Espresso

How to run your Espresso tests in Perfecto

On this page:

This section provides instructions on how to run Espresso tests with the Perfecto Gradle Plugin in Perfecto. It assumes that you are:

- Familiar with Espresso
- Have existing tests to work with
- Are a novice user of Perfecto

The Perfecto Gradle Plugin allows you to:

- Select a device or multiple devices from the Perfecto Lab to run Espresso Tests for applications.
- Install the application and test files onto the selected devices.
- Run the test methods on the devices.
- See the progress of the test set on the console.
- Access the Report Library to view the results of thee tests.

Setting up the Gradle plugin involves these tasks:

- Installing the Gradle plugin to prepare the **build.gradle** file
- Configuring parameters through a configuration file (recommended)
- Activating the plugin, understanding the output, and connecting to Smart Reporting execution reports

The following steps assume that the Espresso application and test files are available in the local disk storage.

Each Gradle task supports the following actions:

- Reading of Perfecto configuration parameters that select the devices to install and run the instrumentation tests
- Installing the application, testing `.apk` files on Perfecto Lab devices, and running the test methods
- Generating output to the Gradle console and the Perfecto single test report (STR) report

A sample project is available here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject

Prerequisites

Before you get started, make sure you have installed the following:

- Mac or Windows machine
- Gradle
- Java
- Android Studio with Android SDK

In addition, you need access to the Perfecto Gradle plugin. You can download it either automatically, after adding the required lines of code to the `build.gradle` file, as described in step 2 below, or, if your organization does not permit direct download, by pre-downloading it to a local libs folder (see also Install the Perfecto Gradle plugin manually).

Note: The Perfecto Gradle plugin is not backward compatible. Make sure you always use + in the classpath to get the latest version of the build, as follows:

```
com.perfectomobile.instrumentedtest.gradleplugin:plugin:+
```

1 | Get started

The starting point is a local Android Studio project without Perfecto configuration: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/tree/master/LocalEspresso. Android Studio projects are integrated with Gradle and include several **`build.gradle`** files – one at the project level and one for each application. In the local sample project:

- The project-level **`build.gradle`** file is located here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/blob/master/LocalEspresso/build.gradle

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        jcenter()
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.1'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
        google()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

- The app-level **`build.gradle`** file is located here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/blob/master/LocalEspresso/app/build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    buildToolsVersion '28.0.3'

    defaultConfig {
        applicationId "com.example.perfecto.tipcalculator"
        minSdkVersion 14
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner 'android.support.test.runner.AndroidJUnitRunner'
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.support:appcompat-v7:24.2.0'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    }
}
```

To get started:

1. Clone the project: https://github.com/PerfectoMobileSA/PerfectoEspressoProject
2. Open Android Studio.
3. Select the LocalEspresso workspace.
4. Right-click the project and select **Synchronize LocalEspresso**.
5. Fix any Gradle-related issues, such as creating a `local.properties` file under base project to set the `sdk.dir` and `ndk.dir`.

2 | Configure the project for Perfecto

In this step, we update both `build.gradle` files with the required Perfecto dependencies. We also create a JSON file that holds all Perfecto configurations, including security information, the Perfecto cloud name, Smart Reporting information, and test data.

The updated project is located here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/tree/master/PerfectoEspresso. The following procedure walks you through the configuration.

Expand a step to view its content.

A | Add the Perfecto plugin dependency to your build.gradle file

In this step, we work with build configuration scripts called **`build.gradle`**. Android Studio projects are integrated with Gradle and include several **`build.gradle`** files – one at the project level and one for each application. In our sample project:

- The project-level **`build.gradle`** file is located here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/blob/master/PerfectoEspresso/build.gradle

```
// Top-level build file where you can add configuration options common to all sub-
projects/modules.

buildscript {
    repositories {
        jcenter()
        google()
        maven {
            url "https://repo1.perfectomobile.com/public/repositories/maven"
       }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.1'
        classpath 'com.perfectomobile.instrumentedtest.gradleplugin:plugin:+'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
        google()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

