Toga Documentation

Release 0.1

Russell Keith-Magee

August 23, 2014

Contents

A Python native, OS native GUI toolkit.

CHAPTER 1

Quickstart

In your virtualenv, install Toga, and then run it:

```
$ pip install toga-demo
$ toga-demo
```

This will pop up a GUI window showing the full range of widgets available to an application using Toga.

1.1 Problems using virtualenv under Ubuntu

Toga uses the system native python GTK+3 bindings for display purposes. However, if you're using a *–no-site-packages* virtualenv, the Python bindings for GTK won't be in your *PYTHONPATH*.

Unfortunately, you can't *pip install* GTK+ bindings, so you have to use a workaround. To make the system GTK+ bindings available to your virtualenv, symlinking the *gi* module from the system dist-packages directory into your virtualenv's site-packages:

```
$ cd <your virtualenv dir>/lib/python2.7/site-packages
$ ln -si /usr/lib/python2.7/dist-packages/gi
```

CHAPTER 2

Documentation

Documentation for Toga can be found on Read The Docs.

CHAPTER 3

Community

Toga 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.

Contents:

3.1 Why Toga?

Toga isn't the worlds first widget toolkit - there are dozens of other options. So why build a new one?

3.1.1 Native widgets - not themes

Toga uses native system widgets, not themes. When you see a Toga app running, it doesn't just *look* like a native app - it *is* a native app. Applying an operating system-inspired theme over the top of a generic widget set is an easy way for a developer to achieve a cross-platform goal, but it leaves the end user with the mess.

It's easy to spot apps that have been built using themed widget sets - they're the ones that don't behave quite like any other app. Widgets don't look *quite* right, or there's a menu bar on a window in an OS X app. Themes can get quite close - but there's always telltale signs.

On top of that, native widgets are almost faster than a themed generic widget. After all, you're using native system capability that has been tuned and optimized, not a drawing engine layered that's been layered on top of a generic widget.

3.1.2 Abstract the broad concepts

It's not enough to just look like a native app, though - you need to *feel* like a native app as well.

A "Quit" option under a "File" menu makes sense if you're writing a Windows app - but it's completely out of place if you're on OS X - the Quit option should be under the application menu.

And besides - why did the developer have to code the location of a Quit option anyway? Every app in the world has to have a quit option, so why doesn't the widget toolkit provide a quit option pre-installed, out of the box?

Although Toga uses 100% native system widgets, that doesn't mean Toga is just a wrapper around system widgets. Wherever possible, Toga attempts to abstract the broader concepts underpinning the construction of GUI apps, and build and API for *that*. So - every toga app has the basic set of menu options you'd expect of every app - Quit, About, and so on - all in the places you'd expect to see them in a native app.

When it comes to widgets, sometimes the abstraction is simple - after all, a button is a button, no matter what platform you're on. But other widgets may not be exposed so literally. What the Toga API aims to expose is a set of mechanisms for achieving a UI goals, not a literal widget set.

3.1.3 Python native

Most widget toolkits start their life as a C or C++ layer, which is then wrapped by other languages. As a result, you end up with APIs that taste like C or C++.

Toga has been designed from the ground up to be a Python native widget toolkit. This means the API is able to exploit language level features like generators and context managers in a way that a wrapper around a C library wouldn't be able to (at least, not easily).

This also means supporting Python 3. Toga supports both Python 2 and Python 3.

3.1.4 *pip install* and nothing more

Toga aims to be no more than a *pip install* away from use. It doesn't require the compilation of C extensions. There's no need to install a binary support library. There's no need to change system paths and environment variables. Just install it, import it, and start writing (or running) code.

3.1.5 Embrace mobile

10 years ago, being a cross-platform widget toolkit meant being available for Windows, OS X and Linux. These days, mobile computing is much more important. But despite this, there aren't many good options for Python programming on mobile platforms, and cross-platform mobile coding is still elusive. Toga aims to correct this.

So... why the name Toga?

We all know the aphorism that "When in Rome, do as the Romans do."

So - what does a well dressed Roman wear? A toga, of course! And what does a well dressed Python app wear? Toga!

So... why the yak mascot?

