGetter*

Implements logic for identifying revent files.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

get(*resource*, ***kwargs*)

get_base_location(*resource*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

register()

validate(**args*, ***kwargs*)

wlauto.resource_getters.standard.**get_from_list_by_extension**(*resource*, *filelist*, *extension*, *version=None*, *variant=None*)

wlauto.resource_getters.standard.**get_from_location_by_extension**(*resource*, *location*, *extension*, *version=None*, *variant=None*)

wlauto.resource_getters.standard.**get_owner_path**(*resource*)

Module contents

wlauto.result_processors package

Subpackages

wlauto.result_processors.ipynb_exporter package

Module contents

class wlauto.result_processors.ipynb_exporter.**IPythonNotebookExporter**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Generates an IPython notebook from a template with the results and r

export_run_result(*result*, *context*)

finalize(**args*, ***kwargs*)

generate_notebook(*result*, *context*)
Generate a notebook from the template and run it

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'ipynb_exporter'

open_file(*output_format*)
Open the exported notebook

open_notebook()
Open the notebook in a browser

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

Submodules

wlauto.result_processors.cpustate module

class wlauto.result_processors.cpustate.**CpuStatesProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Process power ftrace to produce CPU state and parallelism stats.\n\n

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'cpustates'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

set_initial_state(*context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.dvfs module

class wlauto.result_processors.dvfs.**DVFS**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

calculate()

core_modules = []

description = "\n Reports DVFS state residency data form ftrace power events.\n\n This

finalize(**args*, ***kwargs*)

flush_parse_initialize()

Store state, cpu_id for each timestamp from trace.txt and flush all the values for next iterations.

generate_csv(*context*)

generate the ‘‘dvfs.csv’’ with the state, frequency and cores

get_cluster(*cpu_id*, *state*)

get_cluster_freq()

get_state_name(*state*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'dvfs'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

parse()

Parse the trace.txt

```
store timestamp, state, cpu_id
------------------------------------------------------------------------------
↪---
                    |timestamp|                        |state|         |cpu_id|
<idle>-0     [001]   294.554380: cpu_idle:             state=4294967295 cpu_
↪id=1
<idle>-0     [001]   294.554454: power_start:          type=1 state=0 cpu_id=1
<idle>-0     [001]   294.554458: cpu_idle:             state=0 cpu_id=1
<idle>-0     [001]   294.554464: power_end:            cpu_id=1
```

```
<idle>-0     [001]   294.554471: cpu_idle:             state=4294967295 cpu_
↪id=1
<idle>-0     [001]   294.554590: power_start:          type=1 state=0 cpu_id=1
<idle>-0     [001]   294.554593: cpu_idle:             state=0 cpu_id=1
<idle>-0     [001]   294.554636: power_end:            cpu_id=1
<idle>-0     [001]   294.554639: cpu_idle:             state=4294967295 cpu_
↪id=1
<idle>-0     [001]   294.554669: power_start:          type=1 state=0 cpu_id=1
```

percentage()
Normalize the result with total execution time.

populate(*time1*, *time2*)

process_iteration_result(*result*, *context*)
Parse the trace.txt for each iteration, calculate DVFS residency state/frequencies and dump the result in csv and flush the data for next iteration.

unique_freq()
Determine the unique Frequency and state

update_cluster_freq(*state*, *cpu_id*)
Update the cluster frequency and current cluster

update_state(*state*, *cpu_id*)
Update state of each cores in every cluster. This is done for each timestamp.

validate(**args*, ***kwargs*)

wlauto.result_processors.mongodb module

class wlauto.result_processors.mongodb.**MongodbUploader**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Uploads run results to a MongoDB instance.\n\n MongoDB is a popular

export_iteration_result(*result*, *context*)

export_run_result(*result*, *context*)

finalize(**args*, ***kwargs*)

generate_bundle(*context*)
The bundle will contain files generated during the run that have not already been processed. This includes all files for which there isn't an explicit artifact as well as "raw" artifacts that aren't uploaded individually. Basically, this ensures that everything that is not explicilty marked as an "export" (which means it's guarnteed not to contain information not accessible from other artifacts/scores) is avialable in the DB. The bundle is compressed, so it shouldn't take up too much space, however it also means that it's not easy to query for or get individual file (a trade off between space and convinience).

gridfs_directory_exists(*path*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'mongodb'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

upload_artifact(*context*, *artifact*)

validate(**args*, ***kwargs*)

wlauto.result_processors.notify module

class wlauto.result_processors.notify.**NotifyProcessor**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = 'Display a desktop notification when the run finishes\n\n Notifications

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'notify'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.sqlite module

class wlauto.result_processors.sqlite.**SqliteResultProcessor**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Stores results in an sqlite database.\n\n This may be used accumulate

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'sqlite'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.standard module

This module contains a few "standard" result processors that write results to text files in various formats.

class wlauto.result_processors.standard.**CsvReportProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

Creates a `results.csv` in the output directory containing results for all iterations in CSV format, each line containing a single metric.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'csv'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.result_processors.standard.**JsonReportProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

Creates a `results.json` in the output directory containing results for all iterations in JSON format.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'json'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.result_processors.standard.**StandardProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Creates a ``result.txt`` file for every iteration that contains metr

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'standard'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.result_processors.standard.**SummaryCsvProcessor**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

Similar to csv result processor, but only contains workloads' summary metrics.

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'summary_csv'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.status module

class wlauto.result_processors.status.**StatusTxtReporter**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Outputs a txt file containing general status information about which

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'status'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.syeg module

class wlauto.result_processors.syeg.**SyegResult**(*max_iter*)
Bases: object

average

best

deviation

run_values

suite_version

class wlauto.result_processors.syeg.**SyegResultProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Generates a CSV results file in the format expected by SYEG toolchai

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'syeg_csv'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

wlauto.result_processors.uxperf module

class wlauto.result_processors.uxperf.**UxPerfResultProcessor**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Parse logcat for UX_PERF markers to produce performance metrics for

export_iteration_result(*result*, *context*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'result_processor'

name = 'uxperf'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

Module contents

wlauto.tests package

Submodules

wlauto.tests.test_agenda module

class wlauto.tests.test_agenda.**AgendaTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bad_syntax(**arg*, ***kw*)

test_default_id_assignment()

test_defaults()

test_dup_sections(**arg*, ***kw*)

test_duplicate_id()

test_sections()

test_yaml_load()

test_yaml_missing_field()

wlauto.tests.test_config module

class wlauto.tests.test_config.**ConfigLoaderTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_load()

test_load_bad(**arg*, ***kw*)

test_load_duplicate()

class wlauto.tests.test_config.**ConfigTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_case()

test_exetension_params_lists()

test_global_instrumentation()

test_instrumentation_specification()

test_list_defaults_params()

test_remove_instrument()

class wlauto.tests.test_config.**DefaultsWorkload**
Bases: object

class wlauto.tests.test_config.**ListParamstrument**
Bases: object

class wlauto.tests.test_config.**MockAgenda**(**args*)
Bases: object

class wlauto.tests.test_config.**MockExtensionLoader**
Bases: object

get_default_config(*name*)

get_extension_class(*name*, *kind=None*)

has_extension(*name*)

resolve_alias(*name*)

class wlauto.tests.test_config.**NamedMock**(*name*)
Bases: object

wlauto.tests.test_device module

class wlauto.tests.test_device.**RuntimeParametersTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bad_runtime_param(**arg*, ***kw*)

test_core_param()

test_get_unset_runtime_params()

test_param_set_order()

test_runtime_param()

class wlauto.tests.test_device.**TestDevice**(**args*, ***kwargs*)
Bases: *wlauto.core.device.Device*

aliases = AC([])

artifacts = AC([])

core_getter(*core*)

core_modules = []

core_setter(*core*, *value*)

dynamic_modules = AC([])

finalize(**args*, ***kwargs*)

getter()

initialize(**args*, ***kwargs*)

name = 'test-device'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

path_module = 'posixpath'

runtime_parameters = AC(['test_param', 'test_param2', '${core}_param'])

setter(*value*)

validate(**args*, ***kwargs*)

wlauto.tests.test_diff module

class wlauto.tests.test_diff.**InterruptDiffTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_interrupt_diff()

wlauto.tests.test_execution module

class wlauto.tests.test_execution.**BadDevice**(*when_to_fail*, *exception=<class 'wlauto.exceptions.DeviceError'>*)
Bases: *wlauto.core.device.Device*

aliases = AC([])

artifacts = AC([])

connect()

core_modules = []

disconnect()

dynamic_modules = AC([])

finalize(**args*, ***kwargs*)

get_properties(_)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

ping()

runtime_parameters = AC([])

set_device_parameters(**_)

start()

stop()

validate(**args*, ***kwargs*)

class wlauto.tests.test_execution.**BadDeviceMeta**
Bases: *wlauto.core.device.DeviceMeta*

class wlauto.tests.test_execution.**BadWorkload**(*exception*, *when_to_fail*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(_)

setup(_)

teardown(_)

update_result(_)

validate(**args*, ***kwargs*)

class wlauto.tests.test_execution.**Mock**
Bases: object

class wlauto.tests.test_execution.**RunnerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

bad_device(*method*)

errors = 0

signal_check(*expected_signals*, *workloads*, *reboot_policy='never'*, *runner_class=<class 'wlauto.core.execution.BySpecRunner'>*)

test_CTRL_C()

test_bad_connect(**arg*, ***kw*)

test_bad_disconnect()

test_bad_get_properties(**arg*, ***kw*)

test_bad_initialize(**arg*, ***kw*)

test_bad_start()

test_bad_stop()

test_bad_workload_status()

test_multiple_run_byiteration()

test_multiple_run_byspec()

test_no_teardown_after_setup_fail()

test_reboot_policies()

test_single_run()

test_spec_skipping()

test_teardown_on_run_and_result_update_fail()

class wlauto.tests.test_execution.**SignalCatcher**
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

handler(**_*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'Signal Catcher'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.tests.test_extension module

class wlauto.tests.test_extension.**ExtensionMetaTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_duplicate_param_spec(**arg*, ***kw*)

test_initialization()

test_initialization_happens_once()

test_invalid_param_spec(**arg*, ***kw*)

test_param_override()

test_propagation()

test_virtual_methods()

class wlauto.tests.test_extension.**FakeLoader**
Bases: object

get_module(*name*, *owner*, ***kwargs*)

modules = [<class 'wlauto.tests.test_extension.MyCoolModule'>, <class 'wlauto.tests.te

class wlauto.tests.test_extension.**ModuleTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_fizzle()

test_self_fizzle()

class wlauto.tests.test_extension.**MultiValueParamExt**(***kwargs*)
Bases: *wlauto.core.extension.Extension*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'multivalue'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyAcidExtension**(***kwargs*)
Bases: *wlauto.tests.test_extension.MyBaseExtension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'acid'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyBaseExtension**(***kwargs*)
Bases: *wlauto.core.extension.Extension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'base'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyCoolModule**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['fizzle']

core_modules = []

finalize(**args*, ***kwargs*)

fizzle()

initialize(**args*, ***kwargs*)

name = 'cool_module'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyEvenCoolerModule**(*owner*, ***kwargs*)
Bases: *wlauto.core.extension.Module*

aliases = AC([])

artifacts = AC([])

capabilities = ['fizzle']

core_modules = []

finalize(**args*, ***kwargs*)

fizzle()

initialize(**args*, ***kwargs*)

name = 'even_cooler_module'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyMeta**
Bases: *wlauto.core.extension.ExtensionMeta*

virtual_methods = ['validate', 'virtual1', 'virtual2']

class wlauto.tests.test_extension.**MyModularExtension**(***kwargs*)
Bases: *wlauto.core.extension.Extension*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'modular'

parameters = AC(["Param({'_overridden': 'MyModularExtension', 'kind': <type 'list'>,

validate(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyOtherExtension**(***kwargs*)
Bases: *wlauto.tests.test_extension.MyBaseExtension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'other'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyOtherModularExtension**(***kwargs*)
Bases: *wlauto.core.extension.Extension*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'other_modular'

parameters = AC(["Param({'_overridden': 'MyOtherModularExtension', 'kind': <type 'li

validate(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyOtherOtherExtension**(***kwargs*)
Bases: *wlauto.tests.test_extension.MyOtherExtension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'otherother'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyOverridingExtension**(***kwargs*)
Bases: *wlauto.tests.test_extension.MyAcidExtension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'overriding'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**MyThirdTeerExtension**(***kwargs*)
Bases: *wlauto.tests.test_extension.MyOverridingExtension*

aliases = AC([])

artifacts = AC([])

core_modules = []

name = 'thirdteer'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

virtual1(**args*, ***kwargs*)

virtual2(**args*, ***kwargs*)

class wlauto.tests.test_extension.**ParametersTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bad_multivalue_param(**arg*, ***kw*)

test_default_override()

test_duplicate_param_override(**arg*, ***kw*)

test_multivalue_param()

test_overriding_new_param(**arg*, ***kw*)

test_setting()

test_validation_bad_value(**arg*, ***kw*)

test_validation_no_mandatory(**arg*, ***kw*)

test_validation_no_mandatory_in_derived(**arg*, ***kw*)

test_validation_ok()

wlauto.tests.test_extension_loader module

class wlauto.tests.test_extension_loader.**ExtensionLoaderTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_clear_and_reload()

test_list_by_kind()

test_load_device()

wlauto.tests.test_instrumentation module

class wlauto.tests.test_instrumentation.**BadInstrument**
Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'bad'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

teardown()

