GENYMOTION

Gradle Plugin Guide

Version 1.4

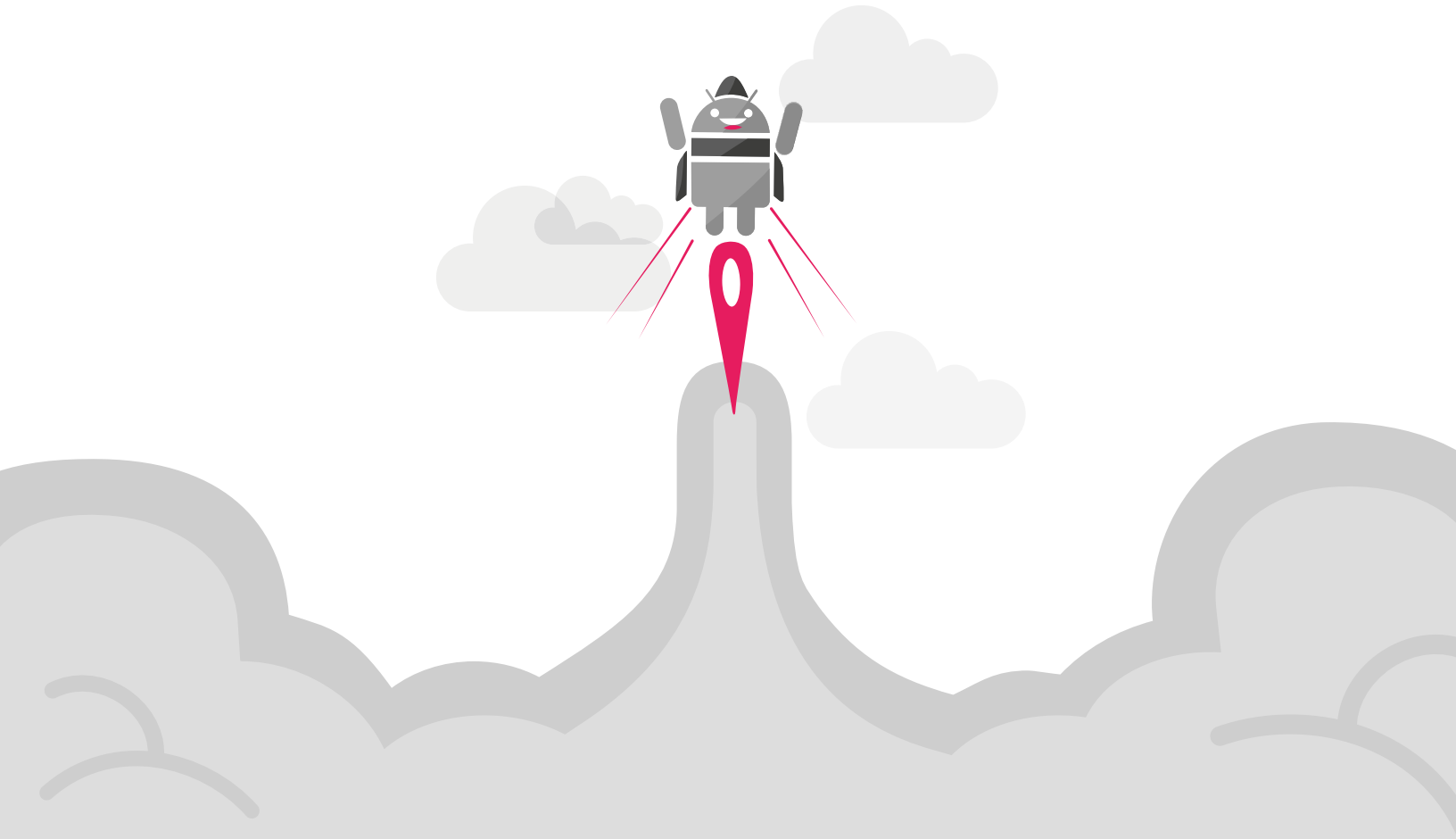

Table of contents

Overview

This feature is only available with Indie and Business licenses.

A valid Genymotion account is required.

The Gradle plugin allows you to combine the use of the Gradle build system with Genymotion. This way, you can declare the virtual devices you want to start before running your tests.

It can be easily used with Android Studio (or the Android Gradle plugin) but can also be used without.

By default, if an Android plugin is detected, all defined virtual devices will be up and ready before the `connectedAndroidTest` task (or its derivatives) is launched. If no Android plugin is detected, you must configure the behavior.

connectedAndroidTest is the Gradle task defined by the Android Gradle plugin to run instrumentation tests on devices.