It's a reflection of the long running joke about yak shaving in computer programming. The story originally comes from MIT, and is related to a Ren and Stimpy episode; over the years, the story has evolved, and now goes something like this:

> You want to borrow your neighbours hose so you can wash your car. But you remember that last week, you broke their rake, so you need to go to the hardware store to buy a new one. But that means driving to the hardware store, so you have to look for your keys. You eventually find your keys inside a tear in a cushion - but you can't leave the cushion torn, because the dog will destroy the cushion if they find a little tear. The cushion needs a little more stuffing before it can be repaired, but it's a special cushion filled with exotic Tibetan yak hair.
>
> The next thing you know, you're standing on a hillside in Tibet shaving a yak. And all you wanted to do was wash your car.

An easy to use widget toolkit is the yak standing in the way of progress of a number of PyBee projects, and the original creator of Toga has been tinkering with various widget toolkits for over 20 years, so the metaphor seemed appropriate.

Lets get started!

Enough theory (and bad puns...) - lets get started with your first Toga app!

3.2 Your first Toga app

In this example, we're going to build a desktop app with a single button, that prints to the console when you press the button.

Here's a complete code listing for our "Hello world" app:

```
from __future__ import print_function, unicode_literals, absolute_import

import toga

def button_handler(widget):
    print("hello")

if __name__ == '__main__':

    app = toga.App('First App', 'org.pybee.helloworld')

    container = toga.Container()

    button = toga.Button('Hello world', on_press=button_handler)

    container.add(button)

    container.constrain(button.TOP == container.TOP + 50)
    container.constrain(button.LEADING == container.LEADING + 50)
    container.constrain(button.TRAILING + 50 == container.TRAILING)
    container.constrain(button.BOTTOM + 50 < container.BOTTOM)

    app.main_window.content = container

    app.main_loop()
```

Lets walk through this one line at a time.

The code starts with imports. First, we have some `__future__` imports. These to make Python 2 behave a bit more like Python 3. If you're using Python 3, you can omit this line:

```
from __future__ import print_function, unicode_literals, absolute_import
```

Next, we import toga:

```
import toga
```

Then, we set up a handler - a wrapper around behavior that we want to activate when the button is pressed. A handler is just a function. The function takes the widget that was activated as the first argument; depending on the type of event that is being handled, other arguments may also be provided. In the case of a simple button press, however, there are no extra arguments:

```
def button_handler(widget):
    print("hello")
```

Now we get into the main body of the app, where we lay out the UI. First, we create the app itself - this is a high level container representing the executable. The app has a name, and a unique identifier. The identifier is used when registering any app-specific system resources. By convention, the identifier is a "reversed domain name":

```
if __name__ == '__main__':

    app = toga.App('First App', 'org.pybee.helloworld')
```

By creating an app, we're declaring that we want to have a main window, with a main menu. However, we've said nothing about the content of the main window.

We want to put a button in the window. However, unless we want the button to fill the entire app window, we can't just put the button into the app window. Instead, we need create a container, and put the button in the container.

A container is an object that can be used to hold multiple widgets, and to define padding around widgets. So, we define a container:

```
container = toga.Container()
```

We can then define a button. When we create the button, we set can the button text, and we also set the behavior that we want to invoke when the button is pressed, referencing the handler that we defined earlier:

```
button = toga.Button('Hello world', on_press=button_handler)
```

Then, we add the button to the container:

```
container.add(button)
```

Now we have to define where the button will sit inside the container. Many widget toolkits do this by specifying an exact pixel position, or by specifying a box model (usually a grid, or some sort of box packing structure).

Toga, however, uses a constraint-based approach. To define how a container is laid out, you specify the spatial relationships between the container and the widget, or between the widget and other widgets.

This is done using the *constrain()* method on a container; the *constrain()* call takes expressions that define the relationships you want to impose:

```
container.constrain(button.TOP == container.TOP + 50)
container.constrain(button.LEADING == container.LEADING + 50)
container.constrain(button.TRAILING + 50 == container.TRAILING)
container.constrain(button.BOTTOM + 50 < container.BOTTOM)
```

In this case, we've defined 4 constraints:

- The top of the button is 50 pixels lower than the top of the container
- The leading edge of the button is 50 pixels further to the right than the leading edge of the container. The "leading" edge is a localization- sensitive way of saying "left" or "right" - in a left-to-right language like English, the leading edge is the left hand side; in a right-to-left language like Hebrew or Arabic, the leading edge is the right hand side. If you really want to use the left or right edge, regardless of language direction, the identifiers `LEFT` and `RIGHT` can be used.
- The trailing edge of the container is 50 pixels further to the right than the trailing end of the button. The "trailing" edge is the right hand side in a right-to-left language.
- The bottom of the container must be more than 50 pixel further down than the bottom of the button.

