Foundations of GTK+ Development

Andrew Krause

Apress®

Foundations of GTK+ Development

ISBN-13 (pbk): 978-1-59059-793-4

ISBN-10 (pbk): 1-59059-793-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Jason Gilmore, Matt Wade
Technical Reviewers: Christiana Evelyn Johnson, Micah Carrick
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Jeff Pepper, Paul Sarknas, Dominic Shakeshaft, Jim Sumser, Matt Wade
Project Manager: Richard Dal Porto
Copy Edit Manager: Nicole Flores
Copy Editor: Heather Lang
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Pat Christenson
Proofreader: Elizabeth Berry
Indexer: Ann Rogers
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 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 `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

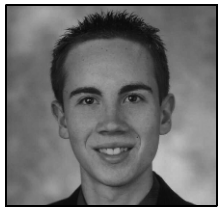
The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section or at the official book site, `http://www.gtkbook.com`.

I dedicate this book to Mrs. Kaminsky, for never allowing me to settle for anything but my best. I hope you can look at this book and see everything that you have done for me, even though I have yet to broaden the scope of my writing beyond technology.

Contents at a Glance

Contents

About the Author



ANDREW KRAUSE is the creator of OpenLDev, an integrated development environment that focuses on C, C++, and GTK+ projects. He is currently attending Pennsylvania State University with a major in computer engineering. Since 1998, Andrew has been developing with many computer and web programming languages, including C, C++, Perl, and PHP, as well as the graphical design libraries GTK+, Gtkmm, and Qt. He also designed flight hardware for the Low Ionosphere Measurement Satellite project at Penn State. More information about Andrew can be found at `www.andrewkrause.net`.

Acknowledgments

I would like to express my gratitude to the many people who have made this book possible. Many thanks go to Josh Hoy and Aaron Sebold, whose assistance has certainly decreased the number of errors in the book. I would also like to thank Christiana Johnson and Micah Carrick 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.

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. I would especially like to thank Matt Wade, Jason Gilmore, Richard Dal Porto, Heather Lang, and Katie Stence, who put up with all of my questions and provided quick help whenever it was needed.

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

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 even more are difficult to understand from only the API documentation. *Foundations of GTK+ 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 will 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 will show you how to bring together what you have learned to accomplish various projects.

The beginning of each chapter provides an overview of what that chapter will cover, 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 of the exercises before continuing, because the best way to learn GTK+ is to use it.

At the end of this book, you will find multiple appendixes that can serve as references for various aspects of GTK+. These appendixes include tables listing signals, styles, and properties for every widget in GTK+ and a complete list of stock items and `GError` types. These appendixes will remain a useful reference even after you have finished reading the book and begin creating your own applications. In addition, Appendix F contains explanations of the solutions to all of the exercises throughout 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 C 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 C 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 GTK+ Development is composed of 13 chapters. Each chapter will give you a broad understanding of its topic. For example, Chapter 3 covers container widgets and will introduce many of the most important widgets derived from the GtkContainer class.

Because of this structure, some chapters can be 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 of the information presented. Also, because many examples span multiple pages, consider focusing on just a few examples at a time and really trying to understand their syntax and intent.

Each chapter provides important information and unique perspectives that will help you to become a proficient GTK+ developer. They are as follows:

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 GLib, GObject, GDK, GdkPixbuf, Pango, and ATK.

Chapter 2 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 will then learn about widget properties and the GtkButton widget.

Chapter 3 begins by introducing the GtkContainer structure. Next, it teaches you about horizontal and vertical boxes, tables, fixed containers, horizontal and vertical panes, notebooks, and event boxes.

Chapter 4 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 5 introduces you to the vast array of built-in dialogs available to you. It also teaches you how to create your own custom dialogs.

Chapter 6 is a general overview of the most useful features in GLib. It covers many of the data types available to you. It also introduces idle functions, timeouts, spawning processes, loading dynamic modules, file utility functions, timers, and other general utility functions.

