Briefcase Documentation

Release 0.1

Russell Keith-Magee

Jul 06, 2017

Contents

Briefcase is a tool for converting a Python project into a standalone native application. It supports producing binaries for:

- macOS,
- Windows,
- iOS,
- Android, and
- tvOS.

CHAPTER 1

About Briefcase

The Briefcase Developer and User community

Briefcase is part of the BeeWare suite. You can talk to the community through:

- @pybeeware on Twitter
- The BeeWare Users Mailing list, for questions about how to use the
- BeeWare suite.
- The BeeWare Developers Mailing list, for discussing the development of
- new features in the BeeWare suite, and ideas for new tools for the suite.

Code of Conduct

The BeeWare community has a strict Code of Conduct. All users and developers are expected to adhere to this code.

If you have any concerns about this code of conduct, or you wish to report a violation of this code, please contact the project founder *Russell Keith- Magee*.

Contributing

If you experience problems with Briefcase, log them on GitHub. If you want to contribute code, please fork the code and submit a pull request.

Frequently Asked Questions

What version of Python does Briefcase support?

Broadly; Python 3. However, the exact versions supported vary depending on the platform being targeted.

Getting Started

In this guide we will walk you through setting up your Briefcase environment for development and testing. We will assume that you have a working Python install, and an existing project.

Install Briefcase

The first step is to install Briefcase. If you're using a virtual environment for your project, don't forget to activate it.

```
$ mkdir tutorial
$ cd tutorial
$ virtualenv -p $(which python3) env
$ . env/bin/activate
$ pip install briefcase
```

Next Steps

You now have a working Briefcase environment, so you can *start the first tutorial*.

CHAPTER 2

Tutorials

These tutorials are step-by step guides for using Briefcase.

Tutorial 0 - Hello, world!

In this tutorial, you'll take a really simple "Hello, world!" program written in Python, convert it into a working iOS project.

Setup

This tutorial assumes you've read and followed the instructions in *Getting Started*. If you've done this, you should have:

- XCode installed on your Mac,
- A `tutorial` directory,
- A activated Python 3.4 virtual environment,
- Briefcase installed in that virtual environment,

Start a new project

Let's get started by using the handy template `briefcase-template`:

```
$ pip install cookiecutter briefcase
$ cookiecutter https://github.com/pybee/briefcase-template
```

This will ask a bunch of questions of you. We'll use an *app_name* of "tutorial_0", and a *formal_name* of "Tutorial 0". Set the other values as you like

You'll now have a few files in this folder, including `tutorial_0`.

Check out what the provided `tutorial_0/app.py` file contains:

```
$ cd tutorial_0
$ cat tutorial_0/app.py
```

```
def main():
    # This needs to return an object that has a main_loop() method.
    return None
```

This won't do much as it is, but we can make it useful.

Add this into the `app.py` to make it useful:

```
class MyApp:
    def main_loop(self):
        print("Hello world")


def main():
    return MyApp()
```

Create an iOS project

It is all ready for using `briefcase`. You can invoke it, using:

```
$ python setup.py ios
```

to create the iOS app.

Open the iOS project with Xcode

There is a new folder in your project called 'iOS', which contains the Xcode project (`Hello World.xcodeproj`). Open it with Xcode and check that your application is the `app` folder. You can also open the application by running:

```
open iOS/Tutorial\ 0.xcodeproj
```

You can test the app by running it in Xcode. As our application only shows a message, the iOS application will show only a blank screen. You can check if it is working in the console logs, which should contain something like this:

```
Tutorial 0.app/Library/Application Support/com.pybee.tutorial0/tutorial_0/tutorial_0/
↪app.py
Hello World!
2016-09-16 10:49:14.564094 Hello World[6791:4292188] subsystem: com.apple.UIKit,␣
↪category: HIDEventFiltered, enable_level: 0, persist_level: 0, default_ttl: 0, info_
↪ttl: 0, debug_ttl: 0, generate_symptoms: 0, enable_oversize: 1, privacy_setting: 2,␣
↪enable_private_data: 0
```

And that is all, you created your first iOS python app!

