Foundations of PyGTK Development

GUI Creation with Python

—

Second Edition

—

W. David Ashley
Andrew Krause

Apress®

Foundations of PyGTK Development

GUI Creation with Python

Second Edition

W. David Ashley
Andrew Krause

Apress®

Foundations of PyGTK Development: GUI Creation with Python

W. David Ashley
AUSTIN, TX, USA

Andrew Krause
Leesburg, VA, USA

ISBN-13 (pbk): 978-1-4842-4178-3
ISBN-13 (electronic): 978-1-4842-4179-0
https://doi.org/10.1007/978-1-4842-4179-0

Library of Congress Control Number: 2018966864

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4178-3. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

I dedicate this book to my wife.
Without you, all of this would not be possible.
—W. David Ashley

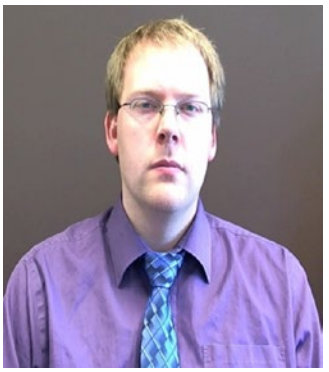
Table of Contents

About the Author

W. David Ashley is a technical writer for SkillSoft, where he specializes in open source, particularly Linux. As a member of the Linux Fedora documentation team, he recently led the Libvert project documentation and wrote the Python programs included with it. He has developed in 20 different programming languages during his 30 years as a software developer and IT consultant. This includes more than 18 years at IBM and 12 years with American Airlines.

About the Technical Reviewers

Jonathan Giszczak is a professional software developer with extensive experience writing software for the military and financial services industries, as well as the game industry. He graduated from the University of Michigan with a degree in computer engineering. He has been writing C, C++, and Python applications since the 1990s, including applications in Motif and PyGTK.



Peter Gill loves spending time with his family in Newfoundland, Canada. He is currently a software developer at TownSuite, where he specializes in release deployment and leading a full stack web development team. Peter loves learning programming language and has used Python, Ruby, Rust, Io, Prolog, Java, C, C++, C#, VB, JavaScript, Typescript, Bash, PowerShell and is currently focused on C# with ASP .NET Core and Typescript. He is a huge advocate open source software. He loves to use Git, Jenkins, Docker, and other tools related to automated deployments.

Acknowledgments

I would like to express my gratitude to the many people who have made this book possible. Many thanks go to Daniel Berrange of Red Hat, whose assistance has certainly decreased the number of errors in the book. I would also like to thank Peter Gill and Jonathan Giszczak for their fine technical reviewing skills. You were very tough on every paragraph I wrote and every example I coded, but this book is better today because of the hard work you put into the project.

I would like to extend a special thanks to Andrew Krause for his encouragement and help. Without him, this update to his original book would not have been possible.

In addition, I would like to thank the people at Apress who put so many hours of hard work into the book. I could not imagine writing for any other publisher. It is a great organization that makes the writing process enjoyable.

Finally, I need to acknowledge my wife, who has supported me in every step of the process. Without you, I would not be who I am today and for that I am forever grateful.

—W. David Ashley

Introduction

One of the most important aspects of an application is the interface that is provided to interact with the user. With the unprecedented popularity of computers in society today, people have come to expect those user interfaces to be graphical, and the question of which graphical toolkit to use quickly arises for any developer. For many, the cross-platform, feature-rich GTK+ library is the obvious choice.

Learning GTK+ can be a daunting task, because many features lack documentation and others are difficult to understand even with the API documentation. *Foundations of PyGTK Development* aims to decrease the learning curve and set you on your way to creating cross-platform graphical user interfaces for your applications.

Each chapter in this book contains multiple examples that help you further your understanding. In addition to these examples, the final chapter of this book provides five complete applications that incorporate topics from the previous chapters. These applications show you how to bring together what you have learned to accomplish in various projects.