Chapter 7 introduces you to scrolled windows. It also gives in-depth instructions on using the text view widget. Other topics include the clipboard and the GtkSourceView library.

Chapter 8 covers two types of widgets that use the GtkTreeModel 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 9 provides two methods of menu creation: manual and dynamic. It covers menus, toolbars, pop-up menus, keyboard accelerators, and the status bar widget.

Chapter 10 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 Libglade.

Chapter 11 teaches you how to create your own custom GTK+ widgets by deriving them from other widgets or creating them from scratch. It also introduces you to implementing and using interfaces.

Chapter 12 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 13 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.

In addition to the chapters, six appendixes are provided as references to widget properties, signals, styles, stock items, `GError` types, and descriptions of exercise solutions.

Conventions

This book uses various typefaces to help you distinguish between GTK+ code and regular English phrases. Actual code is typeset in a `monospace` font. This can include whole lines of code or function names, signals, and properties in a paragraph.

There are other types of conventions used in this book, which follow.

Exercise 0-0. Sample Exercise

These boxes show exercises that test your understanding of the material in the section. They can include questions, code challenges, or various other types of material.

You should complete each of these exercises before proceeding, because they will help you practice the concepts you have learned throughout the current chapter and put them together with concepts from past chapters.

Note These boxes give important notes, tips, and cautions. It is essential that you pay attention to them, because they give you information that you will need when developing your own applications.

```
Textual output in the terminal is shown in a monospace font between these lines,
although most output will be in the form of an image, since GTK+ is graphical.
```

What You Need

Before proceeding, you will need a few things: a compiler, a text editor, a terminal emulator, the GTK+ libraries, the pkg-config application, and this book.

All compiler commands provided by this book are for the GCC compiler available at `http://gcc.gnu.org` or through your package manager. Most standard C or C++ compilers will work, but if you use a compiler other than GCC, you will have to use a different set of commands than those provided.

Any text editor will do, so you should choose the one that suits you best. Some popular text editors that you might consider include Vim, Emacs, Leafpad, and GEdit. Vim and Emacs are terminal-based editors, while Leafpad and GEdit are graphical text editors.

Instructions on installing the GTK+ libraries and the pkg-config application are provided in the last section of Chapter 1.

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 will supplement what you learn in this book. You can also find at this site a link to the downloadable source code for every example in this book. The Apress web site, found at `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 `make` to build all of the files within the current folder. It is also possible to make a single file by using the compile command given in Chapter 2 or by running `make sourcefile`. For example, to build `exercise2-1.c`, you should type make `exercise2-1`.

CHAPTER 1

Getting Started

Welcome to *Foundations of GTK+ Development*! In this book, you will acquire a comprehensive understanding of GIMP Toolkit (GTK+) that can help you to become a proficient graphical programmer. Before continuing, you should be aware that this book is aimed at C programmers, so we will jump right into using GTK+. Time will not be spent covering information you already know.

To get the most out of this book, you should follow along with each of the examples and try the exercises found at the end of most chapters. Getting started with GTK+ on Linux is quite simple, because the majority of modern distributions are typically bundled with the necessary libraries and tools.

Nevertheless, you need to make sure that you already have a few tools installed including the GNU Compiler Collection (GCC), the GTK+ 2.0 libraries, and the associated development packages. Later in this chapter, you will learn how to install these applications. If you do not have a compiler, you can still use this book, but you will get more out of it if you do the exercises. The best way to learn GTK+ is to use it!

Note The compiler of choice for this book is GCC, available for download at `http://gcc.gnu.org`. Any standard C or C++ compiler will work, but you will have to use a different set of commands than those provided. Alternative compiler commands will not be covered in this book.

At the end of most chapters, you will find one or two exercises that illustrate what you have learned up to that point. Make sure you complete each of the exercises before moving on to the next chapter, because they will help reaffirm your knowledge. Each chapter builds on concepts presented in previous chapters, so you will need a firm foundation in the basics to understand more complex examples.

