Gauntlet Automation Framework Overview

Framework to run sessions of projects in Unreal Engine that perform tests and validate results.

Gauntlet is a framework to run sessions of projects in **Unreal Engine** that perform tests and validate results. It is specifically designed for, but not limited to, running Unreal sessions on a variety of platforms. An Unreal **session** is all of the processes needed to execute a game with the Unreal engine. For example, a multiplayer game may require four clients and a server.

Gauntlet does not require any specific game-side automation code or test framework — how your game performs tests is entirely up to you. There is however a Gauntlet Plugin that provides a useful `TestController` class to assist with puppeteering and monitoring a game instance. It is well suited to smoke tests that require several steps to execute, but it is entirely optional.

Gauntlet Does

- Platform independent implementations of builds, devices, and tests as low-level units that provide the functional operations necessary to be configured and combined.
- Provide high-level classes that can launch complicated configurations (e.g. 5v5 clients and a server) for games.
- Provide utility functions for parsing log files, crashes, and accessing /saved data from devices.

Gauntlet Does Not

- Require specific Unreal-side code. The intention is that the execution within Unreal uses whatever makes the most sense. It could be something extensive such as the Unreal Automation Framework, or just some command-line parameters that the game interprets.
- Create builds. You need to provide Gauntlet with a network or locally cooked build.

Architecture

Gauntlet offers three tiers of functionality. Earlier tiers may be (and are!) used in isolation for things like scripting device interactions, non-Unreal unit testing, etc whereas the latter tiers abstract a great amount of complexity behind simple user-configurable options.

Level 1

- Interfaces (devices, builds, apps) that abstract the concept of devices, builds, installations, and processes.
  - ITargetDevice
  - IBuildSource
  - IBuild
  - IAppInstall
  - IAppInstance
- A framework that supports sequential, parallel, and dependent execution of user-created tests.

    - TestExecutor
    - ITestNode

Level 2

- Device and AppInstance implementations for all platforms Unreal supports (PC, Mac, PS4, Xbox One, Switch, iOS, Android).
    - TargetDeviceWindows
    - TargetDeviceMac
    - TargetDevicePS4
    - TargetDeviceXboxOne
    - TargetDeviceSwitch
    - TargetDeviceIOS
    - TargetDeviceAndroid
- Classes that support the configuration and launch of one or more "roles" that exist in an Unreal session (e.g. four clients, one server)
    - UnrealAppConfig
    - UnrealSessionRole
    - UnrealSession
    - UnrealSessionInstance

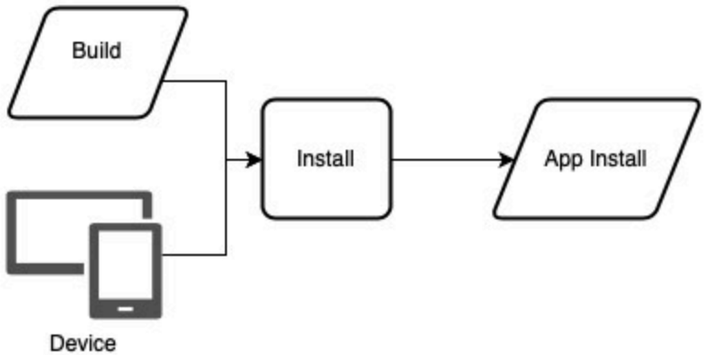
Level 3

- Classes that implement ITestNode and IBuildSource in the context of an Unreal Session
    - UnrealBuildSource
    - UnrealTestConfiguration

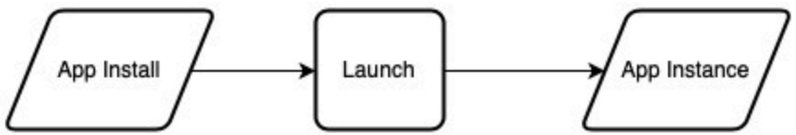
    - UnrealTestContext
    - UnrealTestNode
- RunUnreal - a class that inherits from UAT's BuildCommand and configures a number of parameters before constructing a list of tests and then executing them.

Gauntlet Core Interfaces

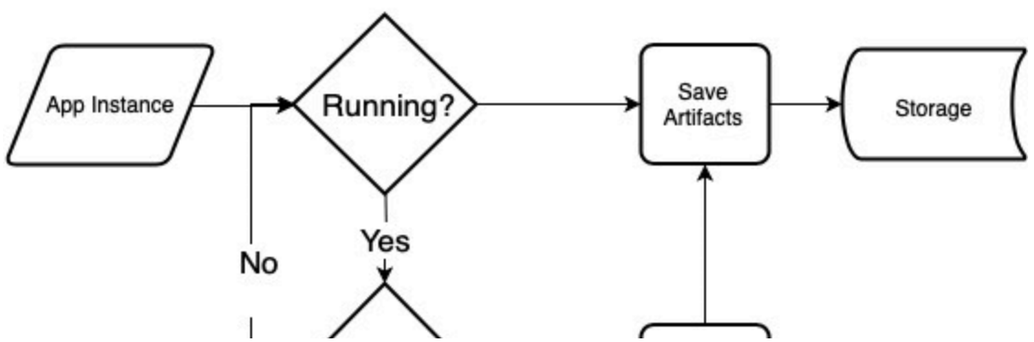
Installation

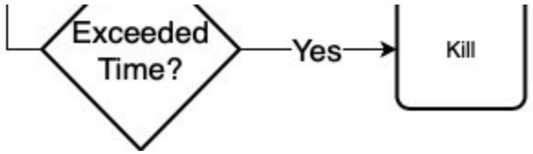


Launching



Monitoring





Core Classes

TestExecutor

The TestExecutor class handles creation, queuing, and monitoring of a set of tests. Multiple tests can be executed in parallel if the TestNode supports such a thing. An example of where care is needed is port usage for Unreal Servers. For the first test a default port is fine, but when tests are executing in parallel both the server and client must use unique ports.

ITestNode

ITestNode is an interface class that describes the API all test nodes must implement.

Unreal Test Node

Unreal Test node implements ITestNode, largely by configuring an UnrealSession object and calling its relevant functions.