Linux Tutorial 0 - Hello, world!

In this tutorial, you'll take a really simple "Hello, world!" program written in Python, convert it into a working Linux project.

Setup

This tutorial assumes you've read and followed the instructions in *Getting Started*. This Tutorial is for Linux Users! * A `tutorial` directory, * An activated Python 3 virtual environment, * Briefcase installed in that virtual environment,

Start a new project

Let's get started by using the handy template `briefcase-template`:

```
$ pip install cookiecutter briefcase
$ cookiecutter https://github.com/pybee/briefcase-template
```

This will ask a bunch of questions of you. We'll use an *app_name* of "tutorial_0", and a *formal_name* of "Tutorial 0". Set the other values as you like

You'll now have a few files in this folder, including `tutorial_0`.

Check out what the provided `tutorial_0/app.py` file contains:

```
(BeeWare)user$ cd tutorial_0
(BeeWare)user$ cat tutorial_0/app.py
```

```
def main():
    # This needs to return an object that has a main_loop() method.
    return None
```

This won't do much as it is, but we can make it useful.

Add this into the `app.py` to make it useful:

```
class MyApp:
    def main_loop(self):
        print("Hello world")


def main():
    return MyApp()
```

Create an Linux project

It is all ready for using `briefcase`. You can invoke it, using:

```
$ python setup.py linux
```

to create the Linux app.

Open the project

There is a new folder in your project called 'linux', which contains the project (tutorial_0). You can open the application by running:

```
./linux/tutorial_0
```

The app should run like this! .. code-block:: bash

$ ./linux/tutorial_0 Hello World!

And that is all, you created your first Linux python app!

Tutorial 1- Toga Hello, World

In this tutorial you will create a simple application using toga framework.

Update your iOS project

In this step we assume that you followed the *previous tutorial*. First at all, you can clean your previous app in your `iostutorial` folder:

```
rm -rf iOS/
```

We are going to use the Toga framework, so we have to include the `toga-ios` requirement in the `ios` section of `setup.py` script:

```
setup(name='HelloWorld',
    ...
    options = {
        ...
        'ios': {
            'app_requires': [
                'toga-ios'
            ]
        }
    }
)
```

And now you can update the application.

We're going to use a version of the Toga Fahrenheit to Celsius converter tutorial found at <https://toga.readthedocs.io/en/latest/tutorial/tutorial-0.html>, modified to be class-based:

```
import toga


class Converter(toga.App):
    def calculate(self, widget):
        try:
            self.c_input.value = (float(self.f_input.value) - 32.0) * 5.0 / 9.0
        except Exception:
            self.c_input.value = '???'

    def startup(self):
        self.main_window = toga.MainWindow(self.name)
        self.main_window.app = self

        # Tutorial 1
        c_box = toga.Box()
        f_box = toga.Box()
        box = toga.Box()

        self.c_input = toga.TextInput(readonly=True)
        self.f_input = toga.TextInput()
```

```
        c_label = toga.Label('Celcius', alignment=toga.LEFT_ALIGNED)
        f_label = toga.Label('Fahrenheit', alignment=toga.LEFT_ALIGNED)
        join_label = toga.Label('is equivalent to', alignment=toga.RIGHT_ALIGNED)

        button = toga.Button('Calculate', on_press=self.calculate)

        f_box.add(self.f_input)
        f_box.add(f_label)

        c_box.add(join_label)
        c_box.add(self.c_input)
        c_box.add(c_label)

        box.add(f_box)
        box.add(c_box)
        box.add(button)

        box.style.set(flex_direction='column', padding_top=10)
        f_box.style.set(flex_direction='row', margin=5)
        c_box.style.set(flex_direction='row', margin=5)

        self.c_input.style.set(flex=1)
        self.f_input.style.set(flex=1, margin_left=160)
        c_label.style.set(width=100, margin_left=10)
        f_label.style.set(width=100, margin_left=10)
        join_label.style.set(width=150, margin_right=10)
        button.style.set(margin=15)

        self.main_window.content = box
        self.main_window.show()


def main():
    return Converter('Converter', 'org.pybee.converter')
```