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**InstrumentationTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

install_local_instrument()

tearDown()

test_bad_argspec(**arg*, ***kw*)

test_check_enabled()

test_check_installed()

test_duplicate_install(**arg*, ***kw*)

test_enable_disable()

test_install()

test_local_instrument()

test_priority_prefix_instrument()

class wlauto.tests.test_instrumentation.**MockInstrument**
Bases: *wlauto.core.instrumentation.Instrument*

after_workload_execution(*context*)

aliases = AC([])

artifacts = AC([])

before_workload_execution(*context*)

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**MockInstrument2**

Bases: *wlauto.core.instrumentation.Instrument*

after_workload_execution(*context*)

after_workload_result_update(*context*)

aliases = AC([])

artifacts = AC([])

before_workload_execution(*context*)

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock_2'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**MockInstrument3**

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock_3'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

slow_before_workload_execution(*context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**MockInstrument4**

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock_4'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

slow_before_first_iteration_boot(*context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**MockInstrument5**

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

core_modules = []

fast_before_first_iteration_boot(*context*)

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock_5'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

class wlauto.tests.test_instrumentation.**MockInstrument6**

Bases: *wlauto.core.instrumentation.Instrument*

aliases = AC([])

artifacts = AC([])

before_first_iteration_boot(*context*)

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'mock_6'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.tests.test_results_manager module

class wlauto.tests.test_results_manager.**MockResultProcessor1**(***kwargs*)

Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'result_processor_with_exception'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_results_manager.**MockResultProcessor2**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'result_processor_with_wa_error'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_results_manager.**MockResultProcessor3**(***kwargs*)
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'result_processor_with_keybaord_interrupt'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_results_manager.**MockResultProcessor4**
Bases: *wlauto.core.result.ResultProcessor*

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

name = 'result_processor'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

process_iteration_result(*result*, *context*)

process_run_result(*result*, *context*)

validate(**args*, ***kwargs*)

class wlauto.tests.test_results_manager.**ResultManagerTest**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_add_result()

test_keyboard_interrupt()

test_process_results()

wlauto.tests.test_utils module

class wlauto.tests.test_utils.**TestCheckOutput**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bad(**arg*, ***kw*)

test_ok()

class wlauto.tests.test_utils.**TestMerge**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_dict_merge()

test_merge_dict_lists()

test_merge_lists()

test_type_mismatch(**arg*, ***kw*)

class wlauto.tests.test_utils.**TestParameterDict**(*methodName='runTest'*)
Bases: unittest.case.TestCase

orig_params = {'none': None, 'string': 'A Test String', 'int': 42, 'float': 1.23,

setUp()

test_contains()

test_get()

test_getEncodedItems()

test_getitem()

test_iteritems()

test_parameter_dict_update()

test_pop()

class wlauto.tests.test_utils.**TestTypes**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_arguments()

test_caseless_string()

test_list_or_conversion()

Module contents

wlauto.tools package

Submodules

wlauto.tools.extdoc module

This module contains utilities for generating user documentation for Workload Automation Extensions.

class wlauto.tools.extdoc.**ExtensionDocumenter**(*ext*)
Bases: object

description
The description for an extension is specified in the `description` attribute, or (legacy) as a docstring for the extension's class. If neither method is used in the Extension, an empty string is returned.

Description is assumed to be formed as reStructuredText. Leading and trailing whitespace will be stripped away.

name

parameters

summary
Returns the summary description for this Extension, which, by convention, is the first paragraph of the description.

class wlauto.tools.extdoc.**ExtensionParameterDocumenter**(*param*)
Bases: object

constraint

default

description

kind

name

wlauto.tools.extdoc.**get_paragraphs**(*text*)
returns a list of paragraphs contained in the text

Module contents

wlauto.utils package

Submodules

wlauto.utils.android module

Utility functions for working with Android devices through adb.

class wlauto.utils.android.**AdbDevice**(*name*, *status*)
Bases: object

class wlauto.utils.android.**AndroidProperties**(*text*)
Bases: object

parse(*text*)

class wlauto.utils.android.**ApkInfo**(*path=None*)
Bases: object

name_regex = <_sre.SRE_Pattern object>

parse(*apk_path*)

version_regex = <_sre.SRE_Pattern object at 0x47fa5d0>

wlauto.utils.android.**adb_background_shell**(*device*, *command*, *stdout=-1*, *stderr=-1*, *as_root=False*)
Runs the sepcified command in a subprocess, returning the the Popen object.

wlauto.utils.android.**adb_command**(*device*, *command*, *timeout=None*)

wlauto.utils.android.**adb_connect**(*device*, *timeout=None*)

wlauto.utils.android.**adb_disconnect**(*device*)

wlauto.utils.android.**adb_get_device**()
Returns the serial number of a connected android device.

If there are more than one device connected to the machine, or it could not find any device connected, *wlauto.exceptions.ConfigError* is raised.

wlauto.utils.android.**adb_list_devices**()

wlauto.utils.android.**adb_shell**(*device*, *command*, *timeout=None*, *check_exit_code=False*, *as_root=False*)

wlauto.utils.android.**fastboot_command**(*command*, *timeout=None*)

wlauto.utils.android.**fastboot_flash_partition**(*partition*, *path_to_image*)

wlauto.utils.android.**poll_for_file**(*device*, *dfile*)

wlauto.utils.cli module

wlauto.utils.cli.**init_argument_parser**(*parser*)

wlauto.utils.cpuinfo module

class wlauto.utils.cpuinfo.**Cpuinfo**(*text*)
Bases: object

architecture

parse(*text*)

wlauto.utils.cros_sdk module

class wlauto.utils.cros_sdk.**CrosSdkSession**(*cros_path*, *password=''*)
Bases: object

get_lines(*timeout=0*, *timeout_only_for_first_line=True*, *from_stderr=False*)

kill_session()

read_line(*timeout=0*)

read_stderr_line(*timeout=0*)

send_command(*cmd*, *flush=True*)

class wlauto.utils.cros_sdk.**OutputPollingThread**(*out*, *queue*, *name*)
Bases: threading.Thread

run()

set_stop()

wlauto.utils.doc module

Utilities for working with and formatting documentation.

wlauto.utils.doc.**count_leading_spaces**(*text*)
Counts the number of leading space characters in a string.

TODO: may need to update this to handle whitespace, but shouldn't be necessary as there should be no tabs in Python source.

wlauto.utils.doc.**format_body**(*text*, *width*)
Format the specified text into a column of specified width. The text is assumed to be a "body" of one or more paragraphs separated by one or more blank lines. The initial indentation of the first line of each paragraph will be presevered, but any other formatting may be clobbered.

wlauto.utils.doc.**format_bullets**(*text*, *width*, *char='-'*, *shift=3*, *outchar=None*)
Formats text into bulleted list. Assumes each line of input that starts with char (possibly preceeded with whitespace) is a new bullet point. Note: leading whitespace in the input will *not* be preserved. Instead, it will be determined by shift parameter.

Text the text to be formated

Width format width (note: must be at least shift + 4).

Char character that indicates a new bullet point in the input text.

Shift How far bulleted entries will be indented. This indicates the indentation level of the bullet point. Text indentation level will be shift + 3.

Outchar character that will be used to mark bullet points in the output. If left as None, char will be used.

wlauto.utils.doc.**format_column**(*text*, *width*)
Formats text into a column of specified width. If a line is too long, it will be broken on a word boundary. The new lines will have the same number of leading spaces as the original line.

Note: this will not attempt to join up lines that are too short.

wlauto.utils.doc.**format_literal**(*lit*)

wlauto.utils.doc.**format_paragraph**(*text*, *width*)
Format the specified text into a column of specified with. The text is assumed to be a single paragraph and existing line breaks will not be preserved. Leading spaces (of the initial line), on the other hand, will be preserved.

wlauto.utils.doc.**format_simple_table**(*rows*, *headers=None*, *align='>'*, *show_borders=True*, *borderchar='='*)
Formats a simple table.

wlauto.utils.doc.**get_description**(*aclass*)
Return the description of the specified extension class. The description is taken either from `description` attribute of the class or its docstring.

wlauto.utils.doc.**get_params_rst**(*ext*)

wlauto.utils.doc.**get_rst_from_extension**(*ext*)

wlauto.utils.doc.**get_summary**(*aclass*)
Returns the summary description for an extension class. The summary is the first paragraph (separated by blank line) of the description taken either from the `descripton` attribute of the class, or if that is not present, from the class' docstring.

wlauto.utils.doc.**get_type_name**(*obj*)
Returns the name of the type object or function specified. In case of a lambda, the definiition is returned with the parameter replaced by "value".

wlauto.utils.doc.**indent**(*text*, *spaces=4*)
Indent the lines i the specified text by `spaces` spaces.

wlauto.utils.doc.**strip_inlined_text**(*text*)
This function processes multiline inlined text (e.g. form docstrings) to strip away leading spaces and leading and trailing new lines.

wlauto.utils.doc.**underline**(*text*, *symbol='='*)

wlauto.utils.formatter module

class wlauto.utils.formatter.**DescriptionListFormatter**(*title=None*, *width=None*)
Bases: *wlauto.utils.formatter.TextFormatter*

add_item(*new_data*, *item_title*)

data = None

format_data()

get_text_width()

name = 'description_list_formatter'

set_text_width(*value*)

text_width

class wlauto.utils.formatter.**TextFormatter**
Bases: `object`

This is a base class for text formatting. It mainly ask to implement two methods which are add_item and format_data. The formar will add new text to the formatter, whereas the latter will return a formatted text. The name attribute represents the name of the foramtter.

add_item(*new_data*, *item_title*)
Add new item to the text formatter.

Parameters

- **new_data** – The data to be added
- **item_title** – A title for the added data

data = None

format_data()
It returns a formatted text

name = None

wlauto.utils.fps module

class wlauto.utils.fps.**FpsProcessor**(*data*, *action=None*, *extra_data=None*)
Bases: object

Provides common object for processing surfaceFlinger output for frame statistics.