- The app-level **`build.gradle`** file is located here: https://github.com/PerfectoMobileSA/PerfectoEspressoProject/blob/master/PerfectoEspresso/app/build.gradle

```
apply plugin: 'com.android.application'
apply plugin: 'com.perfectomobile.instrumentedtest.gradleplugin'

perfectoGradleSettings {
    configFileLocation "configFile.json"
}

android {
    compileSdkVersion 26
    buildToolsVersion '28.0.3'

    defaultConfig {
        applicationId "com.example.perfecto.tipcalculator"
        minSdkVersion 14
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner 'android.support.test.runner.AndroidJUnitRunner'
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.support:appcompat-v7:24.2.0'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:
2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    }
}
```

1. In Android Studio, open your project-level **`build.gradle`** file
2. Add the lines that define the location of the plugin library and the dependency on the plugin to the **`build.gradle`** file.

   Gradle will look to verify that the plugin is installed before performing the task.

   > This is the recommended configuration method. However, if there are problems using the automatic download method described here, you can download the Perfecto Gradle plugin manually.

   Do one of the following, depending on whether the plugin library is already downloaded or you want to locate and download the plugin library automatically:

   - To configure Gradle to automatically locate and download the plugin library, add the following lines to the **`build.gradle`** file:

```
buildscript {
    repositories {
        google()
        jcenter()
        maven {
            url "https://repo1.perfectomobile.com/public/repositories/maven/"
        }
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.1'
        classpath "com.perfectomobile.instrumentedtest.gradleplugin:plugin:+"
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

- If the plugin library is *already downloaded* to a folder (for example the **libs** sub-folder), add the following lines to the ***build.gradle*** file:

```
buildscript {
    repositories {
        google()
        jcenter()

        flatDir dirs: 'libs'
    }

    dependencies {
        classpath "com.perfectomobile.instrumentedtest.gradleplugin:plugin:+"
    }
}
```

3. Save the file.
4. Open the app-level **`build.gradle`** file and add the lines that defines the plugin task.

```
apply plugin: 'com.perfectomobile.instrumentedtest.gradleplugin'
```

5. Add the following line to load any configurations from the `configFile.json` configuration file.

```
perfectoGradleSettings {
    configFileLocation "configFile.json"
}
```

6. Save the file.

B | Create an Espresso configuration file

In this step, you create the JSON text file that contains all configuration settings. This is the recommended practice. Configurations include the URL of the Perfecto lab, the security token to use, which devices to select, which tags to use, and so on.

```
{
  "cloudURL": "<<cloud name>>",
  "securityToken": "<<SECURITY TOKEN>>",
  "devices": [
          {},
      {
                        "platformName": "Android",
                        "platformVersion": "^[678].*"
          }
  ],
  "jobName": "some_job",
  "jobNumber": 1,
  "branch": "some_branch",
  "projectName": "My_Espresso_project",
  "projectVersion": "v1.0",
  "tags": [
    "espresso", "plugin"
  ],
  "apkPath": "app/build/outputs/apk/debug/app-debug.apk",
  "testApkPath": "app/build/outputs/apk/androidTest/debug/app-debug-androidTest.apk",
  "installationDetails" : {"preCleanUp" : "true"},
  "postExecution" : {"uninstall" : "false" },
  "debug": false,
  "failBuildOnFailure": false,
  "takeScreenshotOnTestFailure": true,
  "shard": false,
  "testTimeout" : 60000
}
```

1. Create a configuration file (**`configFile.json`**) as a JSON text file in a known folder.
2. Add the connection parameters to the configuration file.

   ```
   "cloudURL": "<<cloud name>>",
   "securityToken": "<<SECURITY TOKEN>>",
   ```

   where:

   - `cloudURL` is the URL of the Perfecto Lab to connect to. For example *mobilecloud.perfectomobile.com*.

   - `securityToken` is the tester's personal security token for the Perfecto lab. See also Generate security tokens.
3. Add the device selection parameters to the configuration file, as shown in the following examples. For details on platform-specific parameters, see Android configuration parameters for the Gradle Plugin.

   > **NOTE:** If a device has not been explicitly selected (using the **deviceName** property) or when **numOfDevices** is used (without explicitly setting **platformVersion**), the system automatically selects an Android device in accordance with the minimum Android version as defined in your application.

   For example, add the following to select:

   - Any available Android device (based on the gradle task):

     ```
     "devices": [
         {}
       ],
     ```

- Specific devices, such as two devices where one is an Android device of version 6.*, 7.*, or 8.* (using a regular expression) and one is randomly selected by the Gradle plugin:

```
"devices": [
          {},
     {
                         "platformName": "Android",
                         "platformVersion": "^[678].*"
          }
 ],
```

- A number of random Android devices:

```
"numOfDevices": 20
```

4. Add the reporting parameter settings to add tags to the execution report. For more information on adding tags, see Tag -driven reports (RTDD workflow).

```
"jobName": "some_job",
"jobNumber": 1,
"branch": "some_branch",
"projectName": "My_Espresso_project",
"projectVersion": "v1.0",
"tags": [
  "espresso", "plugin"
```

where:

- **`jobName`** is the CI identification of the build, used for classification of the report in the CI Dashboard.
- **`jobNumber`** is the CI job number of the build.
- **`branch`** is the job branch as reported in the execution context.
- **`projectName`** is the name of the project, for classification.
- **`projectVersion`** is the version number assigned to the project for this build.
- **`tags`** is the set of tags to associate with the execution.

5. Add the application parameters to identify where the *application* files are located.

For our Android application, these are **`appPath`** and **`testAppPath`**:

```
"apkPath": "app/build/outputs/apk/debug/app-debug.apk",
"testApkPath": "app/build/outputs/apk/androidTest/debug/app-debug-androidTest.apk",
```

where:

- **`appPath`** refers to the path of the `.apk` file for the application.

- **`testApkPath`** is the path to the UI test application runner `.apk` file.

> **Note**: When testing an Android (*aar* or *jar*) library, set both the **apkPath** and **testApkPath** fields to the location specified for the ***assembleAndroidTest*** gradle task. By default, this location is: *<project folder>/app/build/outputs /apk/androidTest/debug/app-debug-androidTest.apk.* You can supply additional parameters if you need to limit the test application to only execute specific methods or classes

6. Save the configuration file.

3 | Run the plugin and view the report

This step walks you through running the Perfecto Gradle plugin and viewing the test report in Perfecto.

Expand a step to view its content.

A | Execute the plugin

1. Open a command-line (terminal) window in the folder where you want to execute the plugin.
2. Execute the plugin using the following command:

   Here we also supply the full path to the configuration file created in step 3. You can add other configuration parameters (except for device selection) to the command as needed.

For iOS apps
`gradle perfecto-android-inst -PconfigFileLocation=configFile.json`

   This will:

   - Select the devices as specified in the Device selection parameters of the configuration file (or a random device if no specification is provided).
   - Install the application and test *.apk* files onto the device.
   - Run the test methods (based on the configuration parameters).
   - Send output to the console window.
   - Generate an execution report that can be viewed in the Test analysis with Smart Reporting interface.

   The command-line parameters can set any of the configuration parameters, except for device selection parameters.

   For more information, see Android configuration parameters for the Gradle Plugin.

If you run the Gradle plugin over a proxy connection, you need to supply the proxy information in form of the following Java parameters when activating the plugin:

- **http.proxyHost** - IP address of the proxy
- **http.proxyPort** - IP port used for connection
- **http.proxyUser** - username for connection to proxy server
- **http.proxyPassword** - password for connection to proxy server.

For proxies supporting SSL encryption, use the following Java Proxy parameters:

- **https.proxyHost** - IP address of the proxy
- **https.proxyPort** - IP port used for connection
- **https.proxyUser** - username for connection to proxy server
- **https.proxyPassword** - password for connection to proxy server.

For example:

```
gradle perfecto-android-inst -PconfigFileLocation=configFile.json -Dhttp.
proxyHost=10.0.0.100 -Dhttp.proxyPort=8800 -Dhttp.proxyUser=someUserName -
Dhttp.proxyPassword=somePassword
```

During the execution, the plugin reports on the progress of the execution and the completion of each test method to the command-line window.