In this chapter, you will learn the following:

- The history of GTK+ and the X Window System, which will provide you with some context regarding the tremendous impact these two technologies have had on developers
- What GTK+ and its supporting libraries provide to the graphical application developer
- What GTK+ language bindings are available and where to download them
- How to install GTK+ and its dependencies on your computer

A Brief History of GTK+

The GIMP Toolkit (GTK+) was originally designed for a raster graphics editor called the GNU Image Manipulation Program (GIMP). Three individuals, Peter Mattis, Spencer Kimball, and Josh MacDonald created GTK+ in 1997 while working in the eXperimental Computing Facility at the University of California, Berkeley.

Licensed under the Lesser General Public License (LGPL), GTK+ was adopted as the default graphical toolkit of GNOME and XFCE, two of the most popular Linux desktop environments. While it was originally used on the Linux operating system, GTK+ has since been expanded to support other UNIX-like operating systems: Microsoft Windows, BeOS, Solaris, Mac OS X, and others.

■ **Note** The LGPL is one of the things that distinguish GTK+ from other open source graphical toolkits. The LGPL is easier to use alongside proprietary software, unlike many other popular open source licenses. This makes the GNOME desktop environment, which utilizes GTK+, a popular choice throughout commercial industry.

GTK+ is currently in its second stable release cycle, GTK+ 2. The original branch, GTK+ 1, needed to be changed dramatically to include new features and its developers saw fit to break API compatibility.

Since the two branches of GTK+ are not compatible, they can be installed in parallel. You will need to make the distinction to the compiler that you want to use the second branch instead of the first when building an application, which you will learn how to do with GCC in the next chapter.

GTK+ 2 introduced a lot of new features including a font-rendering engine called Pango and a newly enhanced theme engine. Furthermore, improved accessibility support was implemented through the Accessibility Toolkit (ATK).

This book uses version 2 of GTK+ for all code examples. While GTK+ 2.10 has already been released, most of the examples should work with any version in the second branch. GTK+ 2 maintains backward compatibility, which means that any application that works for an earlier release of GTK+ 2 will work on later releases of version 2.

The X Window System

In 1984, Jim Gettys and Bob Scheifler created the X Window System (X11) at Massachusetts Institute of Technology as a platform-independent display environment for debugging the Argus system. Currently developed by The X.Org Foundation, X11 is the standard display manager on Linux and other UNIX-like operating systems. In the most basic terms, X11 provides windowing functionality for bitmap displays.

While the X Window System is used on Linux, many other operating systems such as Microsoft Windows do not use it. Therefore, another advantage of GTK+ is that it masks the need to interact with the underlying rendering system, regardless of what it is. Your code will look the same whether you are writing it for Linux, Windows, or Mac OS X.

Returning to Linux, X11 manages windows in their most basic and abstract form. It draws windows on the screen and handles their movements. X11 also controls input devices, such as mice and keyboards, in graphical environments.

X11's basic programming interface, Xlib, provides the tools necessary to create graphical user interfaces. Although developing with Xlib is possible, most programmers prefer to use a graphical toolkit such as GTK+, since all of the low-level calls are hidden and managed by the library's methods.

One of the major features that makes X11 unique among display managers is that it assumes the client and server are treated independently of each other. This allows the client to exist at a remote location independent of the server.

■ **Note** The definitions of client and server in the X Window System differ from their traditional ones. The client is the machine where the application is run. The server refers to the user's local display, rather than the remote machine.

Another advantage of the X Window System is that it does not strictly mandate user interfaces. This allows the graphical user interfaces (GUI) of window managers to be highly customizable. It is also why window managers can provide such differing interfaces and themes. This enables the freedom of choice Linux users enjoy today.

Ironically, this freedom is also one of the biggest criticisms of X11. Many people fear that it will encourage fragmentation within the community of Linux developers. But for now, we can continue to enjoy the ability to choose the window manager that best suits our own needs.