This set of constraints is enough to uniquely place the button - but it also describes how the button will change size as the window changes size. As the window gets wider, the button will get wider to ensure that constraints 2 and 3

are satisfied. However, the vertical position of the button won't change as the window gets taller; the fourth constraint ensures that any extra height will go into the space below the button.

Note: The Cassowary Algorithm

This approach to GUI layout has a strong mathematical basis - it's based on an algorithm called Cassowary. It's also the basis of the widget auto-layouts tools introduced in OS X 10.7 and iOS 6.

Now we've set up the container, we can set the container as the content of the main app window:

```
app.main_window.content = container
```

Lastly, we can start the main app loop. This is a blocking call; it won't return until you quit the main app:

```
app.main_loop()
```

And that's it! Save this script as `helloworld.py`, and you're ready to go.

3.2.1 Running the app

Before you run the app, you'll need to install toga. Although you *can* install toga by just running:

```
$ pip install toga
```

We strongly suggest that you **don't** do this. We'd suggest creating a virtual environment first, and installing toga in that virtual environment.

Note: Problems under Linux

Unfortunately, GTK+3 doesn't provide a pip-installable version of it's Python bindings, so if you're using a virtual environment with –no-site-packages installed (which is the default), GTK+ won't be in your `PYTHONPATH` inside a virtual environment.

To make the system GTK+ bindings available to your virtualenv, symlinking the *gi* module from the system dist-packages directory into your virtualenv's site-packages:

```
$ cd <your virtualenv dir>/lib/python2.7/site-packages
$ ln -si /usr/lib/python2.7/dist-packages/gi
```

Once you've got toga installed, you can run your script:

```
$ python helloworld.py
```

This should pop up a window with a button. If you click on the button, you should see messages appear in the console. Even though we didn't define anything about menus, the app will have default menu entries to quit the app, and an About page. The keyboard bindings to quit the app, plus the "close" button on the window will also work as expected. The app will have a default Toga icon (a picture of Tiberius the yak).

3.3 Core Widgets

App

Container

Button

TextInput

PasswordInput

3.4 Desktop widgets

Window

SplitPanel

3.5 Mobile Widgets

Screen

3.6 Contributing to Toga

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

3.6.1 Setting up your development environment

The recommended way of setting up your development envrionment for Toga 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/toga.git
$ cd toga
$ mkvirtualenv toga
```

Toga 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 Toga, you have to run the following commands within your virutal envrionment:

```
$ pip install -e .
$ pip install -r requirements_dev.txt
```

In case you are running a python version `< 2.7` please use the `requirements_dev.py26.txt` instead because `unittest2` is not part of the standard library for these version.

Now you are ready to start hacking! Have fun!

3.7 Toga Roadmap

Toga 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.

3.8 Widgets

The core of Toga is it's widget set. Modern GUI apps have lots of native controls that need to be represented. The following widgets have no representation at present, and need to be added.

There's also the task of porting widgets available on one platform to another platform.

3.8.1 Input

Inputs are mechanisms for displaying and editing input provided by the user.

- ComboBox - A free entry TextField that provides options (e.g., text with past choices)
  - Cocoa: NSComboBox
  - GTK+: Gtk.ComboBox.new_with_model_and_entry
  - iOS: ?
- **Switch - A native control for enabled/disabled**
  - Cocoa: NSBUtton with checkbox style
  - GTK+: Gtk.CheckButton (maybe Gtk.Switch?)
  - iOS: UISwitch
- **DateInput - A widget for selecting a date**
  - Cocoa: NSDatePicker, constrained to DMY
  - GTK+: Gtk.Calendar?
  - iOS: UIDatePicker
- **TimeInput - A widget for selecting a time**
  - Cocoa: NSDatePicker, Constrained to Time
  - GTK+: ?
  - iOS: UIDatePicker
- **DateTimeInput - A widget for selecting a date and a time.**
  - Cocoa: NSDatePicker
  - GTK+: Gtk.Calendar + ?
  - iOS: UIDatePicker
- MultilineTextInput - A widget for displaying multiline text, optionally editable.
  - Cocoa: NSTextView inside an NSScrollView
  - GTK+: Gtk.TextView? (is there a simpler version than a full text editor?)
  - iOS: UITextView
- Selection - A button that allows the user to choose from one of a fixed number of options
  - Cocoa: NSPopupButton, with NSMenu for options.
  - GTK+: Gtk.ComboBox.new_with_model
  - iOS: UIPickerView

- **ColorInput - A widget for selecting a color**
  - Cocoa: NSColorWell
  - GTK+: Gtk.ColorButton
  - iOS: ?
- **SliderInput (H & V) - A widget for selecting a value from a range.**
  - Cocoa: NSSlider
  - GTK+: Gtk.Scale
  - iOS: UISlider
- NumberInput - A widget to allow entry of a numerical value, possibly with helper buttons to make it easy to increase/decrease the value.
  - Cocoa: NSTextField with NSStepper
  - GTK+: GTKSpinButton
  - iOS: UITextField with UIStepper
- **Table: A scrollable display of columns of tabular data**
  - Cocoa: Done
  - GTK+: Gtk.TreeView with a Gtk.ListStore
  - iOS: UITableView
- **Tree: A scrollable display of heirarchical data**
  - Cocoa: Done
  - GTK+: Gtk.TreeView with a Gtk.TreeStore
  - iOS: UITableView with navigation
- **SearchInput - A variant of TextField that is decorated as a search box.**
  - Cocoa: NSSearchField
  - GTK+: ?
  - iOS: UISearchBar?

3.8.2 Views

Views are mechanisms for displaying rich content, usually in a readonly manner.

- **Separator - a visual separator; usually a faint line.**
  - Cocoa: NSSeparator
  - GTK+:
  - iOS:
- ProgressBar - A horizontal bar that displays progress, either progress against a known value, or indeterminiate
  - Cocoa: NSProgressIndicator, Bar style
  - GTK+: Gtk.ProgressBar
  - iOS: UIProgressView

- **ActivityIndicator - A spinner widget showing that something is happening**
  - Cocoa: NSProgressIndicator, Spinning style
  - GTK+: Gtk.Spinner
  - iOS: UIActivityIndicatorView
- **ImageView - Display an graphical image**
  - Cocoa: NSImageView
  - GTK+: Gtk.Image
  - iOS: UIImageView
- **VideoView - Display a video**
  - Cocoa: AVPlayerView
  - GTK+: Custom Integrate with GStreamer
  - iOS: MPMoviePlayerController
- **WebView - Display a web page. Just the web page; no URL chrome, etc.**
  - Cocoa: WebView
  - GTK+: Webkit.WebView (via WebkitGtk)
  - iOS: UIWebView
- **PDFView - Display a PDF document**
  - Cocoa: PDFView
  - GTK+: ?
  - iOS: ? Integration with QuickLook?
- **MapView - Display a map**
  - Cocoa: MKMapView
  - GTK+: Probably a Webkit.WebView pointing at Google Maps/OpenStreetMap.org
  - iOS: MKMapView

3.8.3 Container widgets

Containers are widgets that can contain other widgets.

- **Box - A box drawn around a collection of widgets; often has a label**
  - Cocoa: NSBox
  - GTK+:
  - iOS:
- **ButtonContainer - A layout for a group of radio/checkbox options**
  - Cocoa: NSMatrix, or NSView with pre-set constraints.
  - GTK+: ListBox?
  - iOS:
- **ScrollContainer - A container whose internal content can be scrolled.**

  - Cocoa: Done
  - GTK+:
  - iOS: UIScrollView?