```
Task :app:perfecto-android-inst
Parsing configuration file: configFile.json
Parsed configuration file configFile.json successfully

Starting Execution
Your management id is: ****-***-********-******
Uploading files
Files uploaded
Acquiring device based on device management info {"platformName":"Android"}
Acquiring device based on device management info {"platformName":"Android","
platformVersion":"^[678].*"}
Device: <<DEVICE ID>> Samsung Galaxy S10: Operating on device {"deviceId":"
<<DEVICE ID>>","osVersion":"9","screenResolution":"1440*3040","location”:”
******","network”:”******","manufacturer":"Samsung","model":"Galaxy S10","
distributor":"Generic","selectionCriteria":{"platformName":"Android"},"os":"
Android"} according to device details {"platformName":"Android","index":1}
Device: <<DEVICE ID>> Samsung Galaxy S10: Uploading APK to device server
Device: <<DEVICE ID>> Samsung Galaxy S10: Completed uploading APK to device server
Device: <<DEVICE ID>> Samsung Galaxy S10: Starting video recording
Device: <<DEVICE ID>> Samsung Galaxy S10: Enabling instrumented test mode
Device: <<DEVICE ID>> Samsung Galaxy S10: Trying to uninstall package com.example.
perfecto.tipcalculator from device
Device: <<DEVICE ID>> Samsung Galaxy S10: Uninstalled package com.example.
perfecto.tipcalculator successfully
Device: <<DEVICE ID>> Samsung Galaxy S10: Trying to uninstall package com.example.
perfecto.tipcalculator.test from device
Device: <<DEVICE ID>> Samsung Galaxy S10: Uninstalled package com.example.
perfecto.tipcalculator.test successfully
Device: <<DEVICE ID>> Samsung Galaxy S10: Installed APK successfully
Device: <<DEVICE ID>> Samsung Galaxy S10: Installed test APK successfully
Device: <<DEVICE ID>> Samsung Galaxy S10: Starting execution of package com.
example.perfecto.tipcalculator.test with test runner android.support.test.runner.
AndroidJUnitRunner
Device: <<DEVICE ID>> Samsung Galaxy S10: TestResult{className: com.example.
perfecto.tipcalculator.SimpleTest, methodName: simpleTest, status: PASS}
Device: <<DEVICE ID>> Samsung Galaxy S10: TestResult{className: com.example.
perfecto.tipcalculator.SimpleTest, methodName: simpleTestt, status: PASS}
Device: <<DEVICE ID>> Samsung Galaxy S10: Disabling instrumented test mode
Device: <<DEVICE ID>> Samsung Galaxy S10: Ending test execution
Device: <<DEVICE ID>> Samsung Galaxy S10: Stopping video recording
Device: <<DEVICE ID>> Samsung Galaxy S10: Closing device
```

At the end of the execution, the command-line window displays a *high-level summary report* of the completion status for each device. The report includes:

- Resolution of configuration settings: The configuration file used and the list of the configuration settings for the test run
- Progress notifications as the test is configured, installed, and executed, including notifications of the start and completion of:
    - Allocating the device
    - Resigning the application
    - Installing the application and UI runner files
    - Executing the test
    - Connecting the video recording
- Completion status for each test method on each device

At the end of the summary report, the plugin provides the URL of the single test report (STR) for the execution run.

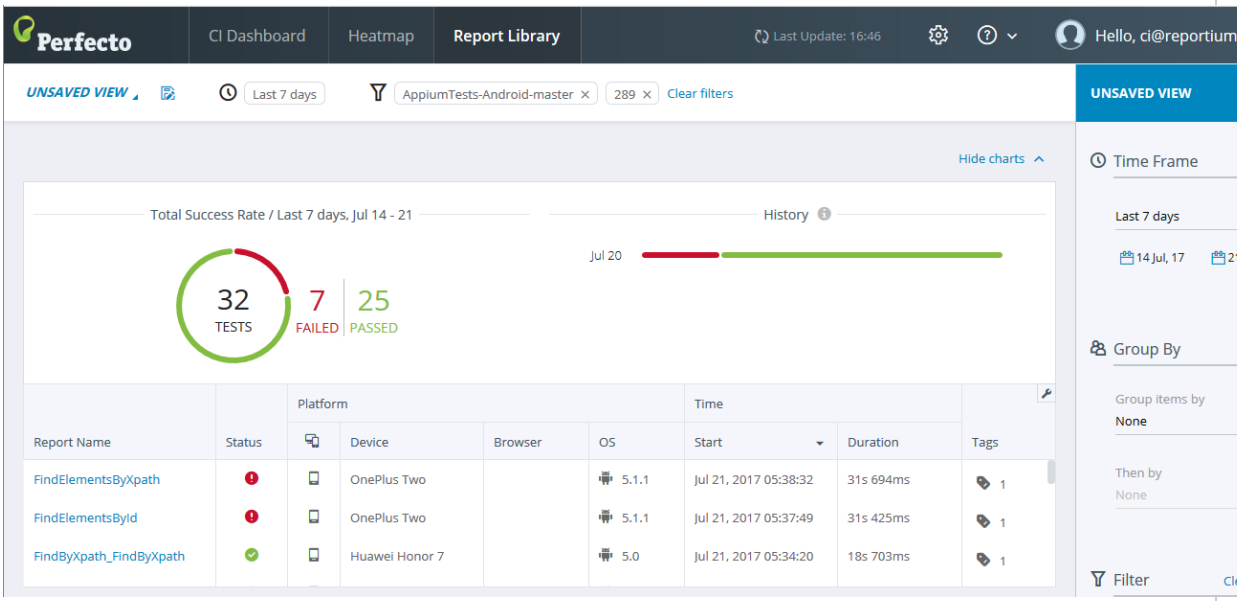
```
--------------------------------------------------------------------------------