This processor returns the four frame statistics below:

FPS Frames Per Second. This is the frame rate of the workload.

frame_count The total number of frames rendered during the execution of the workload.

janks The number of "janks" that occurred during execution of the workload. Janks are sudden shifts in frame rate. They result in a "stuttery" UI. See http://jankfree.org/jank-busters-io

not_at_vsync The number of frames that did not render in a single vsync cycle.

percentiles()

process(*refresh_period*, *drop_threshold*)
Generate frame per second (fps) and associated metrics for workload.

refresh_period - the vsync interval drop_threshold - data points below this fps will be dropped

class wlauto.utils.fps.**GfxInfoFrame**(*Flags*, *IntendedVsync*, *Vsync*, *OldestInputEvent*, *NewestInputEvent*, *HandleInputStart*, *AnimationStart*, *PerformTraversalsStart*, *DrawStart*, *SyncQueued*, *SyncStart*, *IssueDrawCommandsStart*, *SwapBuffers*, *FrameCompleted*)
Bases: tuple

AnimationStart
Alias for field number 6

DrawStart
Alias for field number 8

Flags
Alias for field number 0

FrameCompleted
Alias for field number 13

HandleInputStart
Alias for field number 5

IntendedVsync
Alias for field number 1

IssueDrawCommandsStart
Alias for field number 11

NewestInputEvent
Alias for field number 4

OldestInputEvent
Alias for field number 3

PerformTraversalsStart
Alias for field number 7

SwapBuffers
Alias for field number 12

SyncQueued
Alias for field number 9

SyncStart
Alias for field number 10

Vsync
Alias for field number 2

class wlauto.utils.fps.**SurfaceFlingerFrame**(*desired_present_time*, *actual_present_time*, *frame_ready_time*)

Bases: tuple

actual_present_time
Alias for field number 1

desired_present_time
Alias for field number 0

frame_ready_time
Alias for field number 2

wlauto.utils.hwmon module

class wlauto.utils.hwmon.**HwmonSensor**(*device*, *kind*, *device_name*, *label*, *filepath*)

Bases: object

clear_readings()

take_reading()

wlauto.utils.hwmon.**discover_sensors**(*device*, *sensor_kinds*)

Discovers HWMON sensors available on the device.

device Device on which to discover HWMON sensors. Must be an instance of AndroidDevice.

sensor_kinds A list of names of sensor types to be discovered. The names must be as they appear prefixed to *_input files in sysfs. E.g. 'energy'.

returns A list of *HwmonSensor* instantces for each found sensor. If no sensors of the specified types were discovered, an empty list will be returned.

wlauto.utils.ipython module

wlauto.utils.ipython.**export_notebook**(*nbbasename*, *output_directory*, *output_format*)

Generate a PDF or HTML from the ipython notebook

output_format has to be either 'pdf' or 'html'. These are the only formats currently supported.

ipython nbconvert claims that the CLI is not stable, so keep this function here to be able to cope with inconsistencies

wlauto.utils.ipython.**parse_valid_output**(*msg*)
Parse a valid result from an execution of a cell in an ipython kernel

wlauto.utils.ipython.**run_cell**(*kernel_client*, *cell*)
Run a cell of a notebook in an ipython kernel and return its output

wlauto.utils.ipython.**run_notebook**(*notebook*)
Run the notebook

wlauto.utils.log module

class wlauto.utils.log.**BaseLogWriter**(*name*, *level=10*)
Bases: object

close()

flush()

class wlauto.utils.log.**ColorFormatter**(*fmt=None*, *datefmt=None*)
Bases: logging.Formatter

Formats logging records with color and prepends record info to each line of the message.

BLUE for DEBUG logging level GREEN for INFO logging level YELLOW for WARNING logging level RED for ERROR logging level BOLD RED for CRITICAL logging level

format(*record*)

class wlauto.utils.log.**ErrorSignalHandler**(*level=0*)
Bases: logging.Handler

Emits signals for ERROR and WARNING level traces.

emit(*record*)

class wlauto.utils.log.**LineFormatter**(*fmt=None*, *datefmt=None*)
Bases: logging.Formatter

Logs each line of the message separately.

format(*record*)

class wlauto.utils.log.**LineLogWriter**(*name*, *level=10*)
Bases: *wlauto.utils.log.BaseLogWriter*

write(*data*)

class wlauto.utils.log.**LogWriter**(*name*, *level=10*)
Bases: *wlauto.utils.log.BaseLogWriter*

write(*data*)

class wlauto.utils.log.**StreamLogger**(*name*, *stream*, *level=10*, *klass=<class 'wlauto.utils.log.LogWriter'>*)
Bases: threading.Thread

Logs output from a stream in a thread.

run()

wlauto.utils.log.**add_log_file**(*filepath*, *level=10*)

wlauto.utils.log.**init_logging**(*verbosity*)

wlauto.utils.misc module

Miscellaneous functions that don't fit anywhere else.

exception wlauto.utils.misc.**CalledProcessErrorWithStderr**(**args*, ***kwargs*)
Bases: subprocess.CalledProcessError

exception wlauto.utils.misc.**LoadSyntaxError**(*message*, *filepath*, *lineno*)
Bases: exceptions.Exception

exception wlauto.utils.misc.**TimeoutError**(*command*, *output*)
Bases: exceptions.Exception

Raised when a subprocess command times out. This is basically a WAError-derived version of subprocess.CalledProcessError, the thinking being that while a timeout could be due to programming error (e.g. not setting long enough timers), it is often due to some failure in the environment, and there fore should be classed as a "user error".

wlauto.utils.misc.**as_relative**(*path*)
Convert path to relative by stripping away the leading '/' on UNIX or the equivant on other platforms.

wlauto.utils.misc.**capitalize**(*text*)
Capitalises the specified text: first letter upper case, all subsequent letters lower case.

wlauto.utils.misc.**check_output**(*command*, *timeout=None*, *ignore=None*, ***kwargs*)
This is a version of subprocess.check_output that adds a timeout parameter to kill the subprocess if it does not return within the specified time.

wlauto.utils.misc.**commonprefix**(*file_list*, *sep='/'*)
Find the lowest common base folder of a passed list of files.

wlauto.utils.misc.**convert_new_lines**(*text*)
Convert new lines to a common format.

wlauto.utils.misc.**diff_tokens**(*before_token*, *after_token*)
Creates a diff of two tokens.

If the two tokens are the same it just returns returns the token (whitespace tokens are considered the same irrespective of type/number of whitespace characters in the token).

If the tokens are numeric, the difference between the two values is returned.

Otherwise, a string in the form [before -> after] is returned.

wlauto.utils.misc.**ensure_directory_exists**(*dirpath*)
A filter for directory paths to ensure they exist.

wlauto.utils.misc.**ensure_file_directory_exists**(*filepath*)
A filter for file paths to ensure the directory of the file exists and the file can be created there. The file itself is *not* going to be created if it doesn't already exist.

wlauto.utils.misc.**enum_metaclass**(*enum_param*, *return_name=False*, *start=0*)
Returns a type subclass that may be used as a metaclass for an enum.

Paremeters:

enum_param the name of class attribute that defines enum values. The metaclass will add a class attribute for each value in enum_param. The value of the attribute depends on the type of enum_param and on the values of return_name. If return_name is True, then the value of the new attribute is the name of that attribute; otherwise, if enum_param is a list or a tuple, the value will be the index of that param in enum_param, optionally offset by start, otherwise, it will

be assumed that `enum_param` implementa a dict-like inteface and the value will be `enum_param[attr_name]`.

return_name If `True`, the enum values will the names of enum attributes. If `False`, the default, the values will depend on the type of `enum_param` (see above).

start If `enum_param` is a list or a tuple, and `return_name` is `False`, this specifies an "offset" that will be added to the index of the attribute within `enum_param` to form the value.

`wlauto.utils.misc.`**`escape_double_quotes`**(*text*)
Escape double quotes, and escaped double quotes, in the specified text.

`wlauto.utils.misc.`**`escape_quotes`**(*text*)
Escape quotes, and escaped quotes, in the specified text.

`wlauto.utils.misc.`**`escape_single_quotes`**(*text*)
Escape single quotes, and escaped single quotes, in the specified text.

`wlauto.utils.misc.`**`format_duration`**(*seconds, sep=' ', order=['day', 'hour', 'minute', 'second']*)
Formats the specified number of seconds into human-readable duration.

`wlauto.utils.misc.`**`geomean`**(*values*)
Returns the geometric mean of the values.

`wlauto.utils.misc.`**`get_article`**(*word*)
Returns the appropriate indefinite article for the word (ish).

Note: Indefinite article assignment in English is based on sound rather than spelling, so this will not work correctly in all case; e.g. this will return `"a hour"`.

`wlauto.utils.misc.`**`get_cpu_mask`**(*cores*)
Return a string with the hex for the cpu mask for the specified core numbers.

`wlauto.utils.misc.`**`get_meansd`**(*values*)
Returns mean and standard deviation of the specified values.

`wlauto.utils.misc.`**`get_null`**()
Returns the correct null sink based on the OS.

`wlauto.utils.misc.`**`get_pager`**()
Returns the name of the system pager program.

`wlauto.utils.misc.`**`get_random_string`**(*length*)
Returns a random ASCII string of the specified length).

`wlauto.utils.misc.`**`get_traceback`**(*exc=None*)
Returns the string with the traceback for the specifiec exc object, or for the current exception exc is not specified.

`wlauto.utils.misc.`**`getch`**(*count=1*)
Read `count` characters from standard input.

`wlauto.utils.misc.`**`isiterable`**(*obj*)
Returns `True` if the specified object is iterable and *is not a string type*, `False` otherwise.

`wlauto.utils.misc.`**`list_to_mask`**(*values, base=0*)
Converts the specified list of integer values into a bit mask for those values. Optinally, the list can be applied to an existing mask.

`wlauto.utils.misc.`**`list_to_ranges`**(*values*)
Converts a list, e.g `[0,2,3,4]`, into a sysfs-style ranges string, e.g. `"0,2-4"`

wlauto.utils.misc.**load_class**(*classpath*)
Loads the specified Python class. `classpath` must be a fully-qualified class name (i.e. namspaced under module/package).

wlauto.utils.misc.**load_struct_from_file**(*filepath*)
Attempts to parse a Python structure consisting of basic types from the specified file. Raises a `ValueError` if the specified file is of unkown format; `LoadSyntaxError` if there is an issue parsing the file.

wlauto.utils.misc.**load_struct_from_python**(*filepath=None*, *text=None*)
Parses a config structure from a .py file. The structure should be composed of basic Python types (strings, ints, lists, dicts, etc.).

wlauto.utils.misc.**load_struct_from_yaml**(*filepath=None*, *text=None*)
Parses a config structure from a .yaml file. The structure should be composed of basic Python types (strings, ints, lists, dicts, etc.).

wlauto.utils.misc.**local_to_utc**(*dt*)
Convert naive datetime to UTC, assuming local time zone.

wlauto.utils.misc.**mask_to_list**(*mask*)
Converts the specfied integer bitmask into a list of indexes of bits that are set in the mask.

wlauto.utils.misc.**memoized**(*func*)
A decorator for memoizing functions and methods.

wlauto.utils.misc.**merge_dicts**(**args*, ***kwargs*)

wlauto.utils.misc.**merge_lists**(**args*, ***kwargs*)

wlauto.utils.misc.**normalize**(*value*, *dict_type=<type 'dict'>*)
Normalize values. Recursively normalizes dict keys to be lower case, no surrounding whitespace, underscore-delimited strings.

wlauto.utils.misc.**open_file**(*filepath*)
Open the specified file path with the associated launcher in an OS-agnostic way.

wlauto.utils.misc.**parse_value**(*value_string*)
parses a string representing a numerical value and returns a tuple (value, units), where value will be either int or float, and units will be a string representing the units or None.

wlauto.utils.misc.**preexec_function**()

wlauto.utils.misc.**prepare_table_rows**(*rows*)
Given a list of lists, make sure they are prepared to be formatted into a table by making sure each row has the same number of columns and stringifying all values.

wlauto.utils.misc.**ranges_to_list**(*ranges_string*)
Converts a sysfs-style ranges string, e.g. `"0,2-4"`, into a list ,e.g `[0,2,3,4]`

wlauto.utils.misc.**sha256**(*path*, *chunk=2048*)
Calculates SHA256 hexdigest of the file at the specified path.