Create the iOS app

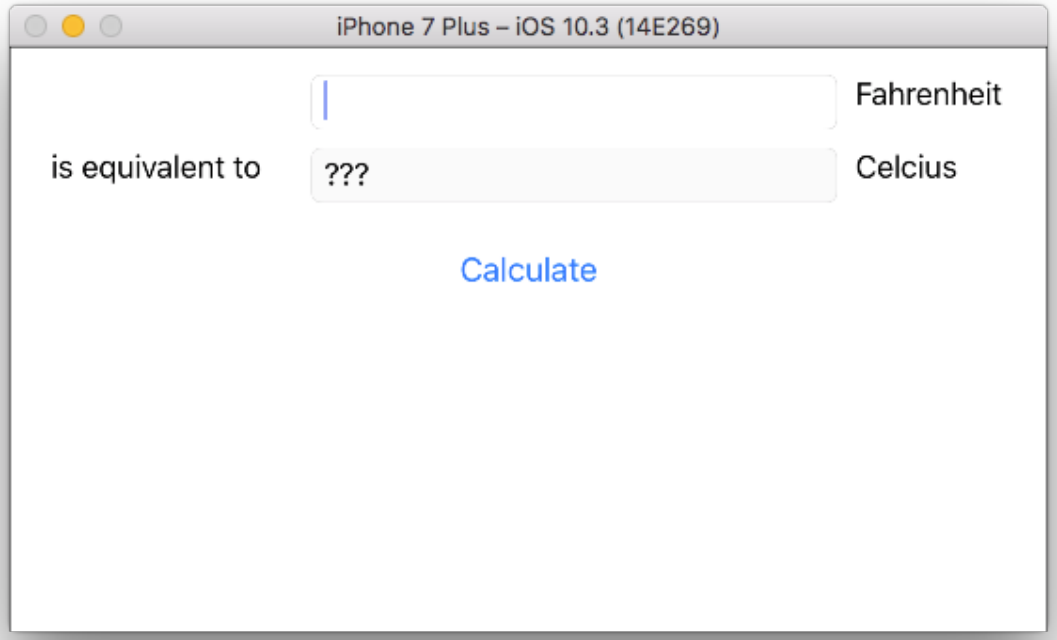
Now you can invoke setuptools again:

```
$ python setup.py ios
```

Notice that the `app_packages` is not empty after the update, and it contains toga packages and their requirements.

Open the iOS project with Xcode

If you the ios project in Xcode you will see a Toga application:



If you click on the button, you should see messages appear in the console.

Use the *same code*, but for the web

Edit the `setup.py` file to include a package helper for Django:

```
setup(name='HelloWorld',
    ...
    options = {
        ...
        'django': {
            'app_requires': [
                'toga-django'
            ]
        }
    }
)
```

Building Django has Javascript dependencies that are installed with NPM. Pick the appropriate command to install NPM on your OS, or visit the NPM website and follow the instructions.

```
$ brew install npm  # OSX with Homebrew (https://brew.sh)
$ apt-get install nodejs  # Ubuntu
$ pacman -S npm  # Arch Linux
```

