Rubicon Documentation

Release 0.2.6

Russell Keith-Magee

Oct 11, 2022

CONTENTS

Note: This project has been archived. It was used by Briefcase 0.3.9 and earlier to support Android projects; however, that role is now performed by Chaquopy.

Rubicon Java is a bridge between the Java Runtime Environment and Python. It enables you to:

- Instantiate objects defined in Java,
- Invoke static and instance methods on objects defined in Java,
- Access and modify static and instance fields on objects defined in Java, and
- Write and use Python implementations of interfaces defined in Java.

It also includes wrappers of the some key data types from the Java standard library (e.g., `java.lang.String`).

CHAPTER

ONE

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

TWO

COMMUNITY

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

- @pybeeware on Twitter
- beeware/general on Gitter

2.1 Tutorials

These tutorials are step-by step guides for using Rubicon Java.

2.1.1 Your first bridge

In this example, we're going to use Rubicon to access the Java standard library, and the `java.net.URL` class in that library. `java.net.URL` is the class used to represent and manipulate URLs.

This tutorial assumes you've set up your environment as described in the *Getting started guide*.

Accessing `java.net.URL`

TODO

Time to take over the world!

You now have access to *any* method, on *any* class, in any library, in the entire Java ecosystem! If you can invoke something in Java, you can invoke it in Python - all you need to do is:

- load the library with ctypes;
- register the classes you want to use; and
- Use those classes as if they were written in Python.

Next steps

The next step is to write your own classes, and expose them into the Java runtime. That's the subject of the *next tutorial*.

2.1.2 Writing your own class

Eventually, you'll come across an Java API that requires you to provide a class instance (usually one implementing an interface) as an argument. For example, when using Java GUI classes, you often need to define "handler" classes to describe how a GUI element will respond to mouse clicks and key presses.

TODO

Next steps

???

2.1.3 Tutorial 1 - Your first bridge

In *Your first bridge*, you will use Rubicon to invoke an existing Java library on your computer.

2.1.4 Tutorial 2 - Writing your own class

In *Writing your own class*, you will write a Python class, and expose it to the Java runtime.

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 Getting Started

TODO

2.2.2 How to contribute code to Rubicon

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

Set up your development environment

The recommended way of setting up your development environment for Rubicon is to install a virtual environment, install the required dependencies and start coding:

```
$ python3 -m venv venv
$ source venv/bin/activate.sh
(venv) $ python -m pip install --upgrade pip
(venv) $ python -m pip install --upgrade setuptools
(venv) $ git clone https://github.com/beeware/rubicon-java.git
(venv) $ cd rubicon-java
(venv) $ python -m pip install -e .
```

Rubicon uses tox to describe it's testing environment. To install tox, run:

```
(venv) $ python -m pip install tox
```

You can then run the full test suite:

```
(venv) $ tox
```

By default this will run the test suite multiple times, once on each Python version supported by Rubicon, as well as running some pre-commit checks of code style and validity. This can take a while, so if you want to speed up the process while developing, you can run the tests on one Python version only:

```
(venv) $ tox -e py
```

Or, to run using a specific version of Python:

```
(venv) $ tox -e py37
```

substituting the version number that you want to target. You can also specify one of the pre-commit checks *flake8*, *docs* or *package* to check code formatting, documentation syntax and packaging metadata, respectively.

Now you are ready to start hacking on Rubicon. Have fun!

2.2.3 Contributing to the documentation

Here are some tips for working on this documentation. You're welcome to add more and help us out!

First of all, you should check the Restructured Text (reST) and Sphinx CheatSheet to learn how to write your .rst file.

Create a .rst file

Look at the structure and choose the best category to put your .rst file. Make sure that it is referenced in the index of the corresponding category, so it will show on in the documentation. If you have no idea how to do this, study the other index files for clues.

Build documentation locally

To build the documentation locally, *set up a development environment*, and run:

```
$ tox -e docs
```

The output of the file should be in the `build/sphinx/html` folder. If there are any markup problems, they'll raise an error.

2.3 Background

Want to know more about the Rubicon project, it's history, community, and plans for the future? That's what you'll find here!

2.3.1 Why "Rubicon"?

So... why the name Rubicon?

The Rubicon is a river in Italy. It was of importance in ancient times as the border of Rome. The Roman Army was prohibited from crossing this border, as that would be considered a hostile act against the Roman Senate.

In 54 BC, Julius Caesar marched the Roman Army across the Rubicon, signaling his intention to overthrow the Roman Senate. As he did so, legend says he uttered the words "Alea Iacta Est" - The die is cast. This action led to Julius being crowned as Emperor of Rome, and the start of the Roman Empire.

Of course, in order to cross any river, you need to use a bridge.

This project provides a bridge between the open world of the Python ecosystem, and the Java ecosystem.

2.3.2 The Rubicon Java Developer and User community

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

- @pybeeware on Twitter
- The beeware/general channel on Gitter.

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 Rubicon, 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 Rubicon in use? Here's some:

2.3.4 Release History

0.2.6 (2022-01-06)

Features

- Added support for arrays in arguments and return values. (#55)
- Added the ability to cast Java objects from one class to another. (#58)
- Local JNI Instances can be converted into global JNI instances using `__global__()` (#59)
- Added support for Python 3.9 and 3.10. (#62, #66)
- Methods that accept a Java `NULL` as an argument are now supported. (#63)
- Methods that return `NULL` now return typed `NULL` objects. (#67)

0.2.5 (2021-01-05)

Features

- Added support for passing Python lists into Java numeric array types (#53)

0.2.4 (2020-08-06)

Features

- Added support for implementing Java interfaces that return `bool`. (#52)

0.2.3 (2020-07-27)

Bugfixes

- The asyncio event loop can now start on Python 3.6. (#51)

0.2.2 (2020-07-03)

Features

- Python's AsyncIO event loop is now integrated with the Android event loop. (#40, #49)
- `sys.stdout` & `sys.stderr` are now routed to the Android debug log using an `android` Python module. (#44)

Misc

- #45, #46, #47, #48

0.2.1 (2020-06-17)

Features

- Add the ability to implement Java interface methods that return *int*. (#42)

Misc

- #31, #37

0.2.1 (2020-06-17)

Features

- Add the ability to implement Java interface methods that return *int*. (#42)

Misc

- #31, #37

0.2.0

Changes since v0.1.0:

- Port to Python 3, removing Python 2 support.
- Add support to Linux, allowing it to run on both macOS and Linux.
- Enable cross-compiling by adding support in the `Makefile` for specifying the compiler to use, the Python version whose headers to use, and to not require executing Python code during the build process.
- Adjust `Python.run()` to take a module name, not a filename.
- Add more documentation, although some is skeletal. Improvements to this are welcome.
- Add towncrier support. Future releases will rely on towncrier for release notes.
- Rename `pybee` to `beeware`.
- Add GitHub Actions configuration for testing on macOS & Linux and Python 3.5, 3.6, 3.7, and 3.8.

- Support the `JAVA_HOME` environment variable to choose a Java compiler.

This version specifically targets Java 8 to simplify Android support.

Thanks to the contributors, listed by GitHub username in alphabetical order:

- @freakboy3742
- @glasnt
- @jacebrowning
- @paulproteus
- @RomanKharin

2.3.5 Rubicon Roadmap

2.4 Reference

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