Briefcase Documentation

Release 0.2.9

Russell Keith-Magee

Apr 13, 2019

Contents

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

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

CHAPTER 1

Table of contents

1.1 Tutorial

Get started with a hands-on introduction for beginners

1.2 How-to guides

Guides and recipes for common problems and tasks, including how to contribute

1.3 Background

Explanation and discussion of key topics and concepts

1.4 Reference

Technical reference - commands, modules, classes, methods

CHAPTER 2

Community

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

- @pybeeware on Twitter
- pybee/general on Gitter

2.1 Tutorials

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

2.1.1 Tutorial 0 - Hello, world!

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

Setup

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

- A `tutorial` directory,
- An activated Python 3.5+ virtual environment,
- Briefcase installed in that virtual environment,
- Any platform-specific dependencies installed.

Start a new project

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

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

This will ask a bunch of questions of you. We'll use an *app_name* of "helloworld", a *formal_name* of "Hello World", using the Toga GUI toolkit. You can use the default values for the other questions (or update them to reflect your own name if you want).

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

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

```
# If you are on macOS or Linux
$ cd helloworld
$ cat helloworld/app.py
```

```
# If you are on Windows command line
> cd helloworld
> type helloworld/app.py
```

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW


class HelloWorld(toga.App):
    def startup(self):
        # Create a main window with a name matching the app
        self.main_window = toga.MainWindow(title=self.name)

        # Create a main content box
        main_box = toga.Box()

        # Add the content on the main window
        self.main_window.content = main_box

        # Show the main window
        self.main_window.show()


def main():
    return HelloWorld('Hello World', 'com.example.helloworld')
```

Put it in a briefcase

Your project is now ready to use `briefcase`.

Windows

You must use Windows command line (`cmd`) to run this step. In case you have installed MingW32, Linux compatibility layer for Windows or similar, make sure it doesn't interfere with `cmd` environment.

To create and run the application, run:

```
> python setup.py windows -s
```

This will produce a `windows` subdirectory that will contain a `HelloWorld-0.0.1.msi` installer. If you get an error stating that Wix Tools cannot be found, and you have already installed them, try restarting your computer.

macOS

To create and run the application, run:

```
$ python setup.py macos -s
```

This will produce a `macOS` subdirectory that contains a `Hello World.app` application bundle. This bundle can be dragged into your Applications folder, or zipped and distributed to anyone else.

Linux

To create and run the application, run:

```
$ python setup.py linux -s
```

This will produce a `linux` subdirectory that contains a `Hello World` script that will start the application.

iOS

To create and run the application, run:

```
$ python setup.py ios -s
```

This will start the iOS simulator (you may be asked to select an API and a simulator device on which to run the app) and run your app.

It will also produce an `ios` subdirectory that contains an XCode project called `Hello World.xcodeproj`. You can open this project in XCode to run your application.

Android

To create and run the application, run:

```
$ python setup.py android -s
```

This will produce an `android` subdirectory that contains a Gradle project. It will also launch the app on the first Android device or simulator that can be found running on (or attached to) your computer.

What should happen

When the application runs, you should see a window with a title of "Hello World" appear. The window won't contain any content - but it will be a native application, with a native icon in your task bar (or wherever icons appear on your platform).

You've just packaged your first app with Briefcase! Now, let's *make the app actually do something interesting*.

2.1.2 Tutorial 1 - Fahrenheit to Celsius

In this tutorial we will make your application do something interesting.

Add code to your project

In this step we assume that you followed the *previous tutorial*. Put the following code into `helloworld\app.py`, replacing the old code:

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW, LEFT, RIGHT


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):
        # Create a main window with a name matching the app
        self.main_window = toga.MainWindow(title=self.name)

        # Create a main content box
        f_box = toga.Box()
        c_box = toga.Box()
        box = toga.Box()

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

        self.c_label = toga.Label('Celsius', style=Pack(text_align=LEFT))
        self.f_label = toga.Label('Fahrenheit', style=Pack(text_align=LEFT))
        self.join_label = toga.Label('Is equivalent to', style=Pack(text_align=RIGHT))

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

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

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

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

        box.style.update(direction=COLUMN, padding_top=10)
        f_box.style.update(direction=ROW, padding=5)
        c_box.style.update(direction=ROW, padding=5)

        self.c_input.style.update(flex=1)
        self.f_input.style.update(flex=1, padding_left=160)
        self.c_label.style.update(width=100, padding_left=10)
```