TOTAL SUMMARY
FAILED TO RUN ON 0 devices
RAN SUCCESSFULLY ON 1 device
--------------------------------------------------------------------------------

DEVICE 1
Status: OK
Device:
<<DEVICE ID>> Samsung Galaxy S10

Results:
2 FAILED
2 PASSED
--------------------------------------------------------------------------------

View the detailed report at:
https://demo.reporting.perfectomobile.com/reporting/library?tags[0]
=*************************

Finished flow execution
```

B | Access the execution report

To access the execution report, copy the report URL from the summary report on your console and open the URL in your browser. For example:

```
View the detailed report at: https://demo.reporting.perfectomobile.com/library?
startExecutionTime[1]=lastMonth&tags[0]=5af27a82-54cc-405a-8c6e-fa46fcae874b
Finished flow execution
```

Here, you can drill down to see the screenshots of the different test method executions and what went wrong, if anything. For more information on Smart Reporting, see Test analysis with Smart Reporting. For details on the STR, see Single test report (STR).



Samples of plugin use

The Perfecto GitHub repository https://github.com/PerfectoCode/Espresso includes different samples that demonstrate how you can use the Perfecto Gradle plugin for the following different modes/configurations:

- defaultAndroidProjectSample: An Android project with the plugin Json config file located in the default place
- localJarSample: An Android project configured to run with the Perfecto plugin jar file locally
- pluginConfigurationSample: An Android project with the cloudURL/security token configured in the `build.gradle` file
- remoteRunSample: A sample showing how to run the plugin without the source code, only with .apk files
- configFileSamples: Different examples of configuring the plugin using a `.json` file. Except for devices, all parameters can be overriden by the command line.

Demo video

The following video demonstrates these steps.