- **SplitContainer - An adjustable separator bar between 2+ visible panes of content**
  - Cocoa: Done
  - GTK+:
  - iOS:
- **FormContainer - A layout for a "key/value" or "label/widget" form**
  - Cocoa: NSForm, or NSView with pre-set constraints.
  - GTK+:
  - iOS:
- OptionContainer - (suggestions for better name welcome) A container view that holds a small, fixed number of subviews, only one of which is visible at any given time. Generally rendered as something with "lozenge" style buttons over a box. Examples of use: OS X System preference panes that contain multiple options (e.g., Keyboard settings have an option layout for "Keyboard", "Text", "Shortcuts" and "Input sources")
  - Cocoa: NSTabView
  - GTK+: GtkNotebook (Maybe GtkStack on 3.10+?)
  - iOS: ?
- SectionContainer - (suggestions for better name welcome) A container view that holds a small number of subviews, only one of which is visible at any given time. Each "section" has a name and icon. Examples of use: top level navigation in Safari's preferences panel.
  - Cocoa: NSTabView
  - GTK+: ?
  - iOS: ?
- TabContainer - A container view for holding an unknown number of subviews, each of which is of the same type - e.g., web browser tabs.
  - Cocoa: ?
  - GTK+: GtkNotebook
  - iOS: ?
- NavigationContainer - A container view that holds a navigable tree of subviews; essentially a view that has a "back" button to return to the previous view in a heirarchy. Example of use: Top level navigation in the OS X System Preferences panel.
  - Cocoa: No native control
  - GTK+: No native control; Gtk.HeaderBar in 3.10+
  - iOS: UINavigationBar + NavigationController

3.8.4 Dialogs and windows

GUIs aren't all about widgets - sometimes you need to pop up a dialog to query the user.

- **Info - a modal dialog providing an "OK" option**

  - Cocoa: NSAlert
  - GTK+: Gtk.MessageDialog, type Gtk.MessageType.INFO, buttons Gtk.ButtonsType.OK
  - iOS:
- **Error - a modal dialog showing an error, and a continue option.**
  - Cocoa: NSAlert
  - GTK+: Gtk.MessageDialog, type Gtk.MessageType.ERROR, buttons Gtk.ButtonsType.CANCEL
  - iOS:
- **Question - a modal dialog that asks a Yes/No question**
  - Cocoa: NSAlert with pre-canned buttons
  - GTK+: Gtk.MessageDialog, type Gtk.MessageType.QUESTION, buttons Gtk.ButtonsType.YES_NO
  - iOS:
- **Confirm - a modal dialog confirming "OK" or "cancel"**
  - Cocoa: NSAlert with pre-canned buttons, "proceed" name
  - GTK+: Gtk.MessageDialog, type Gtk.MessageType.WARNING, buttons Gtk.ButtonsType.OK_CANCEL
  - iOS:
- **StackTrace - a modal dialog for displaying a long stack trace.**
  - Cocoa: Custom NSWindow
  - GTK+: Custom Gtk.Dialog
  - iOS:
- **File Open - a mechanism for finding and specifying a file on disk.**
  - Cocoa:
  - GTK+: Gtk.FileChooserDialog
  - iOS:
- **File Save - a mechanism for finding and specifying a filename to save to.**
  - Cocoa:
  - GTK+:
  - iOS:

3.8.5 Miscellaneous

One of the aims of Toga is to provide a rich, feature-driven approach to app development. This requires the development of APIs to support rich features.

- Long running tasks - GUI toolkits have a common pattern of needing to periodically update a GUI based on some long running background task. They usually accomplish this with some sort of timer-based API to ensure that the main event loop keeps running. Python has a "yield" keyword that can be prepurposed for this.
- Toolbar - support for adding a toolbar to an app definition. Interpretation in mobile will be difficult; maybe some sort of top level action menu available via a slideout tray (e.g., GMail account selection tray)

- Preferences - support for saving app preferences, and visualizing them in a platform native way.
- Easy handling of long running tasks - possibly using generators to yield control back to the event loop.
- Notification when updates are available
- Easy Licening/registration of apps. Monetization is not a bad thing, and shouldn't be mutually exclusive with open source.

3.9 Platforms

Toga currently has good support for Cocoa on OS X, GTK+, and iOS. Proof-of-concept support exists for Windows Win32. Support for a more modern Windows API would be desirable.

In the mobile space, it would be great if Toga supported Android, Windows Phone, or any other phone platform.

3.10 Release History

3.10.1 0.1.0 - In development

Initial public release.

3.11 Cocoa

The Toga Cocoa bindings uses

3.12 GTK+

The Toga GTK+ backend uses the native GTK+ bindings for Python.

3.13 Win32

3.14 iOS

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*