5 STEPS TO BUILD PYTHON NATIVE GUI WIDGETS FOR

BeeWare is

CROSS-PLATFORM

NATIVE

APP DEVELOPMENT

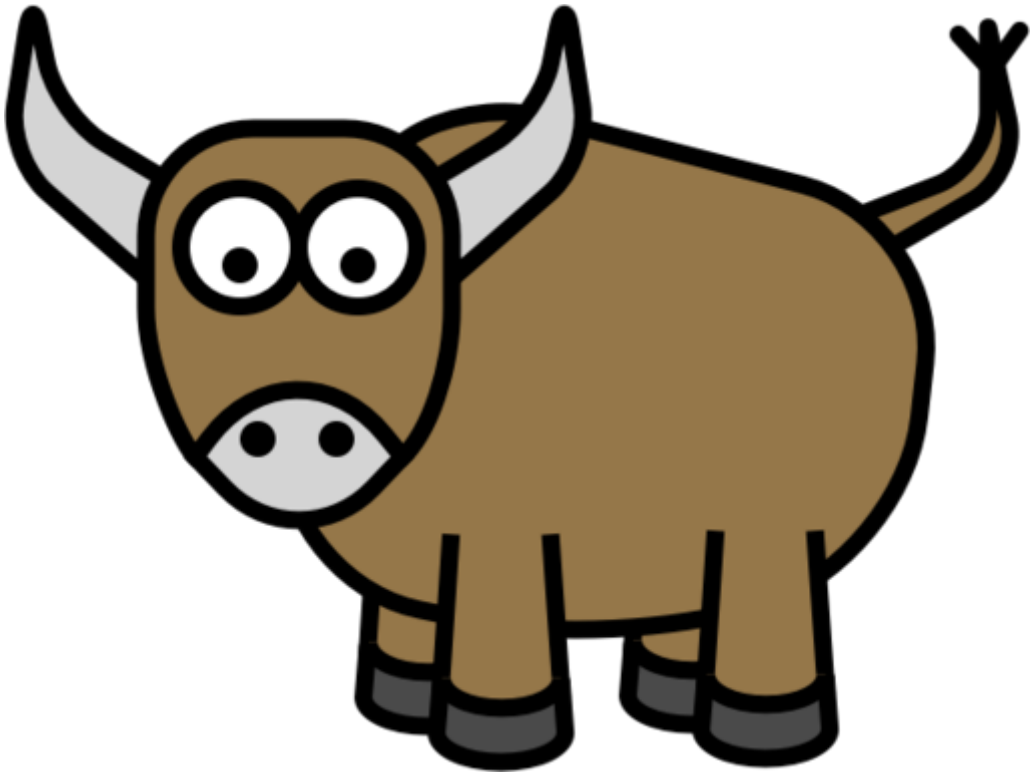
WITH PYTHON

and

SIMPLE APP DEPLOYMENT

TOGA

BEEWARE'S GUI TOOLKIT

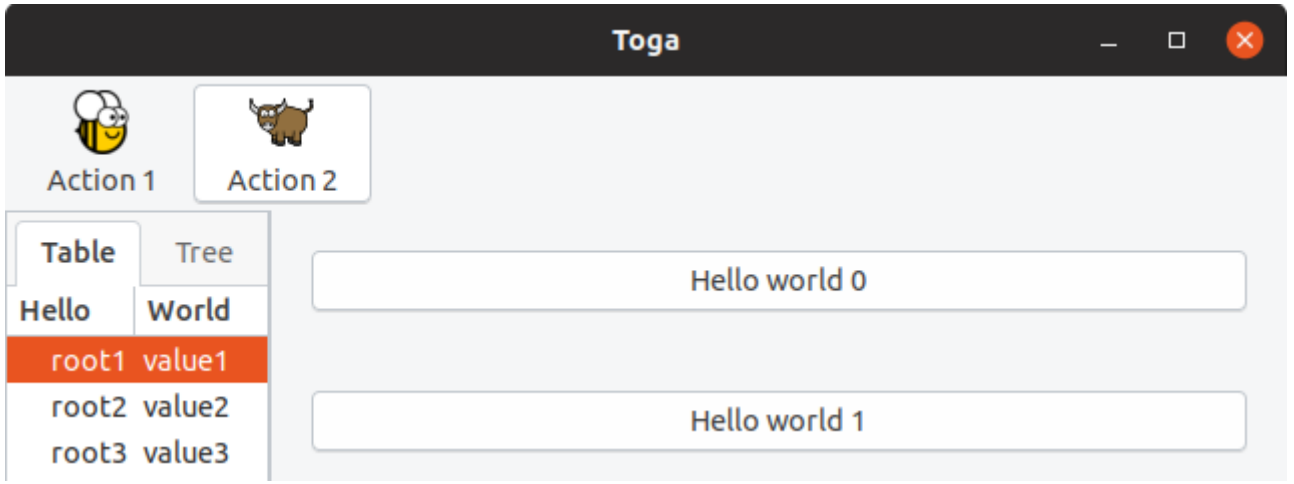


HELLO WORLD

```
import toga


class HelloWorld(toga.App):
    def startup(self):
        self.main_window = toga.MainWindow(title=self.name)
        main_box = toga.Box()
        self.main_window.content = main_box
        self.main_window.show()


def main():
    return HelloWorld('Hello World', 'org.pybee.helloworld')
```

