

Util

Package: `e2e/src/test/java/eu/nets/test/util`

This package contains utility classes, used to handle test environment (operating system, emulator/simulator, global variables, etc.) and external tools (emails, reports, etc.)

AllureUtil

Utility helper for integrating [Allure reporting](#) into our test automation suite. It provides methods to log steps, attach artifacts (e.g., logs, screenshots, HTML), manage the results folder, and generate custom reports.

This class is designed to simplify and centralize interaction with the Allure reporting API, and is meant to be used within test flows, setup/teardown phases, and exception handling blocks.

Responsibilities

- Log test steps and messages to Allure reports.
- Attach different file types to the report (plain text, HTML, images).
- Clear Allure result directories between test runs.
- Create timestamped custom report folders.
- Generate custom Allure reports via CLI, platform-aware (Windows/macOS).
- Handle report generation errors and unsupported OS gracefully.

Fields

- `ALLURE_RESULTS_DIR` : path to the standard Allure results folder (resolved via `PathKey.ALLURE_RESULTS`).
- `CUSTOM_REPORTS_DIR` : base directory for storing timestamped, custom-named reports (resolved via `PathKey.CUSTOM_ALLURE_REPORTS`).

Logging methods

- `logStep(String message)` : logs a single step without logic.
- `logStep(String title, ThrowableRunnableVoid step)` : logs a step that executes a block of code.

- `logInfo(String message)` : logs an informational message to Allure.
- `logWarning(String message)` : logs a warning message to Allure.
- `logError(String message)` : logs an error message to Allure and returns the message (e.g. for rethrowing).

Attachment methods

- `attachText(String name, String content)` : attaches plain text to the report (`text/plain`).
- `attachHtml(String name, String html)` : attaches HTML content (`text/html`).
- `attachImage(String name, String format, byte[] screenshotBytes)` : attaches a binary image (e.g. `png` , `jpg`) with appropriate MIME type.

Folder Management methods

- `emptyAllureResults()` : deletes all contents from the default results directory.
- `emptyCustomAllureReports()` : clears the directory for custom report folders.
- *(Note: previous logic for backing up/restoring `history` has been commented out.)*
- `createCustomReportFolder(String flowName)` :
 - Generates a new timestamped directory inside `CUSTOM_REPORTS_DIR`.
 - Format: `report-{flowName}-yyyyMMdd_HHmmss`.

Report Generation methods

`generateCustomReport(Path customReportDir)` :

- Invokes the Allure CLI to generate a report.
- Uses `--clean` to overwrite existing contents.
- Platform-specific behavior:
 - **Windows**: uses `cmd /c allure generate ...`
 - **macOS**: uses `allure generate ...` directly
- Throws an exception with a clear error message if:
 - The OS is unsupported.

- Report generation fails.
-

EnvUtil

Provides a cross-platform environment utility layer for mobile test automation using Appium.

It offers static helper methods for:

- Detecting OS and architecture
- Starting and stopping Android emulators and iOS simulators
- Preparing and configuring environment variables (especially on macOS)
- Managing the Appium server lifecycle
- Generating platform-specific `DesiredCapabilities` and `XCUITestOptions`
- Handling delays, folders, and ADB operations

Responsibilities

- Handle platform-specific differences for:
 - Emulator/Snapshot names
 - ADB and simulator commands
 - CLI tools (`xcrun` , `simctl` , `emulator` , etc.)
- Start Android emulators with optional snapshot loading on Windows or MacOS
- Start iOS simulators on macOS
- Configure environment variables for Android SDK on MacOS
- Generate and provide `DesiredCapabilities` for Appium (Android and iOS)
- Build and configure Appium server using Appium Java Service
- Support basic utilities:
 - `safeSleep`
 - `emptyFolder`
 - `copyFolder` (delegated to `Files.walk`)

Fields

- `OS` : current operating system name.
- `OS_ARCH` : current system architecture.

