Buildozer Documentation

Release 0.11

Kivy's Developers

Sep 16, 2023

Contents

Buildozer is a tool that aim to package mobiles application easily. It automates the entire build process, download the prerequisites like python-for-android, Android SDK, NDK, etc.

Buildozer manages a file named *buildozer.spec* in your application directory, describing your application requirements and settings such as title, icon, included modules etc. It will use the specification file to create a package for Android, iOS, and more.

Currently, Buildozer supports packaging for:

- Android: via Python for Android. You must have a Linux or OSX computer to be able to compile for Android.
- iOS: via Kivy iOS. You must have an OSX computer to be able to compile for iOS.
- Supporting others platform is in the roadmap (such as .exe for Windows, .dmg for OSX, etc.)

If you have any questions about Buildozer, please refer to the Kivy's user mailing list.

CHAPTER 1

Installation

Buildozer is tested on Python 3.8 and above but may work on earlier versions, back to Python 3.3. Depending the platform you want to target, you might need more tools installed. Buildozer tries to give you hints or tries to install few things for you, but it doesn't cover every situation.

First, install the buildozer project with:

```
pip3 install --user --upgrade buildozer
```

1.1 Targeting Android

1.1.1 Android on Ubuntu 20.04 and 22.04 (64bit)

(expected to work as well in later version, but only regularly tested in the latest LTS)

```
sudo apt update
sudo apt install -y git zip unzip openjdk-17-jdk python3-pip autoconf libtool pkg-
↪config zlib1g-dev libncurses5-dev libncursesw5-dev libtinfo5 cmake libffi-dev␣
↪libssl-dev
pip3 install --user --upgrade Cython==0.29.33 virtualenv  # the --user should be␣
↪removed if you do this in a venv

# add the following line at the end of your ~/.bashrc file
export PATH=$PATH:~/.local/bin/
```

If openjdk-17 is not compatible with other installed programs, for Buildozer the minimum compatible openjdk version is 11.

1.1.2 Android on Windows 10 or 11

To use buildozer in Windows you need first to enable Windows Subsystem for Linux (WSL) and install a Linux distribution: https://docs.microsoft.com/en-us/windows/wsl/install.

These instructions were tested with WSL 1 and Ubuntu 18.04 LTS, and WSL2 with Ubuntu 20.04 and 22.04.

After installing WSL and Ubuntu on your Windows machine, open Ubuntu, run the commands listed in the previous section, and restart your WSL terminal to enable the path change.

Copy your Kivy project directory from the Windows partition to the WSL partition, and follow the Quickstart Instructions. **Do not** change to the project directory on the Windows partition and build there, this may give unexpected and obscure fails.

For debugging, WSL does not have direct access to USB. Copy the .apk file to the Windows partition and run ADB (Android Debug Bridge) from a Windows prompt. ADB is part of Android Studio, if you do not have this installed you can install just the platform tools which also contain ADB.

- Go to https://developer.android.com/studio/releases/platform-tools and click on "Download SDK Platform-Tools for Windows".
- Unzip the downloaded file to a new folder. For example, "C:\platform-tools".

1.1.3 Before Using Buildozer

If you wish, clone your code to a new folder, where the build process will run.

You don't need to create a virtualenv for your code requirements. But just add these requirements to a configuration file called buildozer.spec as you will see in the following sections.

Before running buildozer in your code folder, remember to go into the buildozer folder and activate the buildozer virtualenv.

1.1.4 Android on macOS

```
python3 -m pip install --user --upgrade Cython==0.29.33 virtualenv  # the --user␣
↪should be removed if you do this in a venv
```

1.1.5 TroubleShooting

Buildozer stuck on "Installing/updating SDK platform tools if necessary"

Press "y" then enter to continue, the license acceptance system is silently waiting for your input

Aidl not found, please install it.

Buildozer didn't install a necessary package

```
~/.buildozer/android/platform/android-sdk/tools/bin/sdkmanager "build-tools;29.0.0"
```

Then press "y" then enter to accept the license.

python-for-android related errors

See the dedicated p4a troubleshooting documentation.

1.2 Targeting IOS

Install XCode and command line tools (through the AppStore)

Install homebrew (https://brew.sh)

```
brew install pkg-config sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer autoconf␣
↪automake
```

Install pip and virtualenv

```
python3 -m pip install --user --upgrade pip virtualenv kivy-ios
```

CHAPTER 2

Quickstart

Let's get started with Buildozer!