The GTK+ libraries were created so that you, as the programmer, do not need to interface with the X Window System directly. You can create windows and widgets, and you can handle interactions between those widgets and the user, but all direct rendering to the screen and Xlib function calls are handled automatically.

Therefore, this book will not cover the X Window System any further and will focus on the GTK+ libraries instead. You are welcome to find more information about X11 and the X.Org Foundation at `www.x.org`.

GTK+ and Supporting Libraries

GTK+ relies on multiple libraries, each providing the graphical application developer a specific class of functionality.

GTK+ is an object-oriented application programming interface (API) written in the C programming language. It is implemented with the concept of classes in mind to create an extensible system that builds upon itself. The object-oriented framework used was originally developed as a part of the GTK+ library itself, but has since been split from GTK+ and added to GLib as a separate supporting library called GObject. GObject enables full object-orientated development in C, including object inheritance, polymorphism, and, to the extent permissible in C, data hiding.

While making a great deal of functionality from the other libraries transparently available through its own API, the GTK+ library focuses only on providing the necessities of building

graphical user interfaces. The elements implemented in GTK+ itself include widgets such as buttons, labels, text boxes, and windows. It also provides more abstract components used for application layout and extended event capturing functionality. For example, Figure 1-1 is a screenshot of the GIMP application, which uses GTK+.