Now you can invoke setuptools again:

```
$ python setup.py django
```

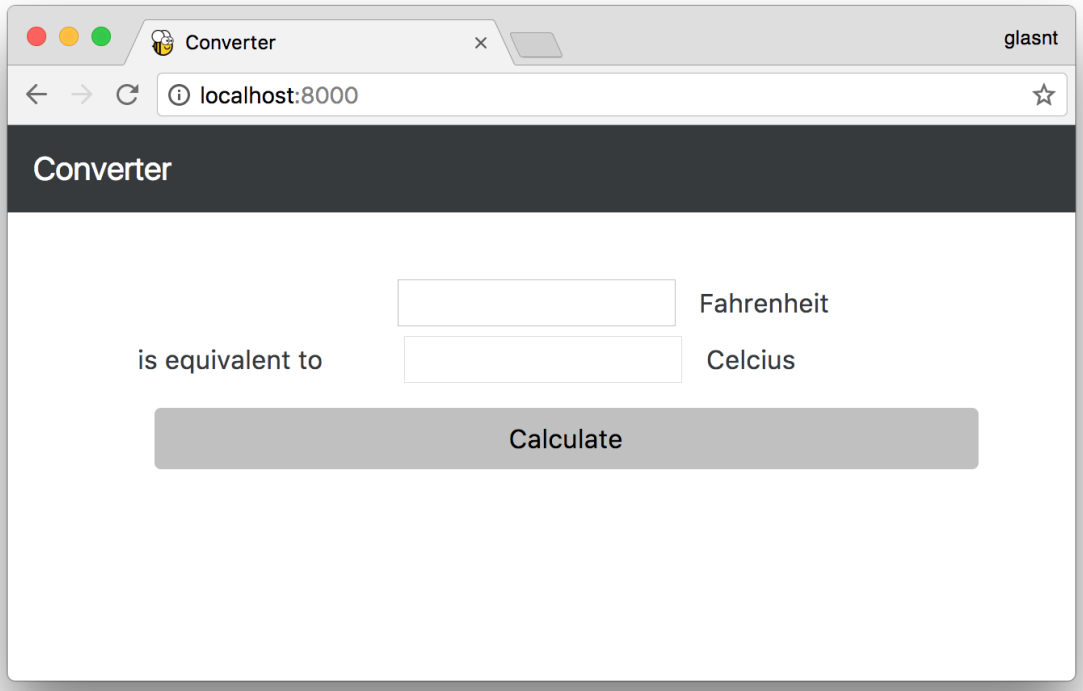
Once this process has completed, there are a couple of steps left (that are helpfully outputted by the last command) to setup the django project:

```
$ cd django
$ ./manage.py migrate
```

Then, we can run the application:

```
$ ./manage.py runserver
```

If you open up `localhost:8000` in your browser, you should see the same application running in the web.



CHAPTER 3

How-to guides

How-to guides are recipes that take the user through steps in key subjects. They are more advanced than tutorials and assume a lot more about what the user already knows than tutorials do, and unlike documents in the tutorial they can stand alone.

CHAPTER 4

Topic guides

Aimed at explaining (at a fairly high level) rather than doing.

CHAPTER 5

Reference

This is the technical reference for public APIs provided by Briefcase.

CHAPTER 6

Project Internals

Contributing to Briefcase

If you experience problems with Briefcase, log them on GitHub. If you want to contribute code, please fork the code and submit a pull request.

Setting up your development environment

The recommended way of setting up your development envrionment for Briefcase is to install a virtual environment, install the required dependencies and start coding. Assuming that you are using `virtualenvwrapper`, you only have to run:

```
$ git clone git@github.com:pybee/briefcase.git
$ cd briefcase
$ mkvirtualenv briefcase
```

Briefcase uses `unittest` (or `unittest2` for Python < 2.7) for its own test suite as well as additional helper modules for testing. To install all the requirements for Briefcase, you have to run the following commands within your virutal envrionment:

```
$ pip install -e .
```

Now you are ready to start hacking! Have fun!

Release History

0.1.8

- Modified template rollout process to avoid API limits on Github.

0.1.7

- Added check for existing directories, with the option to replace existing content.
- Added a Linux backend.
- Added a Windows backend.
- Added a splash screen for Android

0.1.6

- Added a Django backend (@glasnt)

0.1.5

- Added initial Android template
- Force versions of pip (>= 8.1) and setuptools (>=27.0)
- Drop support for Python 2

0.1.4

- Added support for tvOS projects
- Moved to using branches in the project template repositories.

0.1.3

- Added support for Android projects using VOC.

0.1.2

- Added support for having multi-target support projects. This clears the way for Briefcase to be used for watchOS and tvOS projects, and potentially for Python-OSX-support and Python-iOS-support to be merged into a single Python-Apple-support.

0.1.1

- Added support for app icons and splash screens.

0.1.0

Initial public release.

Briefcase Roadmap

Briefcase is a new project - we have lots of things that we'd like to do. If you'd like to contribute, providing a patch for one of these features:

- Windows support
- Linux support (for various distros)
- Apple watchOS support