Each chapter starts with an overview, so that you are able to skip around if you want. Most chapters also contain exercises to test your understanding of the material. I recommend that you complete all the exercises before continuing, because the best way to learn GTK+ is to use it.

At the end of this book, there are multiple appendixes that serve as references for various aspects of GTK+. These appendixes include tables listing signals, styles, and properties for every widget in GTK+. These appendixes will remain a useful reference after you have finished reading the book and begin creating your own applications. In addition, Appendix D explains the solutions to all the exercises in the book.

Who Should Read This Book

Because this book begins with the basics and works up to more difficult concepts, you do not need any previous knowledge of GTK+ development to use this book. This book does assume that you have a decent grasp of the Python programming language. You should

also be comfortable with running commands and terminating applications (Ctrl+C) in a Linux terminal.

In addition to a grasp of the Python programming language, some parts of this book may be difficult to understand without some further knowledge about programming for Linux in general. You will get more out of this book if you already comprehend basic object-oriented concepts. It is also helpful to know how Linux handles processes.

You can still use this book if you do not already know how to implement object orientation or manage processes in Linux, but you may need to supplement this book with one or more online resources. A list of helpful links and tutorials can be found on the book's web site, which is located at `www.gtkbook.com`. You can also find more information about the book at `www.apress.com`.

How This Book Is Organized

Foundations of PyGTK Development is composed of 14 chapters. Each chapter gives you a broad understanding of its topic. For example, Chapter 4 covers container widgets and introduces many of the most important widgets derived from the `Gtk.Container` class.

Because of this structure, some chapters are somewhat lengthy. Do not feel as though you have to complete a whole chapter in one sitting, because it can be difficult to remember all the information presented. Also, because many examples span multiple pages, consider focusing on just a few examples at a time; try to understand their syntax and intent.

Each chapter provides important information and unique perspectives that help you to become a proficient PyGTK developer.

Chapter 1 teaches you how to install the GTK+ libraries and their dependencies on your Linux system. It also gives an overview of each of the GTK+ libraries, including GObject, GDK, GdkPixbuf, Pango, and ATK.

Chapter 2 introduces the `Gtk.Application` and `Gtk.ApplicationWindow` classes. These classes are fundamental classes that wrap the program logic and provide some useful features for your application. While a GTK+ program can be written without utilizing these classes, you will find the creation process much easier and more object-oriented if you utilize these classes.

Chapter 3 steps through two Hello World applications. The first shows you the basic essentials that are required by every GTK+ application. The second expands on the first

while also covering signals, callback functions, events, and child widgets. You then learn about widget properties and the Gtk.Button and Gtk.Label widgets.

Chapter 4 begins by introducing the `Gtk.Container` class. Next, it teaches you about horizontal and vertical boxes, grids, fixed containers, horizontal and vertical panes, notebooks, and event boxes.

Chapter 5 covers basic widgets that provide a way for you to interact with users. These include toggle buttons, specialized buttons, text entries, and spin buttons.

Chapter 6 introduces you to the vast array of built-in dialogs. It also teaches you how to create your own custom dialogs.

Chapter 7 is a general overview of the most useful features of Python. It covers many Python features that are directly useful to the GTK+ programmer but not necessarily covered in depth in many Python introductory texts.

Chapter 8 introduces you to scrolled windows. It also gives in-depth instructions on using the text view widget. Other topics include the clipboard and the Gtk.SourceView library.

Chapter 9 covers two types of widgets that use the `Gtk.TreeModel` object. It gives an in-depth overview of the tree view widget and shows you how to use combo boxes with tree models or strings.

Chapter 10 provides two methods of menu creation: manual and dynamic. It covers menus, toolbars, pop-up menus, keyboard accelerators, and the status bar widget.

Chapter 11 is a short chapter about how to design user interfaces with the Glade user interface builder. It also shows you how to dynamically load your user interfaces using `Gtk.Builder`.

Chapter 12 teaches you how to create your own custom GTK+ widgets by deriving them from other widgets.