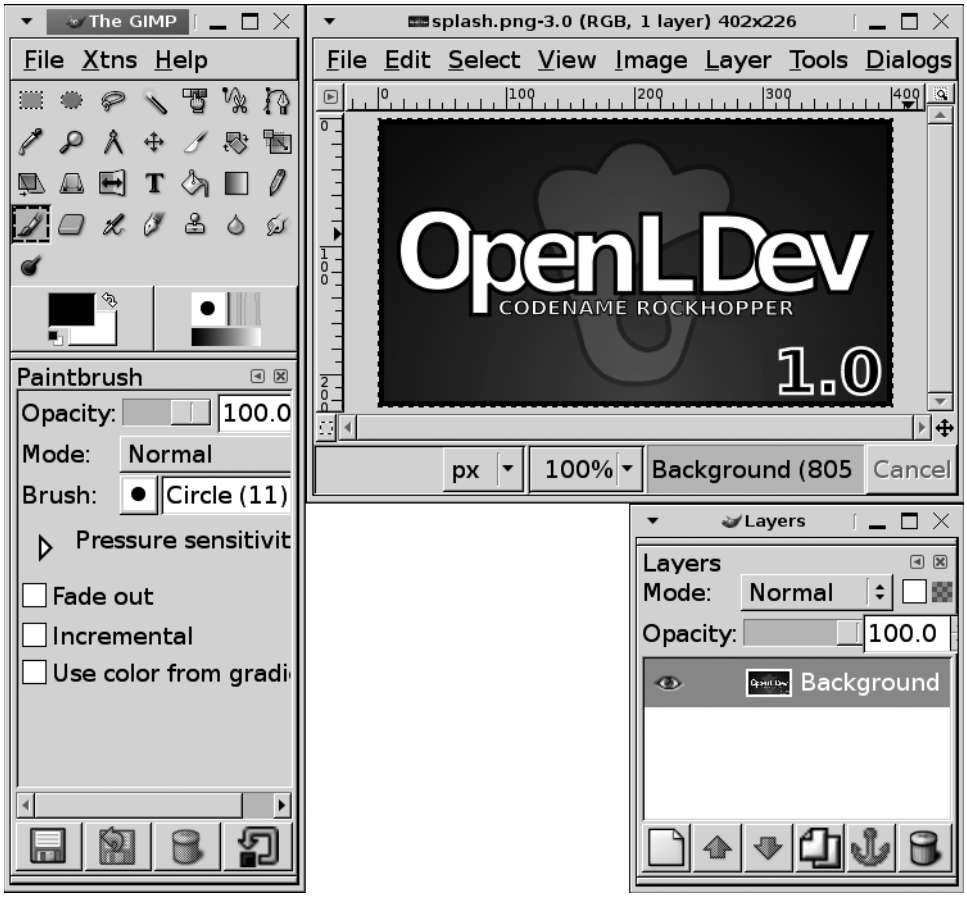


Figure 1-1. *The GIMP*

Other, less visible basics of GUI development, such as synchronous and asynchronous event processing, are supported mainly by other libraries. Yet, GTK+ does give access to many of them through its own API.

A 2-D vector graphics rendering library called Cairo has provided the rendering capabilities to GTK+ since the release of version 2.8. Cairo was created to render vector graphics consistently across all platforms and systems. It also allows the window manager to take advantage of hardware acceleration where available.

Cairo itself will not be covered in this book, with the exception of how it relates to GTK+'s printing API, since its calls lie underneath the layers of GTK+ that you will be interacting with. It is an important aspect you will want to explore if you later choose to hack the GTK+ source code. You can visit `www.cairographics.org` to find more information about Cairo.

GLib

GLib is a general-purpose utility library that is used to implement many useful nongraphical features. While it is required by GTK+, it can also be used independently. Because of this, some applications use GLib without the other GTK+ libraries for the many capabilities it provides.

One of the main benefits of using GLib is that it provides a cross-platform interface that allows your code to be run on any of its supported operating systems *with little to no rewriting of code*! Another advantageous aspect of GLib is the vast array of data types it provides to developers. A list of a few of the data types provided by GLib follows and will be covered in further detail in Chapter 6:

- GLib provides a number of data types to C programmers that are usually included by default in other languages, such as singly and doubly linked lists. Other basic data types include double-ended queues, self-balancing binary trees, and unbalanced n-ary trees.
- Hash tables allow you to create lists of pointers to data. They differ from linked lists, because, instead of accessing elements by an integer reference, you specify a second pointer as the key.
- Strings in GLib are similar to strings in C++, because they are text buffers that grow automatically as data is added. These are also easy to integrate with calls to the `printf()` function family.
- Memory slices are an efficient way to create chunks of memory that are all of the same size. They can be used to create arrays of evenly sized elements. This structure replaced memory chunks when it was introduced in the release of GLib 2.10.
- Caches allow you to share large, complex data structures in an easy API, which helps you to save space. These are used by GTK+ for styles and graphics contexts, since both of these objects consume a lot of resources.

GLib provides other data types, many of which will be introduced in Chapter 6. Furthermore, GLib implements other features besides data types. It also provides you with numerous types of utility functions. For instance, you'll find utility functions for file manipulation, internationalization support, strings, warnings, debugging flags, dynamic module loading, and automatic string completion, just to name a few.

In Chapter 6, you will also learn about idle functions, time-out functions, and timers—all of which open up a variety of interesting possibilities to developers. Idle functions allow you to call a function when the processor is not doing anything else for the application. Timeouts are used to call a function at a specified interval of time provided by you. A timer keeps track of how much time has passed since it was initiated. These could be used to check for updates when the application is idle, implement automatic save functionality, or track elapsed time, respectively.

Because of the cross-platform characteristics of GLib, it makes a convenient library to use for spawning processes, file manipulation, memory allocation, and threads. Any of these can be a nightmare when trying to develop for multiple platforms. GLib takes care of the hassles, so you do not have to worry about cross-platform compatibility issues.

GObject

The GLib Object System (GObject) was originally a part of the GTK+ 1 library in the form of the `GtkObject` class. With the release of GTK+ 2.0, it was moved into its own library, distributed along with GLib.

GObject is often criticized for its complexity, since its APIs can seem extremely drawn out. However, it was originally created to allow easy access to C objects from other programming languages. The ability to easily access C objects from other languages facilitates the large variety of bindings available for other programming languages, even though it is implemented in C.