2.1 Init and build for Android

1. Buildozer will try to guess the version of your application, by searching a line like *__version__ = "1.0.3"* in your *main.py*. Ensure you have one at the start of your application. It is not mandatory but heavily advised.
2. Create a *buildozer.spec* file, with:

```
buildozer init
```

3. Edit the *buildozer.spec* according to the *Specifications*. You should at least change the *title*, *package.name* and *package.domain* in the *[app]* section.
4. Start a Android/debug build with:

```
buildozer -v android debug
```

5. Now it's time for a coffee / tea, or a dinner if you have a slow computer. The first build will be slow, as it will download the Android SDK, NDK, and others tools needed for the compilation. Don't worry, thoses files will be saved in a global directory and will be shared across the different project you'll manage with Buildozer.
6. At the end, you should have an APK or AAB file in the *bin/* directory.

2.2 Run my application

Buildozer is able to deploy the application on your mobile, run it, and even get back the log into the console. It will work only if you already compiled your application at least once:

```
buildozer android deploy run logcat
```

For iOS, it would look the same:

```
buildozer ios deploy run
```

You can combine the compilation with the deployment:

```
buildozer -v android debug deploy run logcat
```

You can also set this line at the default command to do if Buildozer is started without any arguments:

```
buildozer setdefault android debug deploy run logcat

# now just type buildozer, and it will do the default command
buildozer
```

To save the logcat output into a file named *my_log.txt* (the file will appear in your current directory):

```
buildozer -v android debug deploy run logcat > my_log.txt
```

To see your running application's print() messages and python's error messages, use:

```
buildozer -v android deploy run logcat | grep python
```

2.2.1 Run my application from Windows 10

- Plug your Android device on a USB port.
- Open Windows PowerShell, go into the folder where you installed the Windows version of ADB, and activate the ADB daemon. When the daemon is started you must see a number besides the word "device" meaning your device was correctly detected. In case of trouble, try another USB port or USB cable.

```
cd C:\platform-tools\
.\adb.exe devices
```

- Open the Linux distribution you installed on Windows Subsystem for Linux (WSL) and proceed with the deploy commands:

```
buildozer -v android deploy run
```

It is important to notice that Windows ADB and Buildozer installed ADB must be the same version. To check the versions, open PowerShell and type:

```
cd C:\platform-tools\
.\adb.exe version
wsl
cd ~/.buildozer/android/platform/android-sdk/platform-tools/
./adb version
```

2.3 Install on non-connected devices

If you have compiled a package, and want to share it easily with others devices, you might be interested with the *serve* command. It will serve the *bin/* directory over HTTP. Then you just have to access to the URL showed in the console from your mobile:

```
buildozer serve
```

CHAPTER 3

Specifications

This document explains in detail all the configuration tokens you can use in *buildozer.spec*.

3.1 Section [app]

- *title*: String, title of your application.

  It might be possible that some characters are not working depending on the targeted platform. It's best to try and see if everything works as expected. Try to avoid too long titles, as they will also not fit in the title displayed under the icon.

- *package.name*: String, package name.

  The Package name is one word with only ASCII characters and/or numbers. It should not contain any special characters. For example, if your application is named *Flat Jewels*, the package name can be *flatjewels*.

- *package.domain*: String, package domain.

  Package domain is a string that references the company or individual that did the app. Both domain+name will become your application identifier for Android and iOS, choose it carefully. As an example, when the Kivy's team is publishing an application, the domain starts with *org.kivy*.

- *source.dir*: String, location of your application sources.

  The location must be a directory that contains a *main.py* file. It defaults to the directory where *buildozer.spec* is.

- Source Inclusion/Exclusion options.
  - *source.include_exts*: List, file extensions to include.
  - *source.exclude_exts*: List, file extensions to exclude, even if included by *source.include_exts*
  - *source.exclude_dirs*: List, directories to exclude.
  - *source.exclude_patterns*: List, files to exclude if they match a pattern.
  - *source.include_patterns*: List, files to include if they match a pattern, even if excluded by *source.exclude_dirs* or *source.exclude_patterns*

By default, not all files are in your *source.dir* are included. You can use these options to alter which files are included in your app and which are excluded.

Directories and files starting with a "." are always excluded; this cannot be overridden.

Files that have an extension that is not in *source.include_exts* are excluded. (The default suggestion is *py,png,jpg,kv,atlas*. You may want to include other file extensions such as resource files: gif, xml, mp3, etc.) File names that have no extension (i.e contain no ".") are not excluded here. *source.exclude_exts* takes priority over *source.include_exts* - it excludes any listed extensions that were previously included.