This guide explains how to include, use and configure the plugin by presenting several examples.

In this guide, the following instructional icons are used:

Notes, tips or additional information.

Situations that could cause performance issues or data losses.

Including the plugin

This chapter details how to reference and apply the plugin to your development environment.

As the Gradle plugin is based on gmtool command line, it must be installed on the same host.

Referencing the plugin

To use the Gradle plugin, you first need to reference its repository in the `buildscript` block of the `build.gradle` file. The file should look like:

```
buildscript {
   repositories {
      jcenter()
   }
   dependencies {
      classpath 'com.genymotion:plugin:1.4'
   }
}
```

To reference the plugin repository:

1. Add the `jcenter` repository (where the plugin is hosted).
2. Declare the repository as a build dependency.

'1.4' corresponds to the latest version of the plugin. You can get the list of all releases on the Bintray page.

Applying the plugin

To apply the plugin, insert the following line to the `build.gradle` file.

```
apply plugin: "genymotion"
```

Using the plugin

To access all features of the Gradle plugin, you must declare the virtual devices you want to create and start prior to running tests, as shown below:

```
genymotion {
  devices {
    nexus5 {
      template "Google Nexus 5 - 4.4.4 - API 19 - 1080x1920"
    }
  }
}
```

To declare a virtual device:

1. Open a `genymotion.devices` block in your `build.gradle` file.
2. Add a virtual device name block.
   By adding this block to your `build.gradle` file, you ask to create and then start a Nexus 5 prior to running the tests.
3. Define the virtual device configuration. Here the virtual device is defined by a template ("Google Nexus 5 - 4.4.4 - API 19 - 1080x1920").

The virtual device will be started and accessible via adb. The Android Gradle plugin will then run all the instrumentation tests on it.

By default, all virtual devices are created prior to the tests and then deleted on the fly.

Device properties

This section lists all parameters that can be defined for the `genymotion.devices` block:

- `template "Google Nexus 5 - 4.4.4 - API 19 - 1080x1920"`
  creates a virtual device from the specified template.
- `start true|false`
  starts or does not start the virtual device.
- `pushBefore <SOURCE_PATH>:[DEST_PATH], <SOURCE_PATH>:[DEST_PATH]`
  `default DEST_PATH: /sdcard/Downloads`
  pushes a file before running tests.
- `pullBefore <SOURCE_PATH>:<DEST_PATH>, <SOURCE_PATH>:<DEST_PATH>`
  pulls a file before running tests.

- `pushAfter <SOURCE_PATH>:[DEST_PATH], <SOURCE_PATH>:[DEST_PATH] default DEST_PATH: /sdcard/Downloads`
  pushes a file after running tests.
- `pullAfter <SOURCE_PATH>:<DEST_PATH>, <SOURCE_PATH>:<DEST_PATH>`
  pulls a file after running tests.
- `install "ok.apk", "sign.apk”`
  installs an APK file before running tests.
- `flash "path.zip", "other.zip"`
  flashes a device before running tests.
- `productFlavors "flavor1", "flavor2"`
  sets flavors for a virtual device. If defined, the virtual device will be started only for tests concerning the flavors it owns.
- `logcat "OUTPUT_PATH"`
  dumps the virtual device logcat after tests.
- `clearLogAfterBoot true|false`
  clears the logcat after the virtual device boot.
- `deleteWhenFinish true|false`
  deletes or keeps the virtual device when finished.
- `stopWhenFinish true|false`
  stops or keeps the virtual device running when finished.
- `density "SCREEN_DENSITY"`
  changes the screen density (ldpi, mdpi, tvdpi, hdpi, xhdpi, xxhdpi, xxxhdpi).
- `width SCREEN_WIDTH`
  sets the screen width of the virtual device.
- `height SCREEN_HEIGHT`
  sets the screen height of the virtual device.
- `virtualKeyboard true|false`
  activates or deactivates the virtual keyboard in the virtual device.
- `navbarVisible true|false`
  displays or hides the Android navigation bar in the virtual device.
- `nbCpu NUMBER_OF_CPUs`
  sets the number of processors used by the virtual device.
- `ram RAM_IN_MB`
  sets the memory space allocated to the virtual device in MB.
- `networkMode "MODE", "BRIDGED_INTERFACE"`
  MODE can have either "bridge" or "nat" value. In "bridge" mode, BRIDGED_INTERFACE must be provided, e.g., networkMode "bridge", "en0"

Example 1

```
genymotion {
  devices {
    "Nexus 7" {
      template "Google Nexus 7 - 4.2.2 - API 17 - 800x1280"
      density "ldpi"
      deleteWhenFinish false
    }
  }
}
```

In this case:

1. A virtual device named “Nexus 7” is created.
2. The following properties have been set:
   - The template used is "Google Nexus 7 - 4.2.2 - API 17 - 800x1280" (`template`).
   - Screen density is set to "ldpi" (`density`).
   - The virtual device will not be deleted when tests will be finished (`deleteWhenFinish`). In this case, the virtual device will be stopped but it will remain in the Genymotion virtual device list. When the test will be run again, the virtual device will be started without being created again.
     Other properties that can be set are detailed in section *Device properties*.