This is so difficult because each programming language provides a different approach to data types, whether the differences appear on the surface or the internals of each language. For example, in C, you have data types including char, long, and integer. Other languages, such as Perl, do not have similar data types, since the type of each object is decided by how it is used. GObject gets around these limitations, the drawback being that deriving new objects is a convoluted process.

GObject also implements a fully featured object-oriented interface in C, which will be covered in detail throughout this section and the rest of this book. This system is the base for the GTK+ widget hierarchical structure as well as for many of the objects implemented in GTK+'s supporting libraries. GObject's object-oriented interface is implemented in part by a generic, dynamic type system called `GType`. `GType` allows programmers to implement many different dynamic data types through singly-inherited class structure. A singly-inherited class is an object hierarchy where each child class can only be derived directly from a single parent class. This will be discussed in more detail in Chapter 2, after you are introduced to GTK+ widgets.

Along with the ability to create extensible data types, GObject provides programmers with many nonclassed (or fundamental) data types. A nonclassed data type is a root class from which others are derived. It is important to note that the root class is not derived from any other classes itself.

Table 1-1 provides a list of the most important nonclassed data types. The `GType` macro, C variable descriptor, and a description is shown for each, along with its range if applicable.

Table 1-1. *Standard GObject Nonclassed Data Types*

GType	C Type	Description
`G_TYPE_NONE`		An empty type that is equivalent to `void`.
`G_TYPE_CHAR`	`gchar`	Equivalent to the standard C `char` type.
`G_TYPE_INT`	`gint`	Equivalent to the standard C `int` type. Values must be within the range of `G_MININT` to `G_MAXINT`.
`G_TYPE_LONG`	`glong`	Equivalent to the standard C `long` type. Values must be within the range of `G_MINLONG` to `G_MAXLONG`.
`G_TYPE_BOOLEAN`	`gboolean`	A standard Boolean type that holds either `TRUE` or `FALSE`.
`G_TYPE_ENUM`	`GEnumClass`	A standard enumeration equivalent to the C `enum` type.
`G_TYPE_FLAGS`	`GFlagsClass`	Bit fields holding Boolean flags.
`G_TYPE_FLOAT`	`gfloat`	Equivalent to the standard C `float` type. Values must be within the range of negative `G_MAXFLOAT` to `G_MAXFLOAT`.

GType	C Type	Description
G_TYPE_DOUBLE	gdouble	Equivalent to the standard C double type. Values must be within the range of negative G_MAXDOUBLE to G_MAXDOUBLE.
G_TYPE_STRING	gchar*	Equivalent to NULL-terminated C strings.
G_TYPE_POINTER	gpointer	An untyped pointer type similar to void*.

GObject provides GTK+ with two other vital data types: GValue and GObject. GValue is a generic container that can hold any structure of which the system is already aware. This allows functions to return a piece of data of an arbitrary type. Without GValue, the object-oriented nature of GTK+ would not be possible.

G_TYPE_GOBJECT, or GObject, is the fundamental type that the widget class inheritance structure of GTK+ is based on. It allows widgets to inherit the properties of their parents, including style properties and signals.

GObject is a singly-inherited system, where each child class can only have one parent class. The derived child inherits all characteristics of the parent, because in every way, the child *is* the parent. You will learn how to use this system to derive custom GTK+ widgets in Chapter 11.

GObject also provides widgets with a signal system, an object properties system, and memory management. We will explore all of these concepts in the next chapter.

GDK

The GIMP Drawing Kit (GDK) is a computer graphics library originally designed for the X Window System that wraps around low-level drawing and window functions. GDK acts as the intermediary between Xlib and GTK+.

It renders drawings, raster graphics, cursors, and fonts in all GTK+ applications. Also, since it is implemented in every GTK+ program, GDK provides drag-and-drop support and window events.