wlauto.utils.misc.**strip_bash_colors**(*text*)

wlauto.utils.misc.**to_identifier**(*text*)
Converts text to a valid Python identifier by replacing all whitespace and punctuation.

wlauto.utils.misc.**unique**(*alist*)
Returns a list containing only unique elements from the input list (but preserves order, unlike sets).

wlauto.utils.misc.**urljoin**(**parts*)

wlauto.utils.misc.**utc_to_local**(*dt*)
Convert naive datetime to local time zone, assuming UTC.

wlauto.utils.misc.**walk_modules**(*path*)
Given package name, return a list of all modules (including submodules, etc) in that package.

wlauto.utils.misc.**which**(*name*)
Platform-independent version of UNIX which utility.

wlauto.utils.misc.**write_table**(*rows*, *wfh*, *align='>'*, *headers=None*)
Write a column-aligned table to the specified file object.

wlauto.utils.netio module

This module contains utilities for implemening device hard reset using Netio 230 series power switches. This utilizes the KSHELL connection.

class wlauto.utils.netio.**KshellConnection**(*host='ippowerbar'*, *port=1234*, *timeout=None*)
Bases: object

close()

delay = 0.5

disable_port(*port*)
Enable the power supply at the specified port.

enable_port(*port*)
Enable the power supply at the specified port.

login(*user*, *password*)

response_regex = <_sre.SRE_Pattern object>

send_command(*command*)

set_port(*port*, *value*)

exception wlauto.utils.netio.**NetioError**
Bases: exceptions.Exception

wlauto.utils.power module

class wlauto.utils.power.**CorePowerDroppedEvents**(*cpu_id*)
Bases: object

cpu_id

kind = 'dropped_events'

class wlauto.utils.power.**CorePowerTransitionEvent**(*timestamp*, *cpu_id*, *frequency=None*, *idle_state=None*)
Bases: object

cpu_id

frequency

idle_state

kind = 'transition'

timestamp

class wlauto.utils.power.**CpuPowerState**(*frequency=None*, *idle_state=None*)
Bases: object

frequency

idle_state

is_active

is_idling

class wlauto.utils.power.**CpuUtilisationTimeline**(*filepath*, *core_names*, *max_freq_list*)
Bases: object

report()

update(*timestamp*, *core_states*)

class wlauto.utils.power.**ParallelReport**
Bases: object

add(*value*)

write(*filepath*)

class wlauto.utils.power.**ParallelStats**(*core_clusters*, *use_ratios=False*)
Bases: object

report()

update(*timestamp*, *core_states*)

class wlauto.utils.power.**PowerStateProcessor**(*core_clusters*, *num_idle_states*, *first_cluster_state=9223372036854775807*, *first_system_state=9223372036854775807*, *wait_for_start_marker=False*, *no_idle=False*)
Bases: object

This takes a stream of power transition events and yields a timeline stream of system power states.

cpu_states

current_time

process(*event_stream*)

update_power_state(*event*)
Update the tracked power state based on the specified event and return updated power state.

class wlauto.utils.power.**PowerStateStats**(*core_names*, *idle_state_names=None*, *use_ratios=False*)
Bases: object

report()

update(*timestamp*, *core_states*)

class wlauto.utils.power.**PowerStateStatsReport**(*state_stats*, *core_names*, *precision=2*)
Bases: object

write(*filepath*)

class wlauto.utils.power.**PowerStateTimeline**(*filepath*, *core_names*, *idle_state_names*)
Bases: object

report()

update(*timestamp*, *core_states*)

class wlauto.utils.power.**PowerStateTransitions**(*filepath*)
Bases: object

record_transition(*transition*)

report()

update(*timestamp*, *core_states*)

class wlauto.utils.power.**SplitListAction**(*option_strings*, *dest*, *nargs=None*, ***kwargs*)
Bases: argparse.Action

class wlauto.utils.power.**SystemPowerState**(*num_cores*, *no_idle=False*)
Bases: object

copy()

cpus

num_cores

timestamp

class wlauto.utils.power.**TraceMarkerEvent**(*name*)
Bases: object

kind = 'marker'

name

wlauto.utils.power.**build_idle_domains**(*core_clusters*, *num_states*, *first_cluster_state=None*, *first_system_state=None*)
Returns a list of idle domain groups (one for each idle state). Each group is a list of domains, and a domain is a list of cpu ids for which that idle state is common. E.g.

[[[0], [1], [2]], [[0, 1], [2]], [[0, 1, 2]]]

This defines three idle states for a machine with three cores. The first idle state has three domains with one core in each domain; the second state has two domains, with cores 0 and 1 sharing one domain; the final state has only one domain shared by all cores.

This mapping created based on the assumptions

- The device is an SMP or a big.LITTLE-like system with cores in one or more clusters (for SMP systems, all cores are considered to be in a "cluster").
- Idle domain correspond to either individual cores, individual custers, or the compute subsystem as a whole.
- Cluster states are always deeper (higher index) than core states, and system states are always deeper than cluster states.

parameters:

core_clusters a list indicating cluster "ID" of the corresponing core, e.g. `[0, 0, 1]` represents a three-core machines with cores 0 and 1 on cluster 0, and core 2 on cluster 1.

num_states total number of idle states on a device.

first_cluster_state the ID of the first idle state shared by all cores in a cluster

first_system_state the ID of the first idle state shared by all cores.

wlauto.utils.power.**gather_core_states**(*system_state_stream*, *freq_dependent_idle_states=None*)

wlauto.utils.power.**main**()

wlauto.utils.power.**parse_arguments**()

wlauto.utils.power.**record_state_transitions**(*reporter*, *stream*)

wlauto.utils.power.**report_power_stats**(*trace_file*, *idle_state_names*, *core_names*, *core_clusters*, *num_idle_states*, *first_cluster_state=9223372036854775807*, *first_system_state=9223372036854775807*, *use_ratios=False*, *timeline_csv_file=None*, *cpu_utilisation=None*, *max_freq_list=None*, *start_marker_handling='error'*, *transitions_csv_file=None*, *no_idle=False*)

wlauto.utils.power.**stream_cpu_power_transitions**(*events*)

wlauto.utils.revent module

class wlauto.utils.revent.**ReventEvent**(*fh*, *legacy=False*)
Bases: object

class wlauto.utils.revent.**ReventRecording**(*f*, *stream=True*)
Bases: object

Represents a parsed revent recording. This contains input events and device descriptions recorded by revent. Two parsing modes are supported. By default, the recording will be parsed in the "streaming" mode. In this mode, initial headers and device descritions are parsed on creation and an open file handle to the recording is saved. Events will be read from the file as they are being iterated over. In this mode, the entire recording is never loaded into memory at once. The underlying file may be "released" by calling `close` on the recroding, after which further iteration over the events will not be possible (but would still be possible to access the file description and header information).

The alternative is to load the entire recording on creation (in which case the file handle will be closed once the recroding is loaded). This can be enabled by specifying `streaming=False`. This will make it faster to subsequently iterate over the events, and also will not "hold" the file open.

Note: When starting a new iteration over the events in streaming mode, the postion in the open file will be automatically reset to the beginning of the event stream. This means it's possible to iterate over the events multiple times without having to re-open the recording, however it is not possible to do so in parallel. If parallel iteration is required, streaming should be disabled.

---

close()

duration

events

class wlauto.utils.revent.**UinputDeviceInfo**(*fh*)
Bases: object

class wlauto.utils.revent.**absinfo**(*ev_code*, *value*, *min*, *max*, *fuzz*, *flat*, *resolution*)
Bases: tuple

ev_code
Alias for field number 0

flat
Alias for field number 5

fuzz
Alias for field number 4

max
Alias for field number 3

min
Alias for field number 2

resolution
Alias for field number 6

value
Alias for field number 1

wlauto.utils.revent.**count_bits**(*bitarr*)

wlauto.utils.revent.**is_set**(*bitarr*, *bit*)

wlauto.utils.revent.**read_string**(*fh*)

wlauto.utils.revent.**read_struct**(*fh*, *struct_spec*)

wlauto.utils.serial_port module

class wlauto.utils.serial_port.**PexpectLogger**(*kind*)
Bases: *wlauto.utils.log.LogWriter*

wlauto.utils.serial_port.**get_connection**(*timeout*, *init_dtr=None*, **args*, ***kwargs*)

wlauto.utils.serial_port.**open_serial_connection**(**args*, ***kwds*)
Opens a serial connection to a device.

Parameters

- **timeout** – timeout for the fdpexpect spawn object.
- **conn** – bool that specfies whether the underlying connection object should be yielded as well.
- **init_dtr** – specifies the initial DTR state stat should be set.

All arguments are passed into the __init__ of serial.Serial. See pyserial documentation for details:

http://pyserial.sourceforge.net/pyserial_api.html#serial.Serial

Returns a pexpect spawn object connected to the device. See: http://pexpect.sourceforge.net/pexpect.html

wlauto.utils.serial_port.**pulse_dtr**(*conn*, *state=True*, *duration=0.1*)
Set the DTR line of the specified serial connection to the specified state for the specified duration (note: the initial state of the line is *not* checked.

wlauto.utils.ssh module

class wlauto.utils.ssh.**SshShell**(*password_prompt=None*, *timeout=10*, *telnet=False*)
Bases: object

background(*command*, *stdout=-1*, *stderr=-1*)

cancel_running_command()

default_password_prompt = '[sudo] password'

execute(*command*, *timeout=None*, *check_exit_code=True*, *as_root=False*, *strip_colors=True*)

login(*host*, *username*, *password=None*, *keyfile=None*, *port=None*, *timeout=None*)

logout()

max_cancel_attempts = 5

pull_file(*source*, *dest*, *timeout=30*)

push_file(*source*, *dest*, *timeout=30*)

reconnect()

class wlauto.utils.ssh.**TelnetConnection**(*timeout=30*, *maxread=2000*, *searchwindowsize=None*, *logfile=None*, *cwd=None*, *env=None*, *ignore_sighup=True*, *echo=True*, *options={}*, *encoding=None*, *codec_errors='strict'*, *debug_command_string=False*)

Bases: pexpect.pxssh.pxssh

login(*server*, *username*, *password=''*, *original_prompt='[#$]'*, *login_timeout=10*, *auto_prompt_reset=True*, *sync_multiplier=1*, *port=23*)

wlauto.utils.ssh.**check_keyfile**(*keyfile*)

keyfile must have the right access premissions in order to be useable. If the specified file doesn't, create a temporary copy and set the right permissions for that.

Returns either the keyfile (if the permissions on it are correct) or the path to a temporary copy with the right permissions.

wlauto.utils.ssh.**process_backspaces**(*text*)

wlauto.utils.ssh.**ssh_get_shell**(*host*, *username*, *password=None*, *keyfile=None*, *port=None*, *timeout=10*, *telnet=False*, *original_prompt=None*)

wlauto.utils.statedetect module

State detection functionality for revent workloads. Uses OpenCV to analyse screenshots from the device. Requires a 'statedetection' directory in the workload directory that includes the state definition yaml file, and the 'templates' folder with PNGs of all templates mentioned in the yaml file.