Chapter 13 covers many of the remaining widgets that do not quite fit into other chapters. This includes several widgets that were introduced in GTK+ 2.10, including recent files and tray icon support.

Chapter 14 gives you a few longer, real-world examples. They take the concepts you have learned throughout the book and show you how they can be used together.

The appendixes act as references to widget properties, signals, styles, stock items, and descriptions of exercise solutions.

Official Web Site

You can find additional resources on the book's official web site, found at www.gtkbook.com. This web site includes up-to-date documentation, links to useful resources, and articles that supplement what you learn in this book. There is also find a link to the downloadable source code for every example in this book. The Apress web site (www.apress.com,) is another great place to find more information about this book.

When you unzip the source code from the web site, you will find a folder that contains the examples in each chapter and an additional folder that holds exercise solutions. You can run all the files within the current folder.

CHAPTER 1

Getting Started

Welcome to *Foundations of PyGTK Development*. In this book, you acquire a thorough knowledge of the GIMP Toolkit (GTK+), which allows you to create comprehensive graphical programs. Before continuing, you should be aware that this book is aimed at Python programmers, so we assume that you already have a good understanding of the Python language, and you can jump right into using GTK+. Time is not spent on bringing you up to speed on Python.

To get the most out of this book, you should follow each chapter sequentially and study all the examples in each chapter. Getting started with GTK+ on Linux is very easy because most distributions are bundled with everything you need to create and run Python/GTK+ programs. We cover Windows and macOS installation procedures later in this chapter.

There are a few tools that should be installed to get you started without running into trouble. First, Python 3.x should be installed. It is required to run GTK+ 3.x Python programs. Second, the GTK+ 3.x runtime libraries should be installed. These libraries come with many dependencies installed, including GObject, Pango, GLib, GDK, GdkPixbuf, and ATK. Be sure to install all the dependent libraries.

You do *not* need to install the GNU Compiler Collection. You are not compiling any C/C++ programs in the examples provided in this book. You only need Python 3.x and the GTK+ 3.x runtime libraries to be installed to run the example programs.

Differences Between GTK+ 2.x and 3.x

If you are proficient in GTK+ 2.x, you may be surprised by the changes in version 3.x. There are both small and large changes to the GTK+ API and the Python classes that wrap those libraries. While the basics for most widgets are unchanged, there are a lot of small "gotchas" that can cause you grief until you understand why and where the changes have been made.

W. D. Ashley and A. Krause, *Foundations of PyGTK Development*,
https://doi.org/10.1007/978-1-4842-4179-0_1

The reason for most of these changes is due to a change in the GTK+ philosophy. The GTK+ 2.x libraries were designed around consistency between all GTK+ programs, with the use of GTK+ themes as the basis for that consistency. This philosophy completely changed with the GTK+ libraries. While themes are still available, it is now easier to create GTK+ programs that have their own look and feel separate from the current GTK theme. While this gives the developer greater control, it also requires some extra programming steps to achieve the look and feel. It also removes some APIs that make a widget easy to create and control.

The following is a partial list of the differences between GTK+ 2.x and 3.x. Some of these items have simple workarounds, but others require a little more work on the programmer's part because they are different enough to cause source code porting problems.

- Many standard stock icons have been removed, mostly the ones used on push buttons and menu items. If you need these icons, you must provide your own set.
- All the 2.x constants are now grouped in a 3.x Python class as attributes. If you are porting source code, this is a major area that needs to be addressed.
- Some containers have been eliminated. For instance, the `Gtk.Hbox` and `Gtk.Vbox` widgets have been removed and you now must specify the orientation of a `Gtk.Box` via a parameter when creating a new `Gtk.Box` instance. Note that the `Gtk.Box` class is now a real class in GTK+ 3.x, not an abstract class.
- Default packing for containers has been removed; all packing parameters must be supplied to the API.
- Some standard dialogs have been removed. You must create your own dialogs to replace them.
- There are two new major classes that are very useful for the overall control of large and small applications: the `Gtk.Application` class and the `Gtk.ApplicationWindow` class. While these classes are not strictly needed for simple applications, you still find them useful for even the simplest of applications. For that reason, we base all the examples in this book on these two classes to wrap our widget examples.