GDK provides GTK+ widgets the ability to be drawn to the screen. To do this, every widget has an associated GdkWindow object, except for a few widgets that will be discussed in a later chapter. A GdkWindow is essentially a rectangular area located on the screen in which the widget is drawn. GdkWindow objects also allow widgets to detect X Window System events, which will be covered in the next chapter.

GDK has been ported to Windows and Mac OS X. It has also included support for Cairo since the release of GTK+ 2.8.

GdkPixbuf

GdkPixbuf is a small library that provides client-side image manipulation functions. It was created as a replacement for the GNOME Imaging Model (Imlib). Images can be loaded from files or image data can be fed directly into the library functions. We will use this library when adding images to tree views and when creating new GtkImage widgets in later chapters.

One advantage of GdkPixbuf images is that images can be reference-counted. This means that a GdkPixbuf image can be displayed in multiple locations, while only being stored in memory once. It will only be destroyed when all reference counts are decremented.

The GdkPixbuf library takes advantage of Libart, a 2-D drawing library distributed with GNOME, to apply transformations to images. Because of this, you can shear, scale, and rotate images to your heart's delight. The images are then rendered using the GdkRGB library and drawable areas. By using such a wide variety of specialized tools, GdkPixbuf can provide image rendering of a very high class.

GdkPixbuf, while it is a small library, provides a wide variety of functions for manipulating and displaying images. The library will be put to only the most basic of uses throughout this book. For more information on advanced GdkPixbuf topics, you should reference its API documentation.

Pango

While GDK handles rendering images and windows, Pango controls text and font output in conjunction with Cairo or Xft, depending on your GTK+ version. It can also render directly to an in-memory buffer without the use of any secondary libraries.

■ **Note** Pango originated from the Greek word pan, which means "all," and the Japanese word go, which means "language." It was chosen because one of the design goals of Pango is to support all languages by creating a fully internationalized font-rendering system.

On Linux, Pango uses the FreeType and fontconfig libraries for client-side fonts. The thing that makes Pango stand out from the crowd is that it supports a vast array of languages. Virtually all of the world's major scripts are supported, which makes rendering internationalized text a nonissue in your applications.

All text within Pango is represented internally with UTF-8 encoding. UTF-8 is used because it is compatible with 8-bit software, which is prevalent on UNIX platforms. Offsets in UTF-8 are calculated based on characters, not bits, because each character can take up more than one byte. This will be important in Chapter 7 when you learn how to use the `GtkTextView` widget, because you will have to step by character offset, which may not always be one byte.

Pango supports a wide variety of text attributes. These include but are not limited to language, font family, style, weight, stretch, size, foreground color, background color, underline, strikethrough, rise, shape, and scale. Many of these attributes support multiple options themselves.

For convenience, the Pango Text Markup Language provides a simple set of tags that represent the text attributes in a form similar to HTML. With this markup language, you can easily change the font styles for arbitrary parts of text in a widget. This is especially useful when creating user interfaces with Glade User Interface Builder, because you can type tags directly into a widget's textual content field.

We will utilize Pango for many examples in later chapters when we need to change the font of a widget to something other than the user's default. Using the `PangoFontDescription` object or the Pango Text Markup Language can do this.

ATK

When designing an application, it is important to take into consideration the disabilities that some of your users may have. Therefore, the Accessibility Toolkit (ATK) provides all GTK+ widgets with a built-in method of handling accessibility issues.

Some examples of things ATK adds support for are screen readers and high-contrast visual themes for people who are visually impaired and keyboard behavior modifiers, such as sticky keys, for those with diminished motor control.

Although this is an important part of designing an application for production use, this book will not cover ATK. You need to learn how to use GTK+ widgets and how to create your own custom widgets before you can use ATK. Therefore, I will focus on GTK+ and other essentials for the remainder of this book.

It is important that you keep accessibility in the back of your mind and revisit the library when you are ready to deal with ATK in your own applications.