Requires the following Python libraries: numpy, pyyaml (yaml), imutils and opencv-python

exception wlauto.utils.statedetect.**StateDefinitionError**

Bases: exceptions.RuntimeError

wlauto.utils.statedetect.**auto_canny**(*image*, *sigma=0.33*)

wlauto.utils.statedetect.**check_match_state_dependencies**()

wlauto.utils.statedetect.**match_state**(*screenshot_file*, *defpath*, *state_definitions*)

wlauto.utils.statedetect.**verify_state**(*screenshot_file*, *state_defs_path*, *workload_phase*)

wlauto.utils.terminalsize module

wlauto.utils.terminalsize.**get_terminal_size**()

getTerminalSize() - get width and height of console - works on linux,os x,windows,cygwin(windows) originally retrieved from: http://stackoverflow.com/questions/566746/how-to-get-console-window-width-in-python

wlauto.utils.trace_cmd module

class wlauto.utils.trace_cmd.**DroppedEventsEvent**(*cpu_id*)
Bases: object

fields

name

reporting_cpu_id

text

thread

timestamp

class wlauto.utils.trace_cmd.**TraceCmdEvent**(*thread*, *cpu_id*, *ts*, *name*, *body*, *parser=None*)
Bases: object

A single trace-cmd event. This will appear in the trace cmd report in the format

```
<idle>-0     [000]  3284.126993: sched_rq_runnable_load: cpu=0 load=54
   |           |         |               |                 |_____________|
   |           |         |               |                       |
thread        cpu    timestamp        name                    body
```

fields

name

reporting_cpu_id

text

thread

timestamp

class wlauto.utils.trace_cmd.**TraceCmdTrace**(*file_path*, *names=None*, *filter_markers=True*)
Bases: object

has_start_marker

parse()
This is a generator for the trace event stream.

wlauto.utils.trace_cmd.**default_body_parser**(*event*, *text*)
Default parser to attempt to use to parser body text for the event (i.e. after the "header" common to all events has been parsed). This assumes that the body is a whitespace-separated list of key=value pairs. The parser will attempt to convert the value into a numeric type, and failing that, keep it as string.

wlauto.utils.trace_cmd.**regex_body_parser**(*regex*, *flags=0*)
Creates an event body parser form the specified regular expression (could be an `re.RegexObject`, or a string). The regular expression should contain some named groups, as those will be extracted as the event attributes (unnamed groups and the reset of the match will be ignored).

If the specified regex is a string, it will be compiled, in which case `flags` may be provided for the resulting regex object (see `re` standard module documentation). If regex is a pre-compiled object, flags will be ignored.

wlauto.utils.trace_cmd.**sched_stat_parser**(*event*, *text*)
sched_stat_* events unclude the units, "[ns]", in an otherwise regular key=value sequence; so the units need to be stripped out first.

wlauto.utils.trace_cmd.**sched_switch_parser**(*event*, *text*)
Sched switch output may be presented in a couple of different formats. One is handled by a regex. The other format can *almost* be handled by the default parser, if it weren't for the ==> that appears in the middle.

wlauto.utils.trace_cmd.**sched_wakeup_parser**(*event*, *text*)

wlauto.utils.trace_cmd.**split_trace_event_line**(*line*)
Split a trace-cmd event line into the preamble (containing the task, cpu id and timestamp), the event name, and the event body. Each of these is delimited by a ': ' (optionally followed by more whitespace), however ': ' may also appear in the body of the event and in the thread name. This attempts to identify the correct split by ensureing the there is a '[' (used to mark the cpu id and not a valid character for a task name) in the peramble.

wlauto.utils.trace_cmd.**try_convert_to_numeric**(*v*)

wlauto.utils.types module

Routines for doing various type conversions. These usually embody some higher-level semantics than are present in standard Python types (e.g. `boolean` will convert the string `"false"` to `False`, where as non-empty strings are usually considered to be `True`).

A lot of these are intened to stpecify type conversions declaratively in place like `Parameter`'s `kind` argument. These are basically "hacks" around the fact that Python is not the best language to use for configuration.

class wlauto.utils.types.**ParameterDict**(**args*, ***kwargs*)
Bases: `dict`

A dict-like object that automatically encodes various types into a url safe string, and enforces a single type for the contents in a list. Each value is first prefixed with 2 letters to preserve type when encoding to a string. The format used is "value_type, value_dimension" e.g a 'list of floats' would become 'fl'.

get(*name*)

get_encoded_value(*name*)

iter_encoded_items()

iteritems()

pop(*key*)

popitem()

update(**args*, ***kwargs*)

values()

class wlauto.utils.types.**arguments**(*value=None*)
Bases: `list`

Represents command line arguments to be passed to a program.

append(*value*)

extend(*values*)

wlauto.utils.types.**boolean**(*value*)
Returns bool represented by the value. This is different from calling the builtin bool() in that it will interpret string representations. e.g. boolean('0') and boolean('false') will both yield False.

class wlauto.utils.types.**caseless_string**
Bases: `str`

Just like built-in Python string except case-insensitive on comparisons. However, the case is preserved otherwise.

format(**args*, ***kwargs*)

wlauto.utils.types.**counter**(*name=None*)
An auto incremeting value (kind of like an AUTO INCREMENT field in SQL). Optionally, the name of the counter to be used is specified (each counter increments separately).

Counts start at 1, not 0.

wlauto.utils.types.**file_path**(*value*)
Handles expansion of paths containing '~'

wlauto.utils.types.**identifier**(*text*)
Converts text to a valid Python identifier by replacing all whitespace and punctuation.

wlauto.utils.types.**integer**(*value*)
Handles conversions for string respresentations of binary, octal and hex.

wlauto.utils.types.**list_of**(*type_*)
Generates a "list of" callable for the specified type. The callable attempts to convert all elements in the passed value to the specifed `type_`, raising `ValueError` on error.

wlauto.utils.types.**list_of_bools**(*value*, *interpret_strings=True*)
Value must be iterable. All elements will be converted to `bool`s.

Note: By default, `boolean()` conversion function will be used, which means that strings like `"0"` or `"false"` will be interpreted as `False`. If this is undesirable, set `interpret_strings` to `False`.

---

wlauto.utils.types.**list_of_integers**(*value*)
Value must be iterable. All elements will be converted to `int`s.

wlauto.utils.types.**list_of_ints**(*value*)
Value must be iterable. All elements will be converted to `int`s.

wlauto.utils.types.**list_of_numbers**(*value*)
Value must be iterable. All elements will be converted to numbers (either `int`s or `float`s depending on the elements).

wlauto.utils.types.**list_of_strings**(*value*)
Value must be iterable. All elements will be converted to strings.

wlauto.utils.types.**list_of_strs**(*value*)
Value must be iterable. All elements will be converted to strings.

wlauto.utils.types.**list_or**(*type_*)
Generator for "list or" types. These take either a single value or a list values and return a list of the specfied `type_` performing the conversion on the value (if a single value is specified) or each of the elemented of the specified list.

wlauto.utils.types.**list_or_bool**
alias of `list_or_type`

wlauto.utils.types.**list_or_caseless_string**(*value*)
Converts the value into a list of `caseless_string`'s. If the value is not iterable a one-element list with stringified value will be returned.

wlauto.utils.types.**list_or_integer**
alias of `list_or_type`

wlauto.utils.types.**list_or_number**
alias of list_or_type

wlauto.utils.types.**list_or_string**(*value*)
Converts the value into a list of strings. If the value is not iterable, a one-element list with stringified value will be returned.

wlauto.utils.types.**numeric**(*value*)
Returns the value as number (int if possible, or float otherwise), or raises ValueError if the specified value does not have a straight forward numeric conversion.

class wlauto.utils.types.**range_dict**
Bases: dict

This dict allows you to specify mappings with a range.

If a key is not in the dict it will search downward until the next key and return its value. E.g:

If: a[5] = "Hello" a[10] = "There"

Then: a[2] == None a[7] == "Hello" a[999] == "There"

wlauto.utils.types.**regex**(*value*)
Regular expression. If value is a string, it will be complied with no flags. If you want to specify flags, value must be precompiled.

wlauto.utils.types.**reset_counter**(*name=None*)

wlauto.utils.uboot module

class wlauto.utils.uboot.**UbootMenu**(*conn*, *start_prompt='Hit any key to stop autoboot'*)
Bases: object

Allows navigating Das U-boot menu over serial (it relies on a pexpect connection).

boot()

default_timeout = 60

empty_buffer()

enter(*value*, *delay=1*)
Like select() except no resolution is performed – the value is sent directly to the serial connection.

fixed_uboot_version = '2016.03'

getenv()

invalid_regex = <_sre.SRE_Pattern object>

load_delay = 1

nudge()
Send a little nudge to ensure there is something to read. This is useful when you're not sure if all out put from the serial has been read already.

open(*timeout=60*)
"Open" the UEFI menu by sending an interrupt on STDIN after seeing the starting prompt (configurable upon creation of the UefiMenu object.

option_regex = <_sre.SRE_Pattern object>

prompt_regex = <_sre.SRE_Pattern object>

setenv(*variable*, *value*, *force=False*)

uboot_regex = <_sre.SRE_Pattern object>

write_characters(*line*)

wlauto.utils.uefi module

class wlauto.utils.uefi.**UefiConfig**(*config_dict*)
Bases: object

class wlauto.utils.uefi.**UefiMenu**(*conn*, *prompt='The default boot selection will start in'*)
Bases: object

Allows navigating UEFI menu over serial (it relies on a pexpect connection).

create_entry(*name*, *config*)
Create a new UEFI entry using the parameters. The menu is assumed to be at the top level. Upon return, the menu will be at the top level.

default_timeout = 60

delete_entry(*name*)
Delete the specified UEFI entry. The menu is assumed to be at the top level. Upon return, the menu will be at the top level.

empty_buffer()
Read everything from the serial and clear the internal pexpect buffer. This ensures that the next `expect()` call will time out (unless further input will be sent to the serial beforehand. This is used to create a "known" state and avoid unexpected matches.

enter(*value*, *delay=1*)
Like `select()` except no resolution is performed – the value is sent directly to the serial connection.

get_option_index(*text*, *timeout=60*)
Returns the menu index of the specified option text (uses regex matching). If the option is not in the current menu, `LookupError` will be raised.

has_option(*text*, *timeout=60*)
Returns `True` if at least one of the options in the current menu has matched (using regex) the specified text.

invalid_regex = <_sre.SRE_Pattern object>

load_delay = 1

nudge()
Send a little nudge to ensure there is something to read. This is useful when you're not sure if all out put from the serial has been read already.

open(*timeout=60*)
"Open" the UEFI menu by sending an interrupt on STDIN after seeing the starting prompt (configurable upon creation of the `UefiMenu` object.

option_regex = <_sre.SRE_Pattern object>

prompt_regex = <_sre.SRE_Pattern object>

read_menu(*timeout=60*)
Parse serial output to get the menu options and the following prompt.

select(*option*, *timeout=60*)
Select the specified option from the current menu.

Parameters

- **option** – Could be an `int` index of the option, or a string/regex to match option text against.
- **timeout** – If a non-`int` option is specified, the option list may need need to be parsed (if it hasn't been already), this may block and the timeout is used to cap that , resulting in a `TIMEOUT` exception.
- **delay** – A fixed delay to wait after sending the input to the serial connection. This should be set if input this action is known to result in a long-running operation.

write_characters(*line*)
Write a single line out to serial charcter-by-character. This will ensure that nothing will be dropped for longer lines.

wlauto.utils.uxperf module

class wlauto.utils.uxperf.**UxPerfParser**(*context*, *prefix=''*)
Bases: `object`

Parses logcat messages for UX Performance markers.

UX Performance markers are output from logcat under a debug priority. The logcat tag for the marker messages is UX_PERF. The messages associated with this tag consist of a name for the action to be recorded and a timestamp. These fields are delimited by a single space. e.g.

<TAG> : <MESSAGE> UX_PERF : gestures_swipe_left_start 861975087367 ... ... UX_PERF : gestures_swipe_left_end 862132085804

Timestamps are produced using the running Java Virtual Machine's high-resolution time source, in nanoseconds.

add_action_frames(*frames*, *drop_threshold*, *generate_csv*)
Uses FpsProcessor to parse frame.csv extracting fps, frame count, jank and vsync metrics on a per action basis. Adds results to metrics.

add_action_timings()
Add simple action timings in millisecond resolution to metrics

parse(*log*)
Opens log file and parses UX_PERF markers.

Actions delimited by markers are captured in a dictionary with actions mapped to timestamps.