Creating menus is much easier using the `Gtk.Application` and `Gtk.ApplicationWindow` classes. This required complex programming in the GTK+ 2.x environment and is reduced to creating an XML file to represent the menu you want to create in the 3.x environment.

Installing GTK+ 3.x

Before you can create programs, you must install Python, GTK+, and all the dependent libraries. This section covers installing GTK+ on Linux and other Unix-like operating systems. It does not cover how to install GTK+ on macOS or Windows. You need to research the correct way to install GTK+ and Python in those OS environments.

Most modern Linux distributions include Python and GTK+ as part of their respective repositories. You simply need to select Python 3 (this is sometimes installed by default) and GTK+ 3.x (use the latest version available, as shown in Figure 1-1) from the package install program in your Linux distribution and then install those packages along with all the dependent packages.

To test your installation, run the following command.

```
/usr/bin/gtk3-demo
```

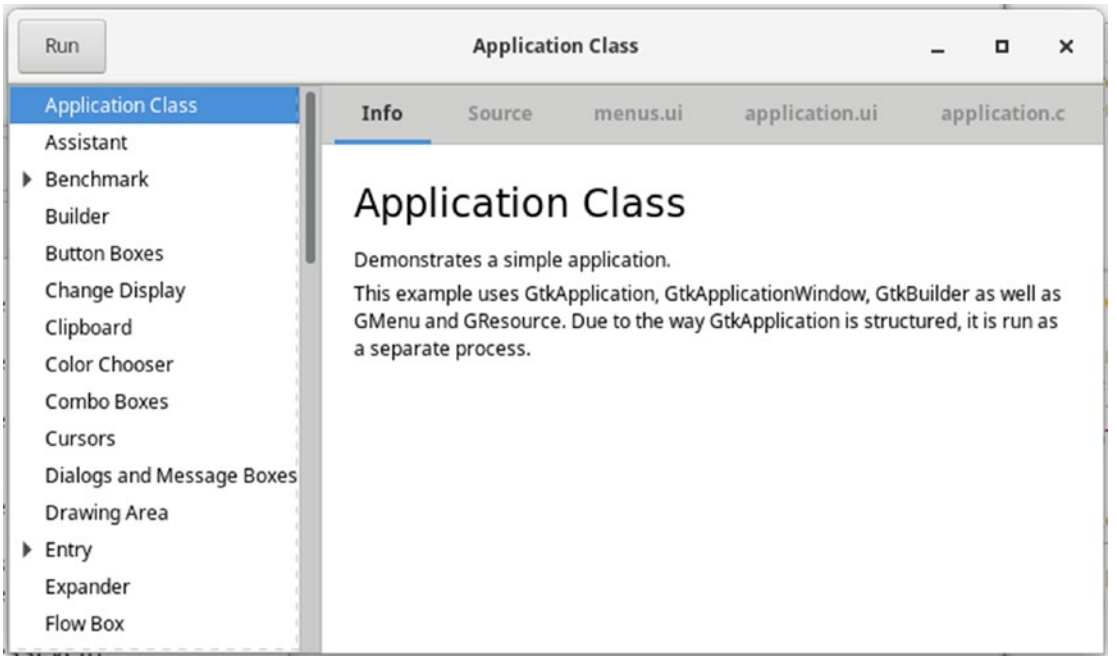


Figure 1-1. *GTK+ 3 demo program*

If the program exists and the widget documentation window appears, then the GTK+ installation was successful.

Summary

This chapter introduced GTK+ Version 3.x and Python 3 along with some installation prerequisites. It presented some post-installation tests to ensure that GTK+ was successfully installed. And it discussed some differences between GTK+ 2.x and 3.x.