- `ADB` : absolute path to the adb binary.
- `EMULATOR` : absolute path to the emulator binary.
- `MAX_TRIES` : maximum number of polling attempts (used for emulator/simulator startup).
- `POLL_INTERVAL_MS` : interval between polling retries in milliseconds (default 1000 ms).

OS and OS Architecture Detection methods

These methods check the host OS and testing target platform using system properties and the [.env file](#).

- Windows: `isWin()`, `isWin32()`, `isWin64()`
- MacOS: `isMac()`, `isMacIntel()`, `isMacAppleSilicon()`
- Mobile: `isAndroid()`, `isIos()`

Android Emulator Management methods

- `startupAndroidEmulator()` : launches emulator without a snapshot.
- `startupAndroidEmulator(AndroidSnapshot snapshot)` : overload of the above, launches emulator with a named snapshot.
- `shutdownAndroidEmulator()` : kills ADB emulator and server.

iOS Simulator Management methods

- `startupIosSimulator()` : boots the simulator via `xcrun simctl boot` and opens the GUI.
- `shutdownIosSimulator()` : shuts down all simulators and closes the GUI.

Appium Server Management methods

- `getAppiumServer()` : builds and returns a default `AppiumDriverLocalService` based on OS.
- `getAppiumServer(String args)` : overload of the above, parses additional CLI arguments.

Android Desired Capabilities and iOS XCUI Test Options methods

- `getDesideredCapabilities` : returns common Appium capabilities (timeout, animation, automation name, permissions, etc.) for `AndroidDriver` configuration.

- `getXCUITestOptions` : returns common Appium options for `IOSDriver` configuration.

MacOS Environment Setup

- `getMacOsEnv()` : generates a full environment map for launching Android tools on MacOS.
- `setProcessBuilderEnv(ProcessBuilder)` : injects the MacOS environment into any ProcessBuilder.

Utility Methods

- `emptyFolder(Path path, boolean deleteSubfolders)` : recursively deletes contents of a folder, optionally including subfolders.
- `safeSleep(int millis)` : wraps `Thread.sleep(...)` with interruption safety and logging.

LokaliseUtil

This class interacts with the [Lokalise API](#) to download language bundles (synchronously or asynchronous) for verify in-app texts within flow tests.

Responsibilities

- Manage the Lokalise ZIP bundle path
- Extract and parse `Localizable.xml` files for multiple languages
- Convert XML to JSON and map strings to key-value dictionaries
- Match UI `WebElement` content with expected Lokalise translations
- Log and throw meaningful exceptions in case of mismatches

Fields

- `API_BASE_URL` : Lokalise API base URL
- `PROJECT_ID` : Lokalise project ID
- `API_PROJECT_URL` : Full Lokalise project URL
- `API_URL_SYNC` : Endpoint for sync file download
- `API_URL_ASYNC` : Endpoint for async file download

- `API_TOKEN_SYNC` : API token for synchronous download
- `API_TOKEN_ASYNC` : API token for asynchronous download
- `BUNDLE_FILEPATH` : Path to the saved Lokalise ZIP bundle

Bundle File and Content Handling methods

- `getBundleFilepath` : returns the current path to the downloaded Lokalise ZIP bundle.
- `setBundleFilepath` : sets the path to the Lokalise ZIP bundle using the provided filename.
- `downloadBundle` : downloads the Lokalise translation bundle using the API. Supports both sync and async APIs.
- `readLanguageFile` : opens the Lokalise ZIP bundle and reads the `Localizable.xml` file for the given language. Converts XML content to a `JSONObject`.
- `getBundle` : extracts all translations from the bundle across all supported `MpaLanguage` values. Returns a language-to-translations map (sorted by `TreeMap`).
- `getContentByName` : finds a translation entry in a Lokalise JSON array by `name`. Returns the corresponding `content` value.
- `match` : Verifies that the text of a `WebElement` matches the expected Lokalise translation.