Module contents

wlauto.workloads package

Subpackages

wlauto.workloads.adobereader package

Module contents

class wlauto.workloads.adobereader.**AdobeReader**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = 'com.adobe.reader.AdobeReader'

aliases = AC([])

artifacts = AC([])

core_modules = []

default_search_strings = ['The quick brown fox jumps over the lazy dog', 'TEST_SEARCH_

deployable_assets = []

description = '\n The Adobe Reader workflow carries out the following typical producti

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'adobereader'

package = 'com.adobe.reader'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

view = ['com.adobe.reader/com.adobe.reader.help.AROnboardingHelpActivity', 'com.adobe.

wlauto.workloads.andebench package

Module contents

class wlauto.workloads.andebench.**Andebench**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = 'com.eembc.coremark.splash'

aliases = AC(['<wlauto.core.extension.Alias object>'])

artifacts = AC([])

core_modules = []

description = '\n AndEBench is an industry standard Android benchmark provided by The

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'andebench'

package = 'com.eembc.coremark'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object>

setup(*context*)

summary_metrics = ['AndEMark Java', 'AndEMark Native']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.androbench package

Module contents

class wlauto.workloads.androbench.**Androbench**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.main'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures the storage performance of an Android device.\n\n Website:

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'androbench'

package = 'com.andromeda.androbench2'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 600

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.angrybirds package

Module contents

class wlauto.workloads.angrybirds.**AngryBirds**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.rovio.ka3d.App'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Angry Birds game.\n\n A very popular Android 2D game.\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'angrybirds'

package = 'com.rovio.angrybirds'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.angrybirds_rio package

Module contents

class wlauto.workloads.angrybirds_rio.**AngryBirdsRio**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.rovio.ka3d.App'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Angry Birds Rio game.\n\n The sequel to the very popular Android 2D

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'angrybirds_rio'

package = 'com.rovio.angrybirdsrio'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.anomaly2 package

Module contents

class wlauto.workloads.anomaly2.**Anomaly2**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.android.Game11Bits.MainActivity'

aliases = AC([])

artifacts = AC([])

asset_file = 'obb:com.elevenbitstudios.anomaly2Benchmark.tar.gz'

core_modules = []

description = '\n Anomaly 2 game demo and benchmark.\n\n Plays three scenes from the g

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

loading_time = 30

name = 'anomaly2'

package = 'com.elevenbitstudios.anomaly2Benchmark'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

reset(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.antutu package

Module contents

class wlauto.workloads.antutu.**Antutu**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.ABenchMarkStart'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n AnTuTu Benchmark is an benchmarking tool for Android Mobile Phone/Pa

device_prefs_directory = '/data/data/com.antutu.ABenchMark/shared_prefs'

device_prefs_file = '/data/data/com.antutu.ABenchMark/shared_prefs/com.antutu.ABenchMa

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

local_prefs_directory = '/home/docs/checkouts/readthedocs.org/user_builds/workload-aut

name = 'antutu'

package = 'com.antutu.ABenchMark'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

summary_metrics = ['score', 'Overall_Score']

update_result(*context*)

valid_versions = ['3.3.2', '4.0.3', '5.3.0', '6.0.1']

validate(**args*, ***kwargs*)

wlauto.workloads.antutu.**extract_metrics**(*fh*)

wlauto.workloads.antutu.**extract_older_version_metrics**(*fh*)

wlauto.workloads.apklaunch package

Module contents

class wlauto.workloads.apklaunch.**ApkLaunchWorkload**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Installs and runs a .apk file, waits wait_time_seconds, and tests if

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'apklaunch'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.applaunch package

Module contents

class wlauto.workloads.applaunch.**Applaunch**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = '\n This workload launches and measures the launch time of applications

finalize(**args*, ***kwargs*)

init_resources(*context*)

init_workload_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'applaunch'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

pass_parameters()

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.appshare package

Module contents

class wlauto.workloads.appshare.**AppShare**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = None

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = '\n Workload to test how responsive a device is when context switching b

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'appshare'

package = []

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = True

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

view = []

wlauto.workloads.audio package

Module contents

class wlauto.workloads.audio.**Audio**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Audio workload plays an MP3 file using the built-in music player. By

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'audio'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.autotest package

Module contents

class wlauto.workloads.autotest.**ChromeAutotest**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Executes tests from ChromeOS autotest suite\n\n .. note:: This work

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'autotest'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['chromeos']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.bbench package

Module contents

class wlauto.workloads.bbench.**BBench**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC(['<wlauto.core.extension.Alias object>'])

artifacts = AC([])

core_modules = []

description = '\n BBench workload opens the built-in browser and navigates to, and\n s

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'bbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

summary_metrics = ['Mean Latency']

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.benchmarkpi package

Module contents

class wlauto.workloads.benchmarkpi.**BenchmarkPi**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.BenchmarkPi'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures the time the target device takes to run and complete the Pi

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'benchmarkpi'

package = 'gr.androiddev.BenchmarkPi'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object at 0x2e92760>

summary_metrics = ['pi calculation']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.blogbench package

Module contents

class wlauto.workloads.blogbench.**Blogbench**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Blogbench is a portable filesystem benchmark that tries to reproduce

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'blogbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.caffeinemark package

Module contents

class wlauto.workloads.caffeinemark.**Caffeinemark**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.Application'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n CaffeineMark is a series of tests that measure the speed of Java\n p

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'caffeinemark'

package = 'com.flexycore.caffeinemark'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object at 0x2ebb8c0>

summary_metrics = ['OverallScore']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.cameracapture package

Module contents

class wlauto.workloads.cameracapture.**Cameracapture**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.common.android.workload.UiAutomatorWorkload*

activity = 'com.android.camera.CameraActivity'

aliases = AC([])

api_packages = {1: 'com.google.android.gallery3d', 23: 'com.google.android.GoogleCam

artifacts = AC([])

core_modules = []

description = '\n Uses in-built Android camera app to take photos.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'cameracapture'

package = 'com.google.android.gallery3d'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.camerarecord package

Module contents

class wlauto.workloads.camerarecord.**Camerarecord**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.common.android.workload.UiAutomatorWorkload*

activity = 'com.android.camera.CameraActivity'

aliases = AC([])

api_packages = {1: 'com.google.android.gallery3d', 23: 'com.google.android.GoogleCam

artifacts = AC([])

camera_modes = ['normal', 'slow_motion']

core_modules = []

description = '\n Uses in-built Android camera app to record the video for given inter

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'camerarecord'

package = 'com.google.android.gallery3d'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 0

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.castlebuilder package

Module contents

class wlauto.workloads.castlebuilder.**Castlebuilder**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.unity3d.player.UnityPlayerProxyActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Castle Builder game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'castlebuilder'

package = 'com.ettinentertainment.castlebuilder'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.castlemaster package

Module contents

class wlauto.workloads.castlemaster.**CastleMaster**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.unity3d.player.UnityPlayerActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Castle Master v1.09 game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 500

kind = 'workload'

name = 'castlemaster'

package = 'com.alphacloud.castlemaster'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.cfbench package

Module contents

class wlauto.workloads.cfbench.**Cfbench**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.MainActivity'

aliases = AC([])

artifacts = AC([])

cfbench_params = ['java_mdflops', 'native_memory_read', 'java_msflops', 'native_disk_r

core_modules = []

description = '\n CF-Bench is (mainly) CPU and memory benchmark tool specifically desi

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'cfbench'

package = 'eu.chainfire.cfbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 300

summary_metrics = ['overall_score']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.citadel package

Module contents

class wlauto.workloads.citadel.**EpicCitadel**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = '.UE3JavaApp'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Epic Citadel demo showcasing Unreal Engine 3.\n\n The game has very

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 120

kind = 'workload'

name = 'citadel'

package = 'com.epicgames.EpicCitadel'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.cyclictest package

Module contents

class wlauto.workloads.cyclictest.**Cyclictest**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures the amount of time that passes between when a timer expires

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'cyclictest'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.dex2oat package

Module contents

class wlauto.workloads.dex2oat.**Dex2oatBenchmark**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

command_template = 'dex2oat --dex-file={} --oat-file={} --instruction-set={} --dump-ti

core_modules = []

description = '\n Benchmarks the execution time of dex2oat (a key part of APK installa

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'dex2oat'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

run_timeout = 300

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

Retrieve the last dex2oat time from the logs. That will correspond with the run() method. The compilation time does not.

Pulls out the compilation time and dex2oat execution time: I/dex2oat (2522): 1.8s Compile Dex File I/dex2oat (2522): dex2oat took 2.366s (threads: 6)

validate(**args*, ***kwargs*)

wlauto.workloads.dhrystone package

Module contents

class wlauto.workloads.dhrystone.**Dhrystone**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

bm_regex = <_sre.SRE_Pattern object at 0x2ee6b10>

core_modules = []

default_mloops = 100

description = '\n Runs the Dhrystone benchmark.\n\n Original source from::\n\n http://

dmips_regex = <_sre.SRE_Pattern object>

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'dhrystone'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

time_regex = <_sre.SRE_Pattern object at 0x2ecc320>

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.dungeondefenders package

Module contents

class wlauto.workloads.dungeondefenders.**DungeonDefenders**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.trendy.ddapp.ddapp'

aliases = AC([])

artifacts = AC([])

asset_file = 'com.trendy.ddapp.tar.gz'

core_modules = []

description = '\n Dungeon Defenders game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

loading_time = 20

name = 'dungeondefenders'

package = 'com.trendy.ddapp'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.ebizzy package

Module contents

class wlauto.workloads.ebizzy.**Ebizzy**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n ebizzy is designed to generate a workload resembling common web\n ap

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'ebizzy'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.facebook package

Module contents

class wlauto.workloads.facebook.**Facebook**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

DELAY = 5

activity = '.LoginActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Uses com.facebook.patana apk for facebook workload.\n This workload

du_activity = 'com.android.vending/.AssetBrowserActivity'

du_apk_file = '/disableupdateapk/com.android.vending-4.3.10.apk'

du_jar_file = '/data/local/wa_usecases/com.arm.wlauto.uiauto.facebook.apk'

du_method_string = 'com.arm.wlauto.uiauto.facebook.UiAutomation#disableUpdate'

du_run_timeout = 240

du_working_dir = '/data/local/wa_usecases'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

max_apk_version = '3.4'

name = 'facebook'

package = 'com.facebook.katana'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.geekbench package

Module contents

class wlauto.workloads.geekbench.**GBScoreCalculator**
Bases: object

Parses logcat output to extract raw Geekbench workload values and converts them into category and overall scores.

category_weights = {'integer': 0.3357231, 'float': 0.3594, 'stream': 0.1054738, 'mem

parse(*filepath*)
Extract results from the specified file. The file should contain a logcat log of Geekbench execution. Iteration results in the log appear as 'I/geekbench' category entries in the following format:

```
|                      worklod ID           value      units    timing
|                          \-------------     |      ----/      ---/
|                                           |     |      |          |
|  I/geekbench(29026): [....] workload 101 132.9 MB/sec 0.0300939s
|       |                |
|       |                -----          |      label    random crap we don't␣
↪care about
```

result_regex = <_sre.SRE_Pattern object at 0x2efcce0>

update_results(*context*)
http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-2-scores

From the website:

Each workload's performance is compared against a baseline to determine a score. These scores are averaged together to determine an overall, or Geekbench, score for the system.

Geekbench uses the 2003 entry-level Power Mac G5 as the baseline with a score of 1,000 points. Higher scores are better, with double the score indicating double the performance.

Geekbench provides three different kinds of scores:

Workload Scores Each time a workload is executed Geekbench calculates a score based on the computer's performance compared to the baseline performance. There can be multiple workload scores for the same workload as Geekbench can execute each workload multiple times with different settings. For example, the "Dot Product" workload is executed four times (single-threaded scalar code, multi-threaded scalar code, single-threaded vector code, and multi-threaded vector code) producing four "Dot Product" scores.

Section Scores A section score is the average of all the workload scores for workloads that are part of the section. These scores are useful for determining the performance of the computer in a particular area. See the section descriptions above for a summary on what each section measures.

