Run Gauntlet Tests

Learn how to run Gauntlet tests.

RunUnreal Command: An Unreal Engine (UE) Tests Wrapper

Gauntlet has UE-specific commands, tests, and classes to handle Engine specifics.

The main Gauntlet command to trigger UE tests is `RunUnreal`. This command utilizes specific classes to handle UE-packaged games and other output.

Some related tests are already implemented to drive common test workflows, including the following:

- `UE.BootTest` and `UE.EditorBootTest` - Starts the project Client or Editor and then exits after initialization completes.
- `UE.EditorAutomation` and `UE.TargetAutomation` - Runs Engine Automation Test Framework on Editor and Client.

- `UE.Networking` - Runs an automated networking test if the target map is set up to trigger `NetTestGauntletClientController` or `NetTestGauntletServerController`.
- `UE.ErrorTest` - Runs automated tests on a target map if it is set up to trigger the `ErrorTest` Gauntlet Controller.
- `UE.PLMTest` - Runs Process Lifetime Management on the target platform.

UE.Automation Tests

The tests under `UE.Automation.cs` simplify how you run C++ and functional tests from a build system.

Gauntlet includes a test to run UE functional tests in the Editor and a packaged game (Client): `UE.EditorAutomation` and `UE.TargetAutomation`.

Editor Command Line

```
RunUAT.bat RunUnreal -test=UE.EditorAutomation -runtest=Mytest.one -project=<path to uproject> -build=editor
```

Copy full snippet

Client Command Line

```
RunUAT.bat RunUnreal -test=UE.TargetAutomation -runtest=Mytest.one -project=<path to uproject> -build=<path to packaged game>
```

Copy full snippet

Target Platform Command Line

```
RunUAT.bat RunUnreal -test=UE.TargetAutomation -runtest=Mytest.one -project=<path to uproject> -build=<path to packaged game> -platform=<pl
```

Copy full snippet

Gauntlet can only deploy console and mobile device builds if you implement the corresponding `ITargetDevice`, `IDeviceFactory`, `IDefaultDeviceSource`, `IAppInstall`, and `IAppInstance`

Resuming Tests on Failure

UE.Automation tests can resume testing after a critical failure, such as a crash in the middle of a run. This behavior is optional as it forces the test controller to save a JSON file before tests start.

To enable resume behavior, add the `-ResumeOnCriticalFailure` argument.

The resume happens up to three times before considering the build too unstable to continue.

Passing Parameters to Gauntlet from Command Line

You can pass custom arguments to Gauntlet from the command line. Gauntlet automatically passes all arguments to the test class, which you can access using specific properties.

You can use the following command line syntax:

```
-test="MyTest(foo,bar='some value')"
```

Copy full snippet

You can access the parameter from the test class using the following syntax:

```
bool MyBoolFromArgumentLine = Context.TestParams.ParseParam("foo");
string MyValueFromArgumentLine = Context.TestParams.ParseValue("bar", "DefaultValue");
```

Copy full snippet

Alternatively, you can use the UAT-executed C# code to parse the global command line arguments (regardless if any other code consumed it) with the following syntax:

```
string MyValueFromArgumentLine = Globals.Params.ParseValue("ArgumentLine", "DefaultValue");
```

Copy full snippet

Lyra Gauntlet Test

This section guides you through running an existing Gauntlet test that is available in the Lyra Sample Game.

Boot Test Node Code

```
using Gauntlet;

namespace LyraTest
```

```
{
/// <summary>
/// A Simple boot test
/// </summary>
public class BootTest : UnrealTestNode<UnrealTestConfiguration>
{
public BootTest(UnrealTestContext InContext)
: base(InContext)
{}

/// <summary>
/// Return a config for BootTest, just need one client and a suitable
/// timeout
/// The test expect the client to quit automatically
/// </summary>
/// <returns></returns>
public override UnrealTestConfiguration GetConfiguration()
{
UnrealTestConfiguration Config = base.GetConfiguration();

// Get a single client
UnrealTestRole ClientRole = Config.RequireRole(UnrealTargetRole.Client);
// Exit when a specific log message is triggered
ClientRole.CommandLineParams.Add("testexit", "GauntletHeartbeat: Idle");

Config.MaxDuration = 5 * 600; // 5 minutes timeout

return Config;
}
}
}
```

Copy full snippet

Run the Boot Test Example

1. Open a command prompt.
2. Change directory to `Engine/Build/BatchFiles` within your Unreal Engine root directory.
3. Enter the following command in the command prompt.

```
    RunUAT BuildCookRun -project=Samples/Games/Lyra/Lyra.uproject -platform=Win64 -configuration=Development -build -cook -pak -stage
```

Copy full snippet

4. After the process completes, enter the following command in the command prompt.

```
    RunUAT RunUnreal -project=Samples/Games/Lyra -platform=Win64 -configuration=Development -build=local -test=LyraTest.BootTest
```

Copy full snippet

Boot Test Flow Explanation

1. `BuildCookRun` generates a Win64 build of Lyra in `Samples/Games/Lyra/Saved/StagedBuilds`.
2. `RunUnreal` launches Gauntlet, which:
    a. Creates an instance of the `LyraTest.BootTest` node, which provides basic rules for running the test.
    b. Discovers local builds for the Lyra Project.

c. Validates that a Win64 Development build was available.
d. Launches Lyra.
e. Monitors the running process for any issues.
f. Detects that Lyra exits.
g. Checks for common issues such as crashes, asserts, and fatal errors.

3. Creates a summary report if the `LyraTest.BootTest` node verifies the test is still running without error.