To add special characters to the device name, write it between quotation marks.

Example 2

```
genymotion {
  devices {
    "Nexus 10" {
      template "Google Nexus 10 - 4.3 - API 18 - 2560x1600"
      flash "PATH_TO_ZIP.zip"
      install "PATH_TO_APK.apk"
      pushBefore "PATH_TO_FILE.txt"
      pullAfter "/system/build.prop":"/temp/tests/"
      logcat "/temp/nexus10.logcat"
    }
  }
}
```

In this case:

- A virtual device named “Nexus 10” is created.
- The virtual device is created from the template "Google Nexus 10 - 4.3 - API 18 - 2560x1600" (`template`).
- A zip file is flashed onto the device (`flash`).
  It can be used if you want to modify the `/system` partition of the machine.
- An APK file is installed (`install`).
  This can be useful if the application to test requires the presence of another app to interact with.
- A file is pushed onto the device before running the tests (`pushBefore`).
  You can also push a file after the tests with `pushAfter`.
- The `build.prop` file is copied from the virtual device to the `/temp/tests/` folder into the host (`pullAfter`).
- The virtual device logcat is dumped into a specific file (`logcat`).

Using the plugin with cloud devices

You can also use Gradle plugin to control cloud devices. To do so, you must declare the devices similarly to local devices, but using the cloudDevices block:

```
genymotion {
  cloudDevices{
    nexus5 {
      template "Google Nexus 5 - 4.4.4 - API 19 - 1080x1920"
    }
  }
}
```

The virtual device will be started and accessible via adb. The Android Gradle plugin will then run all the instrumentation tests on it.

Device properties

This section lists all parameters that can be defined for the `genymotion.devices` block:

- `template "Google Nexus 5 - 4.4.4 - API 19 - 1080x1920"`
  creates a virtual device from the specified template.
- `pushBefore <SOURCE_PATH>:[DEST_PATH], <SOURCE_PATH>:[DEST_PATH]`
  `default DEST_PATH: /sdcard/Downloads`
  pushes a file before running tests.
- `pullBefore <SOURCE_PATH>:<DEST_PATH>, <SOURCE_PATH>:<DEST_PATH>`
  pulls a file before running tests.
- `pushAfter <SOURCE_PATH>:[DEST_PATH], <SOURCE_PATH>:[DEST_PATH]`
  `default DEST_PATH: /sdcard/Downloads`
  pushes a file after running tests.
- `pullAfter <SOURCE_PATH>:<DEST_PATH>, <SOURCE_PATH>:<DEST_PATH>`
  pulls a file after running tests.
- `install "ok.apk", "sign.apk”`
  installs an APK file before running tests.
- `flash "path.zip", "other.zip"`
  flashes a device before running tests.
- `productFlavors "flavor1", "flavor2"`
  sets flavors for a virtual device. If defined, the virtual device will be started only for tests concerning the flavors it owns.

- `logcat "OUTPUT_PATH"`
  dumps the virtual device logcat after tests.
- `clearLogAfterBoot true|false`
  clears the logcat after the virtual device boot.

Configuring the plugin

To configure the Gradle plugin, you must declare the configuration properties, as shown below:

```
genymotion {
  config {
    genymotionPath = "/home/user/app/genymotion/"
    taskLaunch = "myCustomTask"
  }
}
```

The `taskLaunch` property is optional. By default, if an Android plugin is detected, all defined virtual devices will be up and ready before the Android instrumentation tasks (connectedAndroidTest, etc.). If no Android plugin is detected, you must configure the behavior and therefore, define the `genymotion.config.taskLaunch` property.

To configure the plugin:

1. Open a `genymotion.config` block in your `build.gradle` file.
2. If you did not install Genymotion on the default path on Windows and Mac or if the Genymotion folder is not on your PATH environment variable, set the Genymotion installer path (`genymotionPath`) to let the plugin run the GMTool binary.
3. Define the Gradle task where Genymotion tasks will be injected (`taskLaunch`).
   In the example above, virtual devices will be ready before the `runTest` task is launched and will be closed and removed when finished.