Geekbench Score The Geekbench score is the weighted average of the four section scores. The Geekbench score provides a way to quickly compare performance across different computers and different platforms without getting bogged down in details.

workloads = [Blowfish, Text Compress, Text Decompress, Image Compress, Image Decompress

class wlauto.workloads.geekbench.**GBWorkload**(*wlid*, *name*, *pmac_g5_st_score*, *pmac_g5_mt_score*)
Bases: object

Geekbench workload (not to be confused with WA's workloads). This is a single test run by geek bench, such as preforming compression or generating Madelbrot.

add_result(*value*, *units*)

categories = [None, 'integer', 'float', 'memory', 'stream']

clear()

convert_to_kilo(*value*, *units*)

get_scores()

Returns a tuple (single-thraded score, multi-threaded score) for this workload. Some workloads only have a single-threaded score, in which case multi-threaded score will be `None`.

Geekbench will perform four iterations of each workload in single-threaded and, for some workloads, multi-threaded configurations. Thus there should always be either four or eight scores collected for each workload. Single-threaded iterations are always done before multi-threaded, so the ordering of the scores can be used to determine which configuration they belong to.

This method should not be called before score collection has finished.

pmac_g5_base_score = 1000

units_conversion_map = {'K': 1, 'M': 1000, 'G': 1000000}

class wlauto.workloads.geekbench.**Geekbench**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity

aliases = AC([])

artifacts = AC([])

begin_regex = <_sre.SRE_Pattern object at 0x2ef6f70>

core_modules = []

description = '\n Geekbench provides a comprehensive set of benchmarks engineered to q

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

is_corporate = False

kind = 'workload'

name = 'geekbench'

package

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

replace_regex = <_sre.SRE_Pattern object>

summary_metrics = ['score', 'multicore_score']

update_result(*context*)

update_result_2(*context*)

update_result_3(*context*)

update_result_4(*context*)

validate(**args*, ***kwargs*)

versions = {'4.0.1': {'activity': '.HomeActivity', 'package': 'com.primatelabs.geekl

class wlauto.workloads.geekbench.**GeekbenchCorproate**(*device*, ***kwargs*)

Bases: *wlauto.workloads.geekbench.Geekbench*

activity = 'com.primatelabs.geekbench.HomeActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

is_corporate = True

kind = 'workload'

name = 'geekbench-corporate'

package = 'com.primatelabs.geekbench4.corporate'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

versions = ['4.1.0']

wlauto.workloads.geekbench.**namemify**(*basename*, *i*)

wlauto.workloads.geekbench.**versiontuple**(*v*)

wlauto.workloads.glbcorp package

Module contents

class wlauto.workloads.glbcorp.**GlbCorp**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.common.android.workload.ApkWorkload*

activity = 'net.kishonti.benchui.TestActivity'

aliases = AC(['<wlauto.core.extension.Alias object>', '<wlauto.core.extension.Alias ob

artifacts = AC([])

core_modules = []

description = '\n GFXBench GL (a.k.a. GLBench) v3.0 Corporate version.\n\n This is a v

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

launch_package()

name = 'glb_corporate'

package = 'net.kishonti.gfxbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

preamble_regex = None

result_start_regex = None

run(*context*)

setup(*context*)

supported_resolutions = {'720p': {'-ei -h': 720, '-ei -w': 1280}, '1080p': {'-ei -

update_result(*context*)

valid_test_ids = ['gl_alu', 'gl_alu_off', 'gl_blending', 'gl_blending_off', 'gl_driver

validate(**args*, ***kwargs*)

class wlauto.workloads.glbcorp.**GlbRunMonitor**(*device*)
Bases: threading.Thread

new_regex = <_sre.SRE_Pattern object>

old_regex = <_sre.SRE_Pattern object>

run()

stop()

wait_for_run_end(*timeout*)

wlauto.workloads.glbenchmark package

Module contents

class wlauto.workloads.glbenchmark.**Glb**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = 'com.glbenchmark.activities.GLBenchmarkDownloaderActivity'

aliases = AC(['<wlauto.core.extension.Alias object>', '<wlauto.core.extension.Alias ob

artifacts = AC([])

core_modules = []

default_iterations = 1

description = '\n Measures the graphics performance of Android devices by testing\n th

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 500

kind = 'workload'

name = 'glbenchmark'

packages = {'2.7': 'com.glbenchmark.glbenchmark27', '2.5': 'com.glbenchmark.glbenchma

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object>

supported_usecase_aliases = {'2.7': ['t-rex', 'egypt'], '2.5': ['egypt-classic', 'eg

update_result(*context*)

validate(**args*, ***kwargs*)

view = 'com.glbenchmark.glbenchmark27/com.glbenchmark.activities.GLBRender'

wlauto.workloads.gmail package

Module contents

class wlauto.workloads.gmail.**Gmail**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = ''

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = '\n A workload to perform standard productivity tasks within Gmail. The

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'gmail'

package = 'com.google.android.gm'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = True

validate(**args*, ***kwargs*)

view = ['com.google.android.gm/com.google.android.gm.ConversationListActivityGmail', '

wlauto.workloads.googlemap package

Module contents

class wlauto.workloads.googlemap.**GoogleMap**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.google.android.maps.MapsActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Navigation app.\n\n Stock map provided by Google Inc.\n Based on rev

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

loading_time = 20

name = 'googlemap'

package = 'com.google.android.apps.maps'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.googlephotos package

Module contents

class wlauto.workloads.googlephotos.**Googlephotos**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = 'com.google.android.apps.photos.home.HomeActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

default_test_images = ['uxperf_1200x1600.png', 'uxperf_1600x1200.jpg', 'uxperf_2448x32

deployable_assets = []

description = '\n A workload to perform standard productivity tasks with Google Photos

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'googlephotos'

package = 'com.google.android.apps.photos'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

view = ['com.google.android.apps.photos/com.google.android.apps.consumerphotoeditor.fr

wlauto.workloads.googleplaybooks package

Module contents

class wlauto.workloads.googleplaybooks.**Googleplaybooks**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = 'com.google.android.apps.books.app.BooksActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = "\n A workload to perform standard productivity tasks with googleplaybooks

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'googleplaybooks'

package = 'com.google.android.apps.books'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = True

validate(**args*, ***kwargs*)

view = ['com.google.android.apps.books/com.google.android.apps.books.app.HomeActivity'

wlauto.workloads.googleslides package

Module contents

class wlauto.workloads.googleslides.**GoogleSlides**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = ''

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = '\n A workload to perform standard productivity tasks with Google Slides

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'googleslides'

new_doc_name = 'WORKLOAD AUTOMATION'

package = 'com.google.android.apps.docs.editors.slides'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

teardown(*context*)

validate(**args*, ***kwargs*)

view = ['com.google.android.apps.docs.editors.slides/com.google.android.apps.docs.quic

wlauto.workloads.gunbros2 package

Module contents

class wlauto.workloads.gunbros2.**GunBros**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.google.android.vending.expansion.downloader_impl.DownloaderActivity'

aliases = AC([])

artifacts = AC([])

asset_file = 'com.glu.gunbros2.tar.gz'

core_modules = []

description = '\n Gun Bros. 2 game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 500

kind = 'workload'

loading_time = 20

name = 'gunbros2'

ondevice_asset_root = '/data'

package = 'com.glu.gunbros2'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.hackbench package

Module contents

class wlauto.workloads.hackbench.**Hackbench**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Hackbench runs a series of tests for the Linux scheduler.\n\n For de

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'hackbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.homescreen package

Module contents

class wlauto.workloads.homescreen.**HomeScreen**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n A workload that goes to the home screen and idles for the the\n spec

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'homescreen'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

validate(**args*, ***kwargs*)

wlauto.workloads.hwuitest package

Module contents

class wlauto.workloads.hwuitest.**HWUITest**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Tests UI rendering latency on android devices\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'hwuitest'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.idle package

Module contents

class wlauto.workloads.idle.**IdleWorkload**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Do nothing for the specified duration.\n\n On android devices, this

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'idle'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.iozone package

Module contents

class wlauto.workloads.iozone.**Iozone**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Iozone is a filesystem benchmark that runs a series of disk\n I/O pe

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'iozone'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

parse_metrics(*context*, *plist*)

parse_thread_results()

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

write_to_csv(*context*, *csv_table_list*)

wlauto.workloads.ironman package

Module contents

class wlauto.workloads.ironman.**IronMan**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = '.GameActivity'

aliases = AC([])

artifacts = AC([])

asset_file = 'obb:com.gameloft.android.ANMP.GloftIMHM.tar.gz'

core_modules = []

description = '\n Iron Man 3 game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'ironman3'

package = 'com.gameloft.android.ANMP.GloftIMHM'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.krazykart package

Module contents

class wlauto.workloads.krazykart.**KrazyKartRacing**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = '.krazyracers'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Krazy Kart Racing game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'krazykart'

package = 'com.polarbit.sg2.krazyracers'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.linpack package

Module contents

class wlauto.workloads.linpack.**Linpack**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.Linpack'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n The LINPACK Benchmarks are a measure of a system's floating point co

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'linpack'

package = 'com.greenecomputing.linpackpro'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object>

summary_metrics = ['Linpack ST', 'Linpack MT']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.linpack_cli package

Module contents

class wlauto.workloads.linpack_cli.**LinpackCliWorkload**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

binary = None

core_modules = []

description = '\n linpack benchmark with a command line interface\n\n Benchmarks FLOPS

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'linpack-cli'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.lmbench package

Module contents

class wlauto.workloads.lmbench.**lmbench**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Run a subtest from lmbench, a suite of portable ANSI/C microbenchmar

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'lmbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

test_names = ['lat_mem_rd', 'bw_mem']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.manual package

Module contents

class wlauto.workloads.manual.**ManualWorkload**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Yields control to the user, either for a fixed period or based on us

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'manual'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.workloads.manual.**ManualWorkloadConfig**(*duration=None*, *user_triggered=None*, *view=None*, *package=None*, *enable_logcat=True*)

Bases: object

default_duration = 30

wlauto.workloads.memcpy package

Module contents

class wlauto.workloads.memcpy.**MemcpyTest**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Runs memcpy in a loop.\n\n This will run memcpy in a loop for a spec

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'memcpy'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.nenamark package

Module contents

class wlauto.workloads.nenamark.**Nenamark**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.common.android.workload.ApkWorkload*

activity = 'se.nena.nenamark2.NenaMark2'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n NenaMark is an OpenGL-ES 2.0 graphics performance benchmark for Andr

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'nenamark'

package = 'se.nena.nenamark2'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

regex = <_sre.SRE_Pattern object>

run(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.octaned8 package

Module contents

class wlauto.workloads.octaned8.**Octaned8**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Runs the Octane d8 benchmark.\n\n This workload runs d8 binaries bui

executables = ['d8', 'natives_blob.bin', 'snapshot_blob.bin']

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'octaned8'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.peacekeeper package

Module contents

class wlauto.workloads.peacekeeper.**Peacekeeper**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Peacekeeper is a free and fast browser test that measures a browser'

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'peacekeeper'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 900

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.workloads.peacekeeper.**PeacekeeperParser**
Bases: HTMLParser.HTMLParser

handle_data(*data*)

handle_endtag(*tag*)

handle_starttag(*tag*, *attrs*)

wlauto.workloads.power_loadtest package

Module contents

class wlauto.workloads.power_loadtest.**PowerLoadtest**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n power_LoadTest (part of ChromeOS autotest suite) continuously cycles

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'power_loadtest'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['chromeos']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.quadrant package

Module contents

class wlauto.workloads.quadrant.**Quadrant**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.QuadrantProfessionalLauncherActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Quadrant is a benchmark for mobile devices, capable of measuring CPU

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'quadrant'

package = 'com.aurorasoftworks.quadrant.ui.professional'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 600

setup(*context*)

summary_metrics = ['benchmark_score']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.real_linpack package

Module contents

class wlauto.workloads.real_linpack.**RealLinpack**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.RealLinpackActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n This version of `Linpack <http://en.wikipedia.org/wiki/LINPACK_benchm

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'real-linpack'

package = 'com.arm.RealLinpack'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.realracing3 package