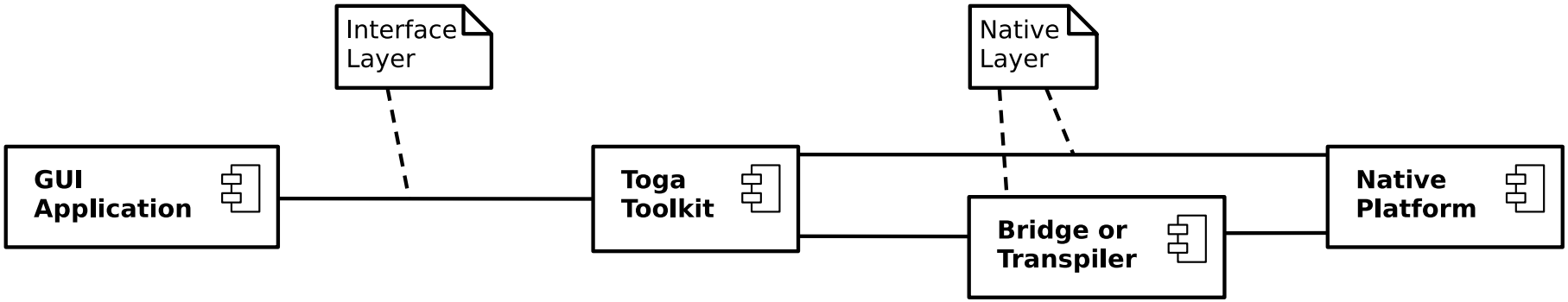
BACKGROUND

A widget is the controls and logic that a user interacts with when using a GUI

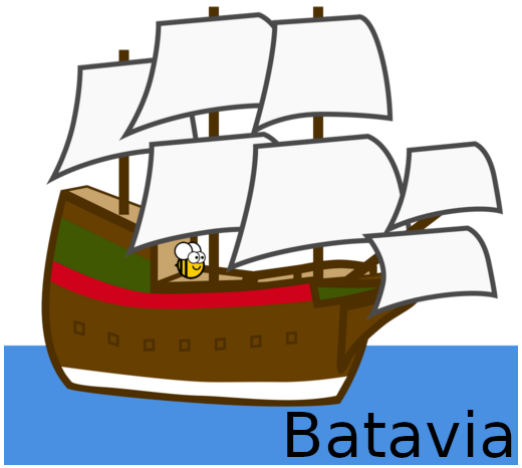


A Canvas widget will be used as an example

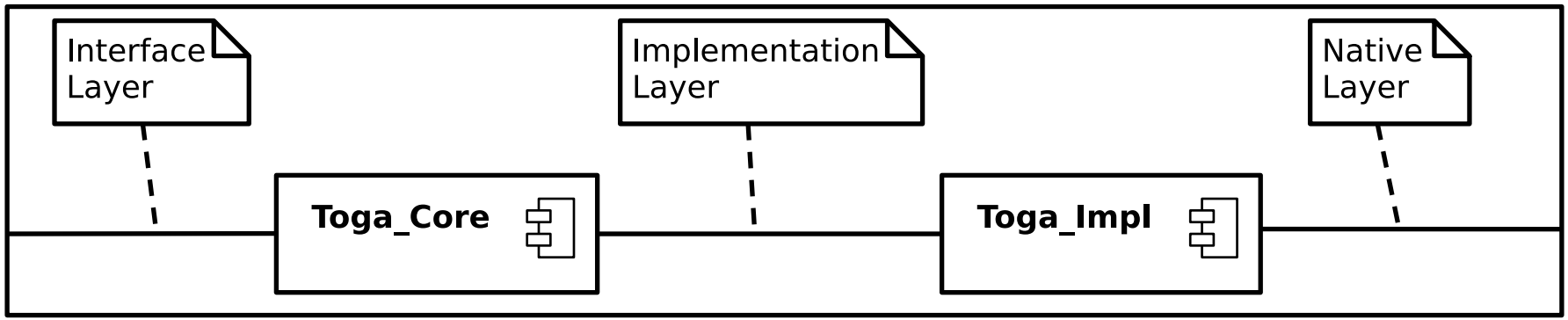
TOGA BLACKBOX



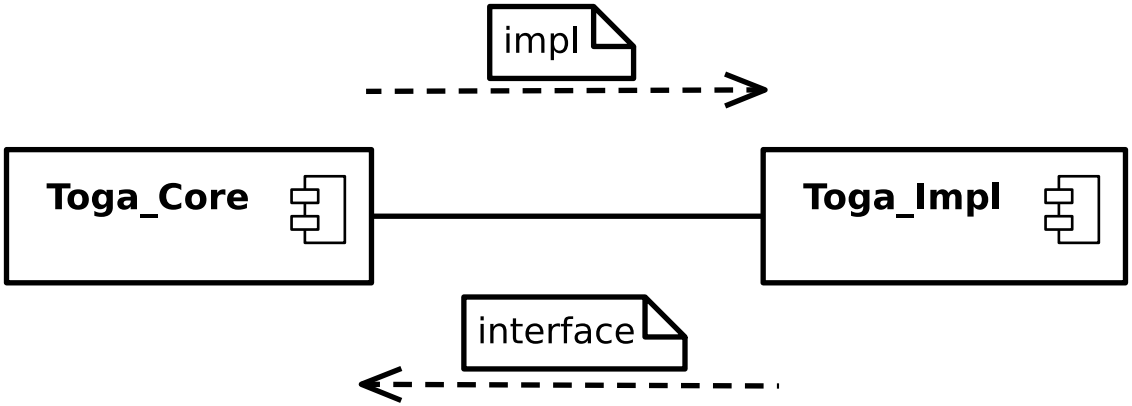
BRIDGE OR TRANSPILER



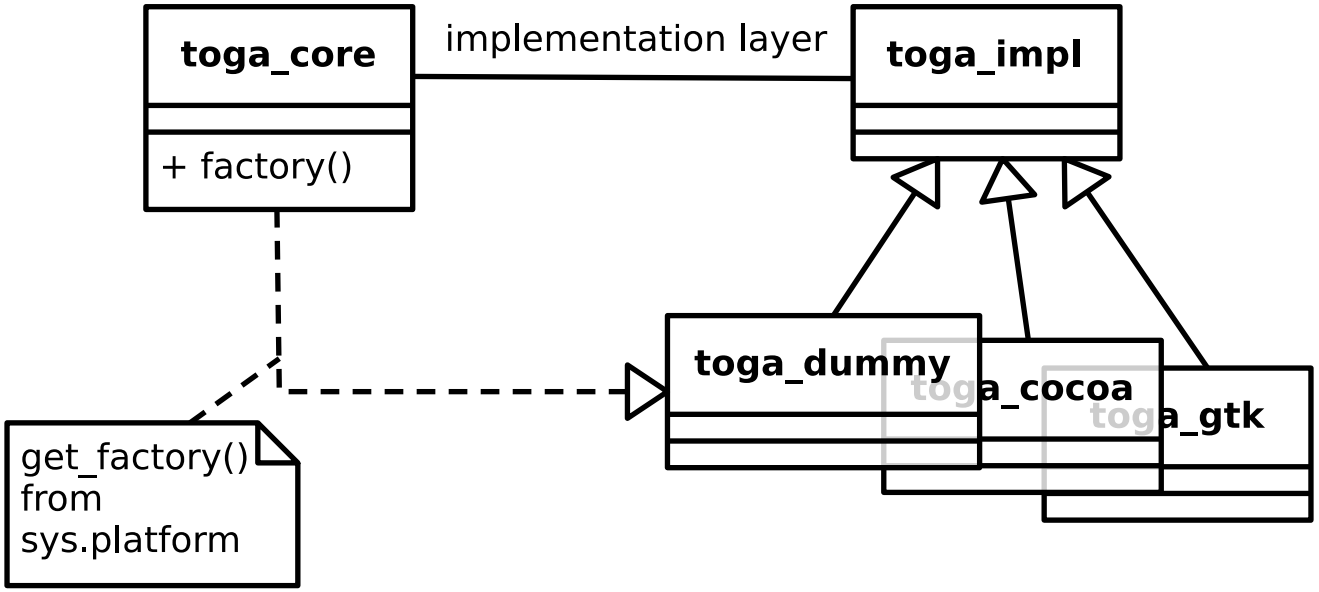
TOGA WHITEBOX



MORE TERMS



TOGA_IMPL FACTORY PATTERN



STEP 0

DEVELOPMENT PLATFORM