Files and directories in directories listed in *source.exclude_dirs* are excluded. For example, you can exclude your *tests* and *bin* directory with:

```
source.exclude_dirs = tests, bin
```

source.exclude_patterns are also excluded. This is useful for excluding individual files. For example:

```
source.exclude_patterns = license
```

These dir and pattern exclusions may be overridden with *source.include_patterns* - files and directories that match will once again be included.

However, *source.include_patterns* does not override the *source.include_exts* nor *source.exclude_exts*. *source.include_patterns* also cannot be used to include files or directories that start with ".")

- *version.regex*: Regex, Regular expression to capture the version in *version.filename*.

  The default capture method of your application version is by grepping a line like this:

  ```
  __version__ = "1.0"
  ```

  The *1.0* will be used as a version.

- *version.filename*: String, defaults to the main.py.

  File to use for capturing the version with *version.regex*.

- *version*: String, manual application version.

  If you don't want to capture the version, comment out both *version.regex* and *version.filename*, then put the version you want directly in the *version* token:

  ```
  # version.regex =
  # version.filename =
  version = 1.0
  ```

- *requirements*: List, Python modules or extensions that your application requires.

  The requirements can be either a name of a recipe in the Python-for-android project, or a pure-Python package. For example, if your application requires Kivy and requests, you need to write:

  ```
  requirements = kivy,requests
  ```

  If your application tries to install a Python extension (ie, a Python package that requires compilation), and the extension doesn't have a recipe associated to Python-for-android, it will not work. We explicitly disable the compilation here. If you want to make it work, contribute to the Python-for-android project by creating a recipe. See *Contribute*.

- *presplash.filename*: String, loading screen of your application.

Presplash is the image shown on the device during application loading. It is called presplash on Android, and Loading image on iOS. The image might have different requirements depending the platform. Currently, Buildozer works well only with Android, iOS support is not great on this.

The image must be a JPG or PNG, preferable with Power-of-two size, e.g., a 512x512 image is perfect to target all the devices. The image is not fitted, scaled, or anything on the device. If you provide a too-large image, it might not fit on small screens.

- *icon.filename*: String, icon of your application.

  The icon of your application. It must be a PNG of 512x512 size to be able to cover all the various platform requirements.

- *orientation*: List, supported orientations of the application.

  Indicate the orientations that your application supports. Valid values are: *portrait*, *landscape*, *portrait-reverse*, *landscape-reverse*. Defaults to *[landscape]*.

- *fullscreen*: Boolean, fullscreen mode.

  Defaults to true, your application will run in fullscreen. Means the status bar will be hidden. If you want to let the user access the status bar, hour, notifications, use 0 as a value.

- *home_app*: Boolean, Home App (launcher app) usage.

  Defaults to false, your application will be listed as a Home App (launcher app) if true.

CHAPTER 4

Contribute

4.1 Write your own recipe

A recipe allows you to compile libraries / python extension for the mobile. Most of the time, the default compilation instructions doesn't work for the target, as ARM compiler / Android NDK introduce specificities that the library you want doesn't handle correctly, and you'll need to patch. Also, because the Android platform cannot load more than 64 inline dynamic libraries, we have a mechanism to bundle all of them in one to ensure you'll not hit this limitation.

To test your own recipe via Buildozer, you need to:

1. Fork Python for Android, and clone your own version (this will allow easy contribution later):

```
git clone https://github.com/YOURNAME/python-for-android
```

2. Change your *buildozer.spec* to reference your version:

```
p4a.source_dir = /path/to/your/python-for-android
```

3. Copy your recipe into *python-for-android/recipes/YOURLIB/recipe.sh*
4. Rebuild.

When you correctly get the compilation and your recipe works, you can ask us to include it in the python-for-android project, by issuing a Pull Request:

1. Create a branch:

```
git checkout --track -b recipe-YOURLIB origin/master
```

2. Add and commit:

```
git add python-for-android/recipes/YOURLIB/*
git commit -am 'Add support for YOURLIB`
```

3. Push to Github

   git push origin master

4. Go to *https://github.com/YOURNAME/python-for-android*, and you should see your new branch and a button "Pull Request" on it. Use it, write a description about what you did, and Send!

CHAPTER 5

Indices and tables

- genindex
- modindex
- search