Other properties that can be set are detailed in section *Configuration properties*.

By default, the plugin uses the current Genymotion configuration (for username, password, license, proxy, etc.) but you can set it using the Gradle plugin:

```
genymotion {
  config {
    username = "myuser"
    password = "password"
    license = "mylicense"
  }
}
```

However, good practice is not to set these data inside the `build.gradle` file. Usernames and passwords must not be published on your versioning repository. Instead, you must configure it from your `local.properties` file or create it if it does not exist yet.

This file is located in the project root folder.

```
genymotion.username=myuser
genymotion.password=password
genymotion.license=mylicense
```

You can set all configuration parameters from the `local.properties` file by prefixing the config key with `genymotion`.

When committing your files, make sure you ignore the `local.properties` file.

Configuration properties

This section lists all parameters that can be defined for the `genymotion.config` block:

- `genymotionPath = GENYMOTION_INSTALLER_PATH`
  sets the Genymotion installer path.
- `fromFile = CONFIG.properties`
  retrieves a whole configuration from a file. The file content gets the priority over the `genymotion.config` block.
- `username = USERNAME`
  sets the user to authenticate.
- `password = PASSWORD`
  sets the password to associate with the username.
- `licenseServer = true|false`
  activates or deactivates the use of a license server.
- `licenseServerAddress = SERVER_ADDRESS`
  sets the license server address.
- `license = LICENSE_KEY`
  registers a license key or renews Genymotion activation.
- `statistics = true|false`
  allows or prevents Genymotion to collect usage statistics.
- `proxy = true|false`
  activates or deactivates the use of a proxy.
- `proxyAddress = PROXY_ADDRESS`
  sets the proxy address.
- `proxyPort = PROXY_PORT`
  sets the proxy port.

- `proxyAuth = true|false`
  activates or deactivates proxy authentication.
- `proxyUsername = PROXY_USERNAME`
  sets the username to be used for proxy authentication.
- `proxyPassword = PROXY_PASSWORD`
  sets the password to be used for proxy authentication.
- `virtualDevicePath = VIRTUAL_DEVICE_PATH`
  sets the path to the virtual device directory, where configurations are stored.
- `androidSdkPath = SDK_PATH`
  sets the path to the custom Android SDK directory.
  This is where a specific ADB tool is retrieved. If not specified, Genymotion ADB tool is used.
- `useCustomSdk = true|false`
  activates or deactivates the use of the custom Android SDK.
- `screenCapturePath = SCREEN_CAPTURE_PATH`
  sets the path to the screen capture directory.
- `taskLaunch = "TASK"`
  defines the task that depends on the Genymotion launch task.
- `automaticLaunch = true|false`
  enables or disables injection of Genymotion tasks.
- `processTimeout = TIMEOUT_IN_MILLISECONDS`
  sets the timeout in seconds for all processes launched in command line.
- `verbose = true|false`
  gives a complete description of the command to help diagnose errors.
- `abortOnError = true|false`
  aborts or continues the task execution if a GMTool error occurs.

Handling flavors

The Android Gradle plugin brings the product flavors in Android. The Gradle plugin also handles this flavor system.

If the Android Gradle plugin is added and flavors are declared, you can set specific virtual devices to be started during tests related to flavors.

In the following example, two flavors are defined in the `build.gradle` file:

```
android {
...
  productFlavors {
    flavor1 {
      applicationId "com.genymotion.binocle.flavor1"
      versionCode 20
    }
    flavor2 {
      applicationId "com.genymotion.binocle.flavor2"
      minSdkVersion 14
    }
  }
}
```

In the following example, three virtual devices are defined. device1 is declared without specifying a flavor. It is considered as common to all flavors and will be started for each of them.

Then, device2 will be launched for flavor1 and device3 will be launched for both flavors.

```
genymotion {
  devices {
    device1 {
      template "Google Nexus 5 - 4.4.4 - API 19 - 1080x1920"
    }
    device2 {
      template "Google Nexus 7 - 4.2.2 - API 17 - 800x1280"
      productFlavors "flavor1"
    }
    device3 {
      template "Google Nexus 10 - 4.3 - API 18 - 2560x1600"
      productFlavors "flavor1", "flavor2"
    }
  }
}
```

Feedback and contributions

The Gradle plugin is open source, so feel free to have a look at the following link to explore it and contribute: *https://github.com/Genymobile/genymotion-gradle-plugin*