- Normally pick the platform that you are most familiar with
- macOS and GTK are the most developed
- Is this a mobile only widget (camera, GPS, etc)?

STEP 1

RESEARCH YOUR WIDGET

- Abstraction requires knowledge of specific examples
- Create use cases or user stories
- Get feedback

RESEARCH YOUR WIDGET

Tkinter

```
canvas = tk.Canvas()
canvas.create_rectangle(10, 10, 100, 100, fill="red")
canvas.pack()
```

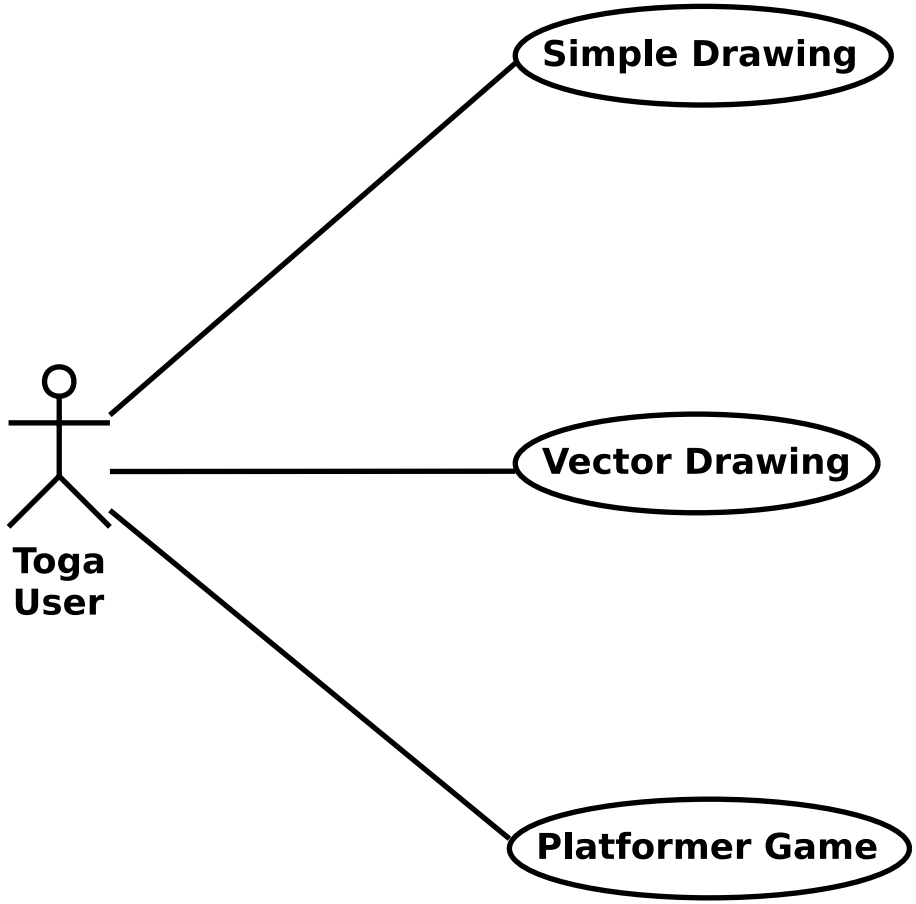
RESEARCH YOUR WIDGET

GTK

```
drawingarea = Gtk.DrawingArea()
drawingarea.connect("draw", draw)
def draw(da, ctx):
    ctx.set_source_rgb(200, 0, 0)
    ctx.rectangle(10, 10, 100, 100)
    ctx.fill()
```

RESEARCH YOUR WIDGET

Use Cases



STEP 2

WRITE DOCS

- Write your API documentation first
- The API provides the set of clearly defined methods of communication (layers) between the software components
- Documentation Driven Development
- This is iterative with Step 1