(continues on next page)

(continued from previous page)

```
        self.f_label.style.update(width=100, padding_left=10)
        self.join_label.style.update(width=150, padding_right=10)

        button.style.update(padding=15, flex=1)

        # Add the content on the main window
        self.main_window.content = box

        # Show the main window
        self.main_window.show()


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

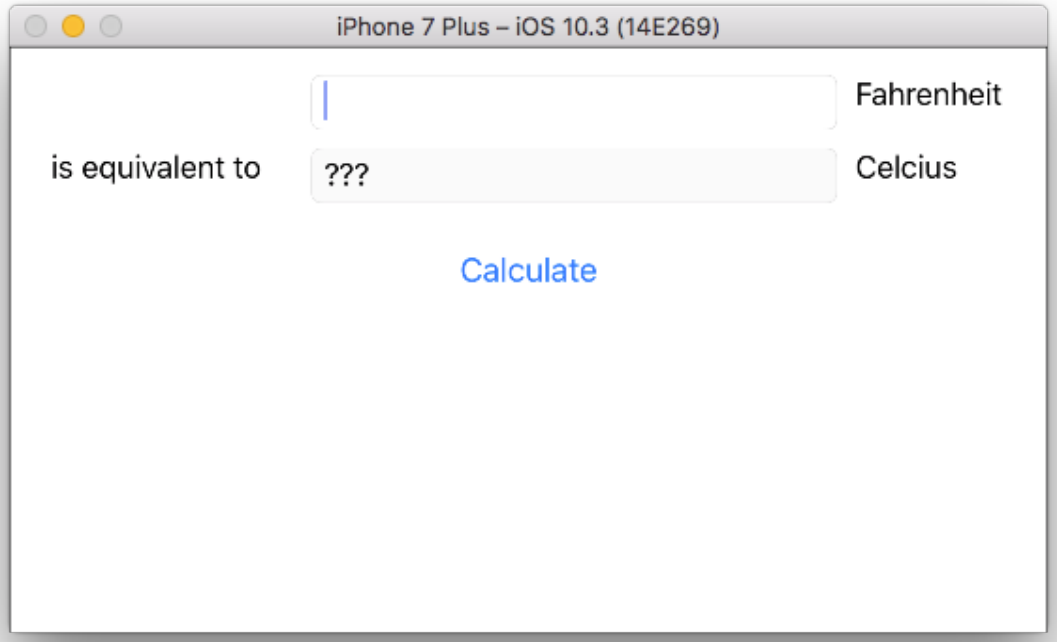
Build and run the app

Now you can invoke briefcase again:

```
$ python setup.py ios -s
```

replacing `ios` with your platform of choice.

You should see something that looks a bit like this:

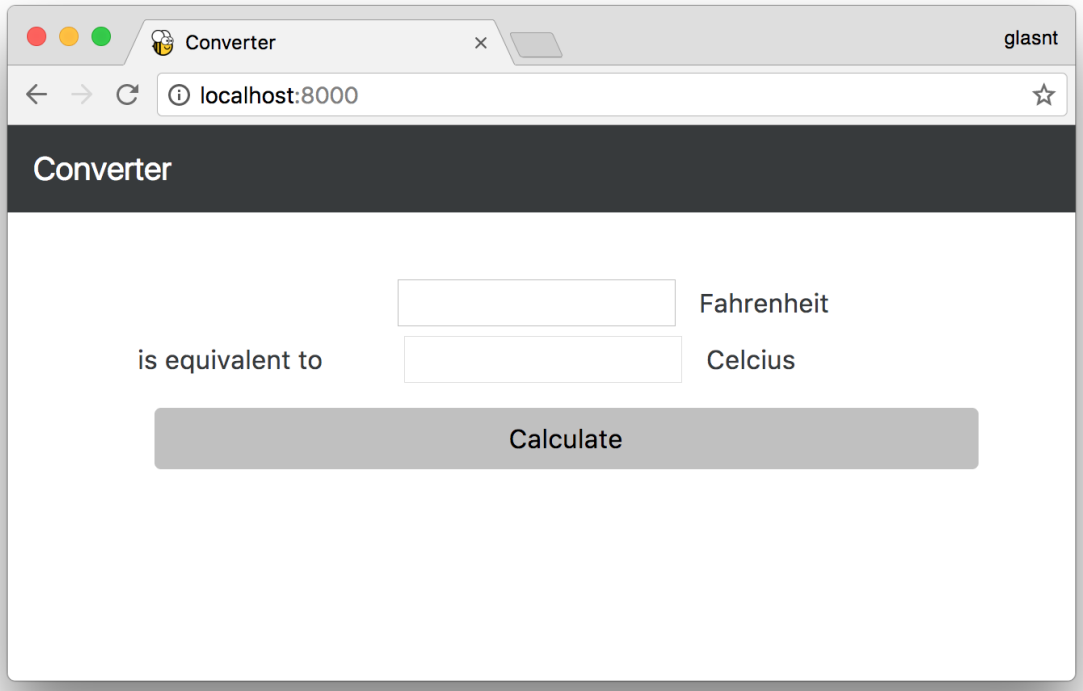


Use the *same code*, but for the web

Now, we're going to deploy the same code, but as a single page web application. Make sure you have the Django dependencies installed (see *Getting Started*), and run the following:

```
$ python setup.py django -s
```

This will gather all the Javascript dependencies, create an initial database, start a Django runserver, and launch a browser. You should see the same application running in your browser:



Note: If you get a "Server could not be contacted" error, it's possible your web browser started faster than the server; reload the page, and you should see the app.

2.2 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.

2.2.1 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:beeware/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:

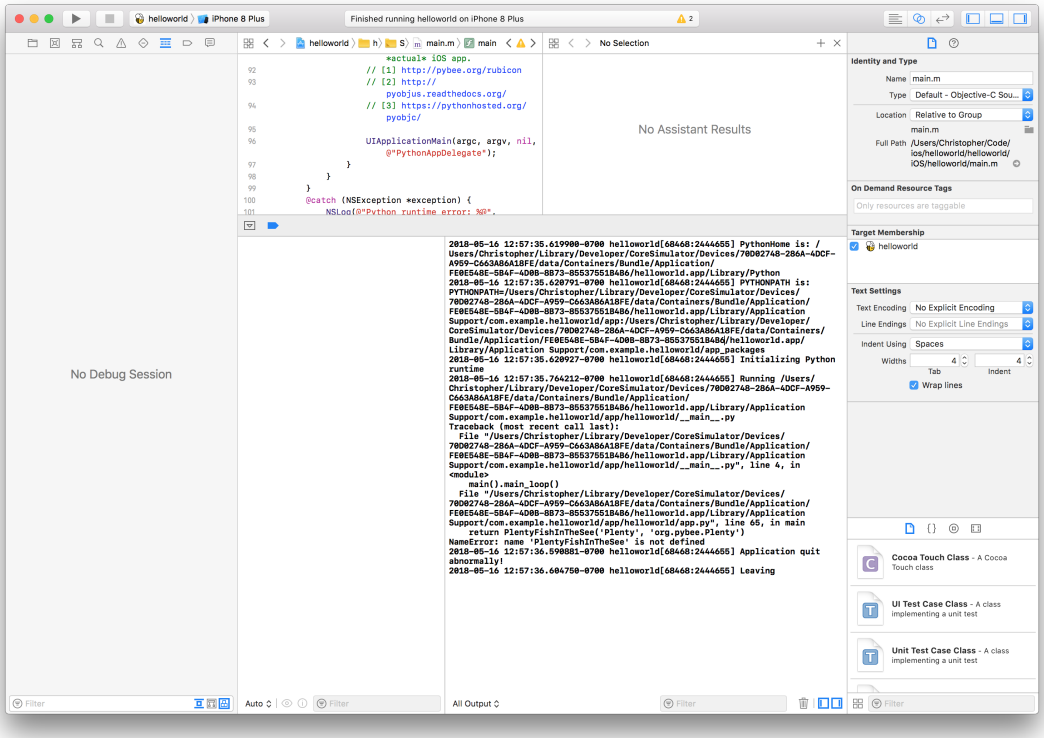
```
$ pip install -e .
```

Now you are ready to start hacking! Have fun!

2.2.2 See Errors on iOS

If you have a beeware iOS project that has a crash, it can be difficult to see the stacktrace. Here's how to do it -

1. Build your iOS project. You don't have to start it.
2. Open that iOS project in Xcode. Click the Run button (looks like an arrow) and wait for the simulator to open. Cause the app to crash.
3. Your stack trace ought to appear in the 'debugger area' at the bottom of the screen. If you can't see that area, you may have to activate it with `View` > `Debug Area` > `Show Debug Area`



2.3 About Briefcase

2.3.1 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.

How do I add a custom app icon to my app?

The `icon` attribute specifies the prefix of a path to a set of image files. The name specified will be appended with a number of suffixes to construct filenames for the various icon sizes needed on each platform. You should provide the following files:

- **On iOS:**
  - `$(icon)-180.png`, a 60x60@3x image (iPhone)
  - `$(icon)-167.png`, an 83.5x83.5@2x image (iPad Pro)
  - `$(icon)-152.png`, a 76x76@2x image (iPad)
  - `$(icon)-120.png`, a 40x40@3x/60x60@2x image (iPad, iPhone)
  - `$(icon)-87.png`, a 29x29@3x image (iPad, iPhone)
  - `$(icon)-80.png`, a 40x40@2x image (iPad, iPhone)
  - `$(icon)-76.png`, a 76x76 image (iPad)
  - `$(icon)-58.png`, a 29x29@2x image (iPad)
  - `$(icon)-40.png`, a 40x40 image (iPad)
  - `$(icon)-29.png`, a 29x29 image (iPad)
- **On Android:**
  - `$(icon)-192.png`, an xxxhdpi image (192x192)
  - `$(icon)-144.png`, an xxhdpi image (144x144)
  - `$(icon)-96.png`, an xhdpi image (96x96)
  - `$(icon)-72.png`, a hdpi image (72x72)
  - `$(icon)-48.png`, an mdpi image (48x48)
  - `$(icon)-36.png`, an ldpi image (36x36)
- **On macOS:**
  - `$(icon).icns`, a composite ICNS file containing all the required icons.
- **On Windows:**
  - `$(icon).ico`, a 256x256 ico file.
- **On Apple TV:**
  - `$(icon)-400-front.png`, a 400x240 image to serve as the front layer of an app icon.
  - `$(icon)-400-middle.png`, a 400x240 image to serve as the middle layer of an app icon.
  - `$(icon)-400-back.png`, a 400x240 image to serve as the back layer of an app icon.

  - `$(icon)-1280-front.png`, a 1280x768 image to serve as the front layer of an app icon.
  - `$(icon)-1280-middle.png`, a 1280x768 image to serve as the middle layer of an app icon.
  - `$(icon)-1280-back.png`, a 1280x768 image to serve as the back layer of an app icon.
  - `$(icon)-1920.png`, a 1920x720 image for the top shelf.

If a file cannot be found, an larger icon will be substituted (if available). If a file still cannot be found, the default briefcase icon will be used.

On Apple TV, the three provided images will be used as three visual layers of a single app icon, producing a 3D effect. As an alternative to providing a `-front`, `-middle` and `-back` variant, you can provide a single `$(icon)-(size).png`, which will be used for all three layers.

The `splash` attribute specifies a launch image to display while the app is initially loading. It uses the same suffix approach as image icons. You should provide the following files:

- **On iOS:**
  - `$(splash)-2048x1536.png`, a 1024x786@2x landscape image (iPad)
  - `$(splash)-1536x2048.png`, a 768x1024@2x portrait image (iPad)
  - `$(splash)-1024x768.png`, a 1024x768 landscape image (iPad)
  - `$(splash)-768x1024.png`, a 768x1024 landscape image (iPad)
  - `$(splash)-640x1136.png`, a 320x568@2x portrait image (new iPhone)
  - `$(splash)-640x960.png`, a 320x480@2x portrait image (old iPhone)
- **On Apple TV:**
  - `$(splash)-1920x1080.icns`, a 1920x1080 landscape image
- **On Android:**
  - `$(splash)-1280×1920.png`, an xxxhdpi (1280×1920) image
  - `$(splash)-960×1440.png`, an xxhdpi (960×1440) image
  - `$(splash)-640×960.png`, an xhdpi (640×960) image
  - `$(splash)-480x720.png`, a hdpi (480x720) image
  - `$(splash)-320×480.png`, an mdpi (320×480) image
  - `$(splash)-240×320.png`, an ldpi (240×320) image

If an image cannot be found, the default briefcase image will be used.

2.3.2 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.

2.3.3 Success Stories

Want to see examples of Briefcase in use? Here's some:

- Travel Tips is an app in the iOS App Store that was packaged for distribution using Briefcase.
- Mu is a simple code editor for beginner programmers. It uses Briefcase to prepare a macOS installer.

2.3.4 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. If you don't have an environment set up yet, this guide will help you get started.

Note: Briefcase (and the whole BeeWare toolchain) requires Python 3. Support for different Python 3 minor versions varies depending on the platform you're targeting; Python 3.5+ will give you the best results.

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
$ python3 -m venv venv
$ . venv/bin/activate
$ pip install briefcase
```