After successfully installing GTK+ 3.x and Python 3, your environment should be ready to build your first Python/GTK+ program.

Chapter 2 further discusses `Gtk.Application` and the `Gtk.ApplicationWindow`, the base classes that you should use for all Python 3 GTK+ 3.x programs.

CHAPTER 2

The Application and ApplicationWindow Classes

A new set of classes were introduced in GTK+ 3.x: `Gtk.Application` and `Gtk.ApplicationWindow`. These classes are designed to be the base instances for your GUI application. They wrap the application and the main window behavior of your application. They have many built-in features and provide containers for the functions in your application. The `Gtk.Application` and `Gtk.ApplicationWindow` classes are described in detail in this chapter because they are the basis for all the example programs in this book.

The Gtk.Application Class

`Gtk.Application` is the base class of a GTK application. Its primary purpose is to separate your program from Python `__main__` function, which is a Python implementation detail. The philosophy of `Gtk.Application` is that applications are interested in being told what needs to happen and when it needs to happen in response to actions from the user. The exact mechanism by which Python starts applications is uninteresting.

`Gtk.Application` exposes a set of signals (or virtual methods) that an application should respond to.

- `startup`: Sets up the application when it first starts. The virtual method name for this signal is `do_startup`.
- `shutdown`: Performs shutdown tasks. The virtual method name for this signal is `do_shutdown`.

W. D. Ashley and A. Krause, *Foundations of PyGTK Development*,
https://doi.org/10.1007/978-1-4842-4179-0_2

- `activate`: Shows the default first window of the application (like a new document). The virtual method name for this signal is `do_activate`.
- `open`: Opens files and shows them in a new window. This corresponds to someone trying to open a document (or documents) using the application from the file browser, or similar. The virtual method name for this signal is `do_open`.

When your application starts, the `startup` signal is fired. This gives you a chance to perform initialization tasks that are not directly related to showing a new window. After this, depending on how the application is started, either `activate` or `open` signal is called next.

The signal name and the receiving method name should not be the same. The receiving method name should have an `on_` prefix. For instance, a signal named `paste` should have a `connect` call that looks something like the following.

```
action = Gio.SimpleAction.new("paste", None)
action.connect("activate", self.on_paste)
self.add_action(action)
```

Note that you have to specify the new signal name and the corresponding method name. By convention in GTK+ 3.x, signals that are built into an existing class have a `do_` prefix for their corresponding method names. Callbacks should have method names with an `on_` prefix. Adding a prefix to the method name prevents inadvertently overriding method names that are not a part of the signal mechanism.

`Gtk.Application` defaults to applications being single-instance. If the user attempts to start a second instance of a single-instance application, then `Gtk.Application` signals the first instance, and you receive additional `activate` or `open` signals. In this case, the second instance exits immediately without calling `startup` or `shutdown` signals.

For this reason, you should do essentially no work at all from Python's `__main__` function. All startup initialization should be done in `Gtk.Application do_startup`. This avoids wasting work in the second-instance case where the program exits immediately.

The application continues to run as long as it needs to. This is usually as long as there are any open windows. You can also force the application to stay alive by using the `hold` method.

On shutdown, you receive a `shutdown` signal where you can do any necessary cleanup tasks (such as saving files to disk).

`Gtk.Application` does not implement `__main__` for you; you must do so yourself. Your `__main__` function should be as small as possible and do almost nothing except create your `Gtk.Application` and run it. The "real work" should always be done in response to the signals fired by `Gtk.Application`.

Primary vs. Local Instance

The *primary instance* of an application is the first instance that is run. A *remote instance* is an instance that has started but is not the primary instance. The term *local instance* is refers to the current process, which may or may not be the primary instance.

`Gtk.Application` only emits signals in the primary instance. Calls to `the Gtk.Application` API can be made in primary or remote instances (and are made from the vantage of being the local instance). When the local instance is the primary instance, method calls on `Gtk.Application` result in signals being emitted locally. When the local instance is a remote instance, method calls result in messages being sent to the primary instance and the signals are emitted there.