MailUtil

This class is used to connect to test inboxes and handle related folder, messages and their contents.

Responsibilities

- Connect to an IMAP mail server using credentials.
- Retrieve and parse OTP (One Time Password) emails.
- Extract a 6-digit OTP code from email body.
- Optionally delete the OTP email after extraction.
- Provide basic email filtering and HTML-stripping functionality.

Fields

- `INBOX` : the Folder object pointing to the user's INBOX.
- `STORE` : the IMAP Store object used to connect to the mail server.

OTP methods

`getOtp` : fetch the latest unread OTP email for a user and extract the 6-digit code.

⚠ Disable any VPN before trying to obtain OTP from an inbox.

Message Handling methods

`filterAndSort` filter unread messages intended for the specified user and sort them by descending date. Validates that recipient address matches user email.

`getContentText` : convert message content into a raw string, regardless of MIME structure.

`getTextFromMimeMultipart` : recursively extract readable text content from a complex MIME multipart object.

Inbox Connection methods

`getSession` : onfigure and create a secure `javax.mail.Session` instance based on email provider.

PropertiesUtil

Utility class responsible for loading and accessing configuration properties and test data used in automated test environments. It supports global variables through `.properties` files and environment configuration through `.env` files, enabling centralized management of platform-specific and project-specific variables.

Responsibilities

- Load `.properties` files from the classpath and return them as `Properties` objects.
- Load `.env` files from the filesystem, extract platform-specific variables (e.g., `ANDROID_` , `IOS_`) based on the selected platform, and return a filtered `Properties` object.
- Provide convenient `getProperty` methods for reading values from files using default or specified charset encodings.

- Normalize and sanitize `.env` keys by removing the platform prefix (e.g., `ANDROID_emulatorDeviceName` → `emulatorDeviceName`) to simplify property access during runtime.
- Ensure cross-platform compatibility (macOS and Windows only) for `.env` file resolution.
- Integrate with `AllureUtil` to log meaningful error messages when files are missing or corrupted.

Fields

- `CONFIG` : loads the `config.properties` file located in the `properties/` directory. Typically includes general configuration for test execution.
- `MPA` : loads the `mpa.properties` file, containing values related to the MPA (My Payments App) test application.
- `ENV` : loads and filters environment-specific variables from a [.env file](#). It parses platform-specific variables and makes them accessible in a uniform way.

Methods

- `loadDotProperties` :
Loads a `.properties` file from the classpath and returns its contents.
- `getProperty` : loads a `.properties` file and retrieves the value of the specified key.
- `loadDotEnv()` : loads the `.env` file from the root directory, filters keys based on the current platform, and exposes them with normalized names. Also ensures the `PLATFORM` value is present and valid.

`.env` file


This file serves as a configuration template for setting up environment variables used by automated test frameworks. It defines platform-specific properties required to run Android or iOS mobile tests via Appium or similar tools.

How to use it:


- Make a copy of the file `.env.example` located in `e2e` repository module and rename it to `.env`
- Fill in the required values marked with placeholders (e.g. `***INSERT_PLATFORM_HERE***`) and change the preset values based on your requirements, if necessary.


Responsibilities of the File:

- Acts as the primary entry point for environment setup in cross-platform mobile testing.
- Enables dynamic resolution of platform-specific configuration through prefix-based keys.
- Supports both local emulators/simulators and physical device execution.
- Designed for integration with classes like `PropertiesUtil` and `EnvUtil` which parse, validate, and consume these variables. Can be integrated with any existing or new class which requires these variables.

 Commit only `.env.example` to the repository, when field changes are required.

Local `.env` file should never be added to the repository and committed, especially if it contains real device credentials or personal UDIDs.

 Any missing or malformed entries may result in runtime exceptions when the test framework loads environment configurations.

 You may extend the file to include more platform-specific or test-specific options if needed.
