
Buildozer Documentation

Release 0.11

Kivy's Developers

January 29, 2017

1	Installation	3
1.1	Targeting Android	3
2	Quickstart	5
2.1	Init and build for Android	5
2.2	Run my application	5
2.3	Install on non-connected devices	6
3	Specifications	7
3.1	Section [app]	7
4	Contribute	11
4.1	Write your own recipe	11
5	Indices and tables	13

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 manage 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](#).

Installation

Buildozer itself doesn't depend on any library, and works on Python 2.7 and ≥ 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:

```
pip install --upgrade buildozer
```

1.1 Targeting Android

If you target Android, you must install at least Cython, few build libs, and a Java SDK. Some binaries of the Android SDK are still in 32 bits, so you need few 32bits libraries available:

1.1.1 Android on Ubuntu 16.04 (64bit)

```
sudo pip install --upgrade cython==0.21
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential ccache git libncurses5:i386 libstdc++6:i386 libgtk2.0-0:i386 lib
```

1.1.2 Android on Ubuntu 15.10 (64bit)

```
sudo pip install --upgrade cython==0.21
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential ccache git libncurses5:i386 libstdc++6:i386 libgtk2.0-0:i386 lib
```

1.1.3 Android on Ubuntu 14.10 (64bit)

```
sudo pip install --upgrade cython==0.21
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential ccache git lib32stdc++6 lib32z1 lib32z1-dev python2.7 python2.7
```

1.1.4 Android on Ubuntu 13.10 (64bit)

```
sudo pip install --upgrade cython==0.21
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential ccache git lib32z1 lib32bz2-1.0 libncurses5:i386 libstdc++6:i386
```

1.1.5 Android on Ubuntu 12.04 (64bit)

```
sudo pip install --upgrade cython==0.21
sudo apt-get install build-essential ccache git lib32z1 lib32bz2-1.0 libncurses5:i386 libstdc++6:i386
```

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, those 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 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
```

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
```

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.include_exts*: List, files' extensions to include.

By default, not all files in your *source.dir* are included, but only some of them (*py,png,jpg,kv,atlas*), depending on the extension. Feel free to add your own extensions, or use an empty value if you want to include everything.

- *source.exclude_exts*: List, files' extensions to exclude.

In contrary to *source.include_exts*, you could include all the files you want except the ones that end with an extension listed in this token. If empty, no files will be excluded based on their extensions.

- *source.exclude_dirs*: List, directories to exclude.

Same as *source.exclude_exts*, but for directories. You can exclude your *tests* and *bin* directory with:

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

- *source.exclude_patterns*: List, files to exclude if they match a pattern.

If you have a more complex application layout, you might need a pattern to exclude files. It also works if you don't have a pattern. For example:

```
source.exclude_patterns = license, images/originals/*
```

- *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](#).

- *garden_requirements*: List, Garden packages to include.

Add here the list of Kivy's garden packages to include. For example:

```
garden_requirements = graph
```

Please note that if it doesn't work, it might be because of the garden package itself. Refer to the author of the package if he already tested it on your target platform, not us.

- *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. Ie, an 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*: String, orientation of the application.

Indicate the orientation that your application supports. Defaults to *landscape*, but can be changed to *portrait* or *all*.

- *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.

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 http://github.com/YOURNAME/python-for-android
```

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

```
android.p4a_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 <http://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!

Indices and tables

- `genindex`
- `modindex`
- `search`