For example, calling the `do_activate` method on the primary instance emits the `activate` signal. Calling it on a remote instance results in a message being sent to the primary instance, and it emits the `activate` signal.

You rarely need to know if the local instance is primary or remote. In almost all cases, you should call the `Gtk.Application` method that you are interested in and have it forwarded or handled locally, as appropriate.

Actions

An application can register a set of actions that it supports in addition to the default `activate` and `open` actions. Actions are added to the application with the `GActionMap` interface, and invoked or queried with the `GActionGroup` interface.

As with the `activate` and `open` signals, calling `activate_action` on the primary instance activates the named action in the current process. Calling `activate_action` on a remote instance sends a message to the primary instance, causing the action to be activated there.

Dealing with the Command Line

Normally, `Gtk.Application` assumes that arguments passed on the command line are files to be opened. If no arguments are passed, then it assumes that an application is being launched to show its main window or an empty document. When files are given, you receive these files (in the form of `GFile`) from the open signal; otherwise, you receive an `activate` signal. It is recommended that new applications make use of this default handling of command-line arguments.

If you want to deal with command-line arguments in more advanced ways, there are several (complementary) mechanisms by which you can do this.

First, the `handle-local-options` signal is emitted, and the signal handler gets a dictionary with the parsed options. To make use of this, you need to register your options with the `add_main_option_entries` method. The signal handler can return a non-negative value to end the process with this exit code, or a negative value to continue with the regular handling of command-line options. A popular use of this signal is to implement a `--version` argument that works without communicating with a remote instance.

If `handle-local-options` is not flexible enough for your needs, you can override the `local_command_line` virtual function to entirely take over the handling of command-line arguments in the local instance. If you do so, you are responsible for registering the application and for handling a `--help` argument (the default `local_command_line` function does this for you).

It is also possible to invoke actions from `handle-local-options` or `local_command_line` in response to command-line arguments. For example, a mail client may choose to map the `--compose` command-line argument to an invocation of its `compose` action. This is done by calling `activate_action` from the `local_command_line` implementation. If the command line being processed is in the primary instance, then the `compose` action is invoked locally. If it is a remote instance, the action invocation is forwarded to the primary instance.

Note in particular that it is possible to use action activations instead of `activate` or `open`. It is perfectly reasonable that an application could start without an `activate` signal ever being emitted. `activate` is only supposed to be the default "started with no options" signal. Actions are meant to be used for anything else.

Some applications may wish to perform even more advanced handling of command lines, including controlling the life cycle of the remote instance and its exit status once it quits, as well as forwarding the entire contents of the command-line arguments, the environment, and forwarding `stdin/stdout/ stderr`. This can be accomplished using the `HANDLES_COMMAND_LINE` option and the `command-line` signal.

Example

Listing 2-1 provides a very simple example of an instance derived from the `Gtk.Application` class.

Listing 2-1. An Example of the Gtk.Application Class

```
class Application(Gtk.Application):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, application_id="org.example.myapp",
                         flags=Gio.ApplicationFlags.HANDLES_COMMAND_LINE,
                         **kwargs)
        self.window = None
        self.add_main_option("test", ord("t"), GLib.OptionFlags.NONE, GLib.
        OptionArg.NONE, "Command line test", None)

        def do_startup(self):
            Gtk.Application.do_startup(self)
            action = Gio.SimpleAction.new("quit", None)
            action.connect("activate", self.on_quit)
            self.add_action(action)

        def do_activate(self):
            # We only allow a single window and raise any existing ones
            if not self.window:
                # Windows are associated with the application
                # when the last one is closed the application shuts down
                self.window = AppWindow(application=self, title="Main Window")
            self.window.present()

        def do_command_line(self, command_line):
            options = command_line.get_options_dict()
            if options.contains("test"):
                # This is printed on the main instance
                print("Test argument received")
            self.activate()
            return 0
```