WRITE DOCS

```
The canvas is used for creating a blank widget that you can
draw on.

## Usage

An example of simple usage is to draw a colored rectangle on
the screen using the `rect` drawing object:

import toga
canvas = toga.Canvas(style=Pack(flex=1))
with canvas.fill(color=rgb(200, 0, 0)) as fill:
    fill.rect(10, 10, 100, 100)
```

WRITE CODE OUTLINE / DOCSTRINGS

```
class Canvas(Widget):
    """Create new canvas.

    Args:
        id (str):  An identifier for this widget.
        style (:obj:`Style`): An optional style object.
        factory (:obj:`module`): A python module that is
            capable to return a implementation of this class.
```

STEP 3

IMPLEMENT TOGA_CORE

(WITH TDD)

- First write tests for Toga_core
- Then code the outline created in Step 2

WRITE TESTS FOR TOGA_CORE

```
def test_widget_created():
    assertEqual(canvas._impl.interface, canvas)
    self.assertActionPerformed(canvas, "create Canvas")
```

WRITE TESTS FOR TOGA_CORE

```
def test_rect_modify():
    rect = canvas.rect(-5, 5, 10, 15)
    rect.x = 5
    rect.y = -5
    rect.width = 0.5
    rect.height = -0.5
    canvas.redraw()
    self.assertActionPerformedWith(
            canvas, "rect", x=5, y=-5, width=0.5, height=-0.5
        )
```

CODE TOGA_CORE

```
class Canvas(Widget):
    def __init__(self, id=None, style=None, factory=None):
        super().__init__(id=id, style=style, factory=factory)

        self._impl = self.factory.Canvas(interface=self)

    def rect(self, x, y, width, height):
        self.impl.rect(
            self.x, self.y, self.width, self.height
        )
```

STEP 4

IMPLEMENT TOGA_IMPL

DUMMY BACKEND

- Dummy is for automatic testing without a native platform
- Code the implementation layer API endpoint, create a method for each call of the API
- Check that all tests now pass

IMPLEMENT TOGA_IMPL

DUMMY BACKEND

```
class Canvas(Widget):
    def create(self):
        self._action("create Canvas")

    def rect(self, x, y, width, height):
        self._action(
            "rect", x=x, y=y, width=width, height=height
        )
```

STEP 5

IMPLEMENT TOGA_IMPL

YOUR PLATFORM

- Copy toga_dummy and create a new endpoint for the platform you chose in Step 1
- Make use of the native interface API for this widget on your platform

IMPLEMENT TOGA_IMPL

YOUR PLATFORM

```
class Canvas(Widget):
    def create(self):
        self.native = Gtk.DrawingArea()
        self.native.connect("draw", self.gtk_draw_callback)

    def gtk_draw_callback(self, canvas, gtk_context):
        self.interface._draw(self, draw_context=gtk_context)

    def rect(self, x, y, width, height, draw_context):
        draw_context.rectangle(x, y, width, height)
```

IMPLEMENT TOGA_IMPL

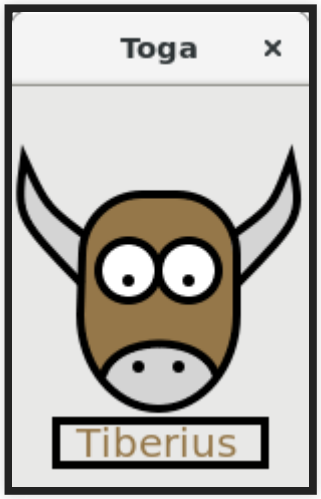
OTHER PLATFORMS

```
class TogaCanvas(NSView):
    @objc_method
    def drawRect_(self, rect: NSRect) -> None:
        context = NSGraphicsContext.currentContext.graphicsPort()


class Canvas(Widget):
    def create(self):
        self.native = TogaCanvas.alloc().init()

    def rect(self, x, y, width, height, draw_context, *args, **kwargs):
        rectangle = CGRectMake(x, y, width, height)
        core_graphics.CGContextAddRect(draw_context, rectangle
```

ITERATE

ITERATE THROUGH STEPS 1-5 TO COMPLETE YOUR WIDGET IMPLEMENTATION

SUBMIT A PULL REQUEST!



SUMMARY

1. Research Your Widget
2. Write Docs
3. Toga_core
4. Toga_impl - Dummy Backend
5. Toga_impl - Your Platform

@danyeaw

github.com/danyeaw

dan.yeaw.me

linkedin.com/in/danyeaw

dan@yeaw.me