Note: On some versions the activate script may be in the venv/Scripts/ folder in which case swap: `$ . venv/bin/activate` for `$ . venv/Scripts/activate`

Install platform dependencies

Next, you'll need to make sure you've got the platform-specific dependencies for the platforms you're going to target.

Windows

- Install the WiX toolset.

If you're using Windows 10, you may need to enable the .NET 3.5 framework on your machine. Select "Programs and Features" from the Start menu, then "Turn Windows features on or off", and ensure ".NET Framework 3.5 (Includes .NET 2.0 and 3.0)" is enabled.

Mac OSX

There are no additional dependencies required to support OSX.

Linux

You'll need install GTK+ 3.10 or later. This is the version that ships starting with Ubuntu 14.04 and Fedora 20. You also need to install the Python 3 bindings to GTK+. If you want to use the WebView widget, you'll also need to have WebKit, plus the GI bindings to WebKit installed. This means you'll need to install the following:

- **Ubuntu 14.04** `apt-get install python3-gi gir1.2-webkit2-3.0`
- **Ubuntu 16.04** `apt-get install python3-gi gir1.2-webkit2-4.0` or `apt-get install python3-gi gir1.2-webkit2-3.0`
- **Fedora 20+** `dnf install python3-gobject pywebkitgtk` or `yum install python3-gobject pywebkitgtk`
- **Debian Stretch** `apt-get install python3-gi gir1.2-webkit2-4.0`

iOS

- Install XCode from the App store. Once you've installed XCode, you must also install the Xcode Command Line Tools. This can be done from the Preference panel within XCode itself.

Android

- Install Android Studio. When you start Android Studio for the first time, you'll be provided a wizard to configure your installation; select a "standard" installation.
- Put the *sdk/tools*, *sdk/platform-tools* and *sdk/tools/bin* directories in your path. - On macOS: *~/Library/Android/sdk/tools*, *~/Library/Android/sdk/platform-tools* and *~/Library/Android/sdk/tools/bin*
- Set the *ANDROID_SDK_HOME* directory - On macOS: *~/Library/Android/sdk*
- Update the SDKs:

```
$ sdkmanager --update
```

- Create a virtual device image, following these instructions.
- Install Gradle.
- Start the emulator:

```
$ emulator @Nexus_5X_API_22
```