Module contents

class wlauto.workloads.realracing3.**RealRacing3**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.firemint.realracing3.MainActivity'

aliases = AC([])

artifacts = AC([])

asset_file = 'com.ea.games.r3_row.tar.gz'

core_modules = []

description = '\n Real Racing 3 game.\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

loading_time = 90

name = 'realracing3'

package = 'com.ea.games.r3_row'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

saved_state_file = 'rr3-save.tar.gz'

validate(**args*, ***kwargs*)

wlauto.workloads.recentfling package

Module contents

class wlauto.workloads.recentfling.**Recentfling**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Tests UI jank on android devices.\n\n For this workload to work, ``r

finalize(**args*, ***kwargs*)

initialise(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'recentfling'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.rt_app package

Module contents

class wlauto.workloads.rt_app.**RtApp**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n A test application that simulates configurable real-time periodic lo

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'rt-app'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.shellscript package

Module contents

class wlauto.workloads.shellscript.**ShellScript**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Runs an arbitrary shellscript on the device.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'shellscript'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.skype package

Module contents

class wlauto.workloads.skype.**Skype**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = ''

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = "\n A workload to perform standard productivity tasks within Skype. The

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

launch_app()

launch_main = False

name = 'skype'

package = 'com.skype.raider'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = True

setup(*context*)

validate(**args*, ***kwargs*)

view = ['com.skype.raider/com.skype.android.app.calling.CallActivity', 'com.skype.raid

wlauto.workloads.smartbench package

Module contents

class wlauto.workloads.smartbench.**Smartbench**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.Smartbench2012'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Smartbench is a multi-core friendly benchmark application that measu

finalize(**args*, ***kwargs*)

game_regex = <_sre.SRE_Pattern object>

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'smartbench'

package = 'com.smartbench.twelve'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

prod_regex = <_sre.SRE_Pattern object>

run_timeout = 600

summary_metrics = ['Smartbench: valueGame', 'Smartbench: valueProd']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.spec2000 package

Module contents

class wlauto.workloads.spec2000.**CommandSpec**
Bases: object

class wlauto.workloads.spec2000.**Spec2000**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC(['<wlauto.core.extension.Alias object>'])

artifacts = AC([])

asset_file = 'spec2000-assets.tar.gz'

core_modules = []

description = '\n SPEC2000 benchmarks measuring processor, memory and compiler.\n\n ht

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

launch_template = '{busybox} taskset {cpumask} {command} 1>/dev/null 2>&1'

loop_template = 'for i in $({busybox} seq 1 {threads}); do {launch_command} 1>/dev/nul

name = 'spec2000'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

rate_run_template = 'cd {datadir}; time ({loop}; wait)'

run(*context*)

setup(*context*)

speed_run_template = 'cd {datadir}; time ({launch_command})'

summary_metrics = <wlauto.workloads.spec2000._SPECSummaryMetrics object>

timing_regex = <_sre.SRE_Pattern object>

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.workloads.spec2000.**SpecBenchmark**
Bases: object

get_binary(*cpu*)

wlauto.workloads.sqlite package

Module contents

class wlauto.workloads.sqlite.**Sqlite**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.Main'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures the performance of the sqlite database. It determines withi

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'sqlitebm'

package = 'com.redlicense.benchmark.sqlite'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

score_regex = <_sre.SRE_Pattern object>

summary_metrics = ['Overall']

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.stream package

Module contents

class wlauto.workloads.stream.**Stream**(*device*, ***kwargs*)
Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Measures memory bandwidth.\n\n The original source code be found on:

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'stream'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.stress_ng package

Module contents

class wlauto.workloads.stress_ng.**StressNg**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n stress-ng will stress test a computer system in various selectable w

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'stress_ng'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.sysbench package

Module contents

class wlauto.workloads.sysbench.**Sysbench**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n SysBench is a modular, cross-platform and multi-threaded benchmark t

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'sysbench'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.sysbench.**extract_metric**(*metric*, *line*, *result*, *prefix=''*, *lower_is_better=True*)

wlauto.workloads.sysbench.**extract_threads_fairness_metric**(*metric*, *line*, *result*)

wlauto.workloads.sysbench.**find_line_with**(*text*, *fh*)

wlauto.workloads.telemetry package

Module contents

class wlauto.workloads.telemetry.**Telemetry**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC([])

artifacts = AC([])

build_command()

core_modules = []

description = "\n Executes Google's Telemetery benchmarking framework\n\n Url: https:

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'telemetry'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android', 'chromeos']

update_result(*context*)

validate(**args*, ***kwargs*)

class wlauto.workloads.telemetry.**TelemetryResult**(*kind=None*, *url=None*, *values=None*, *units=None*)

Bases: object

average

rows

std

wlauto.workloads.telemetry.**parse_telemetry_results**(*filepath*)

wlauto.workloads.templerun package

Module contents

class wlauto.workloads.templerun.**Templerun**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.unity3d.player.UnityPlayerProxyActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Templerun game.\n\n '

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 500

kind = 'workload'

name = 'templerun'

package = 'com.imangi.templerun'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.thechase package

Module contents

class wlauto.workloads.thechase.**TheChase**(*device*, *_call_super=True*, ***kwargs*)

Bases: *wlauto.common.android.workload.ApkWorkload*

activity = 'com.unity3d.player.UnityPlayerNativeActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n The Chase demo showcasing the capabilities of Unity game engine.\n\n

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

install_timeout = 200

kind = 'workload'

name = 'thechase'

package = 'com.unity3d.TheChase'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

validate(**args*, ***kwargs*)

view = 'SurfaceView'

wlauto.workloads.truckerparking3d package

Module contents

class wlauto.workloads.truckerparking3d.**TruckerParking3D**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.GameWorkload*

activity = 'com.tapinator.truck.parking.bus3d.GCMNotificationActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = "\n Trucker Parking 3D game.\n\n (yes, apparently that's a thing...)\n "

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'truckerparking3d'

package = 'com.tapinator.truck.parking.bus3d'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.vellamo package

Module contents

class wlauto.workloads.vellamo.**Vellamo**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

aliases = AC([])

artifacts = AC([])

benchmark_types = {'3.2.4': ['Browser', 'Metal', 'Multi'], '2.0.3': ['html5', 'metal

core_modules = []

description = '\n Android benchmark designed by Qualcomm.\n\n Vellamo began as a mobil

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'vellamo'

non_root_update_result(*context*)

package = 'com.quicinc.vellamo'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run_timeout = 900

setup(*context*)

summary_metrics = None

update_result(*context*)

update_result_v3(*context*)

update_result_v3_2(*context*)

valid_versions = ['3.2.4', '2.0.3', '3.0']

validate(**args*, ***kwargs*)

class wlauto.workloads.vellamo.**VellamoResult**(*name*)

Bases: object

add_metric(*data*)

class wlauto.workloads.vellamo.**VellamoResultParser**

Bases: HTMLParser.HTMLParser

exception StopParsingException

Bases: exceptions.Exception

feed(*text*)

handle_data(*data*)

handle_endtag(*tag*)

handle_starttag(*tag*, *attrs*)

wlauto.workloads.video package

Module contents

class wlauto.workloads.video.**VideoWorkload**(*device*, ***kwargs*)

Bases: *wlauto.core.workload.Workload*

aliases = AC(['<wlauto.core.extension.Alias object>', '<wlauto.core.extension.Alias ob

artifacts = AC([])

core_modules = []

description = '\n Plays a video file using the standard android video player for a pre

enum_video_files()

finalize(**args*, ***kwargs*)

host_video_file

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'video'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

run(*context*)

setup(*context*)

supported_platforms = ['android']

teardown(*context*)

update_result(*context*)

validate(**args*, ***kwargs*)

wlauto.workloads.videostreaming package

Module contents

class wlauto.workloads.videostreaming.**Videostreaming**(*device*, ***kwargs*)

Bases: *wlauto.common.android.workload.AndroidUiAutoBenchmark*

activity = '.MainActivity'

aliases = AC([])

artifacts = AC([])

core_modules = []

description = '\n Uses the FREEdi video player to search, stream and play the specifie

finalize(**args*, ***kwargs*)

init_resources(*context*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'videostreaming'

package = 'tw.com.freedi.youtube.player'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

validate(**args*, ***kwargs*)

wlauto.workloads.youtube package

Module contents

class wlauto.workloads.youtube.**Youtube**(*device*, ***kwargs*)
Bases: *wlauto.common.android.workload.AndroidUxPerfWorkload*

activity = ''

aliases = AC([])

artifacts = AC([])

core_modules = []

deployable_assets = []

description = "\n A workload to perform standard productivity tasks within YouTube.\n

finalize(**args*, ***kwargs*)

initialize(**args*, ***kwargs*)

kind = 'workload'

name = 'youtube'

package = 'com.google.android.youtube'

parameters = AC(["Param({'kind': <type 'list'>, 'mandatory': None, 'name': 'modules

requires_network = True

validate(**args*, ***kwargs*)

view = ['com.google.android.youtube/com.google.android.apps.youtube.app.WatchWhileActi

Module contents

Submodules

wlauto.config_example module

Default config for Workload Automation. DO NOT MODIFY this file. This file gets copied to ~/.workload_automation/config.py on initial run of run_workloads. Add your configuration to that file instead.

wlauto.exceptions module

exception wlauto.exceptions.**CommandError**
Bases: *wlauto.exceptions.WAError*

Raised by commands when they have encountered an error condition during execution.

exception wlauto.exceptions.**ConfigError**
Bases: *wlauto.exceptions.WAError*

Raised when configuration provided is invalid. This error suggests that the user should modify their config and try again.

exception wlauto.exceptions.**DeviceError**
Bases: *wlauto.exceptions.WAError*

General Device error.

exception wlauto.exceptions.**DeviceNotRespondingError**(*device*)
Bases: *wlauto.exceptions.WAError*

The device is not responding.

exception wlauto.exceptions.**HostError**
Bases: *wlauto.exceptions.WAError*

Problem with the host on which WA is running.

exception wlauto.exceptions.**InstrumentError**
Bases: *wlauto.exceptions.WAError*

General Instrument error.

exception wlauto.exceptions.**LoaderError**(*message*, *exc_info=None*)
Bases: *wlauto.exceptions.WAError*

Raised when there is an error loading an extension or an external resource. Apart form the usual message, the __init__ takes an exc_info parameter which should be the result of sys.exc_info() for the original exception (if any) that caused the error.

exception wlauto.exceptions.**ModuleError**
Bases: *wlauto.exceptions.WAError*

Problem with a module.

Note: Modules for specific extension types should raise execeptions appropriate to that extension. E.g. a `Device` module should raise `DeviceError`. This is intended for situation where a module is unsure (and/or doesn't care) what its owner is.

---

exception wlauto.exceptions.**NotFoundError**
Bases: *wlauto.exceptions.WAError*

Raised when the specified item is not found.

exception wlauto.exceptions.**ResourceError**
Bases: *wlauto.exceptions.WAError*

General Resolver error.

exception wlauto.exceptions.**ResultProcessorError**
Bases: *wlauto.exceptions.WAError*

General ResultProcessor error.

exception wlauto.exceptions.**ToolError**
Bases: *wlauto.exceptions.WAError*

Raised by tools when they have encountered an error condition during execution.

exception wlauto.exceptions.**ValidationError**
Bases: *wlauto.exceptions.WAError*

Raised on failure to validate an extension.

exception wlauto.exceptions.**WAError**
Bases: exceptions.Exception

Base class for all Workload Automation exceptions.

exception wlauto.exceptions.**WorkerThreadError**(*thread*, *exc_info*)
Bases: *wlauto.exceptions.WAError*

This should get raised in the main thread if a non-WAError-derived exception occurs on a worker/background thread. If a WAError-derived exception is raised in the worker, then it that exception should be re-raised on the main thread directly – the main point of this is to preserve the backtrace in the output, and backtrace doesn't get output for WAErrors.

exception wlauto.exceptions.**WorkloadError**
Bases: *wlauto.exceptions.WAError*

General Workload error.

Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

W

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y