Django

If you are going to create a web app with Django, you need:

- Install an LTS version of Node (6.9.x)
- Install NPM 4.x or higher

Next Steps

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

2.3.5 Quickstart

In your virtualenv, install Briefcase:

```
$ pip install briefcase
```

Then, add extra options to your `setup.py` file to provide the app-specific properties of your app. Settings that are applicable to any app can be set under the `app` key; platform specific settings can be specified using a platform key:

```
setup(
    ...
    options={
        'app': {
            'formal_name': 'My First App',
            'bundle': 'org.example',
        },
        'macos': {
            'app_requires': [
                'toga-cocoa'
            ],
            'icon': 'icons/macos',
        },
        'ios': {
            'app_requires': [
                'toga-ios'
            ],
            'icon': 'images/ios_icon',
            'splash': 'images/ios_splash',
        },
        'android': {
            'app_requires': [
                'toga-android'
            ],
            'icon': 'images/android_icon',
            'splash': 'images/android_splash',
        },
        'tvos': {
            'app_requires': [
                'toga-ios'
            ]
        },
        'django': {
            'app_requires': [
                'toga-django'
```

(continues on next page)

(continued from previous page)

```
            ]
        },
    }
)
```

At a minimum, you must set a `formal_name` key (the full, formal name for the app) and a `bundle` key (the bundle identifier for the author organization - usually a reverse domain name).

Alternatively, if you're starting from scratch, you can use cookiecutter to generate a stub project with the required content:

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

Then, you can invoke `briefcase`, using:

- macOS: `$ python setup.py macos`
- Windows: `$ python setup.py windows`
- Linux: `$ python setup.py linux`
- iOS: `$ python setup.py ios`
- Android: `$ python setup.py android`
- tvOS: `$ python setup.py tvos`

You can also use the `-b` (or `--build`) argument to automatically perform any compilation step required; or use `-s` (`--start`) to start the application.

For desktop OS's (macOS, Windows, Linux) the entry point(s) to your program can be defined in `setup.py` as console and gui scripts:

```
setup(
    ...
    entry_points={
        'gui_scripts': [
            'Example = example.gui:main [GUI]',
        ],
        'console_scripts': [
            'utility = example.main:main',
        ]
    }
    ...
```

For more details on the format see: http://setuptools.readthedocs.io/en/latest/setuptools.html#dynamic-discovery-of-services-and-plugins

On Windows and Linux this allows for multiple executables to be defined. macOS will use the entry point with the same name as your *formal_name* as the main application, any others will be available in the Contents/MacOS folder inside the application bundle.

For other platforms the entry point is defined in the platform template, typically they require the __main__.py module to be defined explicitly in code.

2.3.6 Release History

0.2.9

- Updated mechanism for starting the iOS simulator
- Added support for environment markers in `install_requires`
- Improved error handling when Wix isn't found

0.2.8

- Corrects packaging problem with urllib3, caused by inconsistency between requests and boto3.
- Corrected problems with Start menu targets being created on Windows.

0.2.7

- Added support for launch images for iPhone X, Xs, Xr, Xs Max and Xr Max
- Completed removal of internal pip API dependencies.

0.2.6

- Added support for registering OS-level document type handlers.
- Removed dependency on an internal pip API.
- Corrected invocation of gradlew on Windows
- Addressed support for support builds greater than b9.

0.2.5

- Restored download progress bars when downloading support packages.

0.2.4

- Corrected a bug in the iOS backend that prevented iOS builds.

0.2.3

- Bugfix release, correcting the fix for pip 10 support.

0.2.2

- Added compatibility with pip 10.
- Improved Windows packaging to allow for multiple executables
- Added a `--clean` command line option to force a refresh of generated code.
- Improved error handling for bad builds

0.2.1

- Improved error reporting when a support package isn't available.

0.2.0

- Added `-s` option to launch projects
- Switch to using AWS S3 resources rather than Github Files.

0.1.9

- Added a full Windows installer backend

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.

2.3.7 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

2.4 Reference

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