Language Bindings

GTK+, in its original form, can be used with the C programming language, but bindings have been created for many others. The most popular language bindings are in the following list, although a full list is available at `www.gtk.org/bindings.html`:

- Gtkmm is the official set of C++ bindings. You can use GTK+ with C++ because of backward compatibility, but Gtkmm provides all of the GTK+ features in a series of classes, the style of which will be familiar to all C++ programmers. The sources for Gtkmm, GLibmm, Libglademm, and other dependencies are available at `www.gtkmm.org`.

- PyGTK, available at `www.pygtk.org`, provides Python bindings for the GTK+ libraries. The advantage of using PyGTK is that it takes care of memory management and type casting for you. This alleviates problems that can plague programmers using other language bindings.

- Gtk2-perl, available at `http://gtk2-perl.sf.net`, provides all of the GTK+ libraries in an object-oriented Perl toolkit. Each of the libraries is split into modules called `Glib`, `Gtk2`, and `Gtk2::GladeXML`. Like most GTK+ bindings for scripting languages, memory management is handled by the language's facilities.

- PHP-GTK allows for handling PHP language bindings for GTK+. The PHP bindings allow you to create client-side cross-platform GUI applications. PHP-GTK is available at `http://gtk.php.net`. This topic is also covered in the Apress book *Pro PHP-GTK*, authored by Scott Mattocks (Berkeley, 2006).

- Java-Gnome, much like Gtkmm, provides a true object-oriented platform for the GTK+ libraries. Available at `http://java-gnome.sf.net`, it provides all of the essential libraries for developing GTK+ applications in Java.

- Gtk# provides GTK+ bindings for C# applications on a wide variety of operating systems. It is provided by the Mono Project at `www.mono-project.com`.

Installing GTK+

Before you can begin programming, you must install GTK+ and its dependencies on your system. This section covers installing GTK+ on Linux and other UNIX-like operating systems.

It is important to note, if you are using a Linux distribution with a package manager including Ubuntu, Debian, Fedora Core, or one of many others, you should install the precompiled binaries provided. You will need the GTK+ 2 libraries, pkg-config, and their dependencies.

The development packages of GTK+ and each of its dependencies are also required. In Debian and Debian-based distributions, these packages will end in `-dev`. In Fedora Core and other distributions that use the RedHat Package Manager (RPM), they will end in `-devel`. If you install the development package of GTK+, most modern package managers will take care of all of the necessary dependencies automatically. You should reference your Linux distribution's documentation for more information on installing distributed packages.

If you are going to install GTK+ and its dependencies from the source archives, the rest of this section is for you. GTK+ uses the standard GNU tools for compiling: autoconf is used for configuration and dealing with portability issues, automake for building makefiles, libtool for building shared libraries, and make for compiling and installing binaries.

The most recent GTK+ sources can be found at `www.gtk.org/download`. You will need to download the latest versions of ATK, GLib, GTK+, and Pango. You will also need Cairo, JPEG, libpng, pkg-config, and tiff from the dependencies directory.

If you are using an older version of Linux, you will need to install libiconv. Most systems already have this package, so it is safe to continue without it and install the library in the future if you run into any problems. You may also need to install libintl, fontconfig, and FreeType, although these are packages provided as standard on most modern Linux distributions.

You should also note that these packages *must* be installed in a precise order for the following procedure to work. After installing all of the packages from the dependencies directory on the GTK+ FTP site, you will need to install GLib, Pango, ATK, and GTK+ in that specific order.

The following procedure should be used on each source package, one at a time. Each library must be successfully installed before continuing on to the next, or the procedure will not work.

You are now ready to install GTK+, so let's begin. Once you have downloaded a package from the GTK+ FTP site, you can use one of the following commands to extract the file, depending on the type of archive you downloaded.

```
tar –xvzf package-name.tar.gz
tar –xvjf package-name.tar.bz2
```

By moving into the directory of the extracted archive, you will see a shell script called `configure`. This script will recursively parse through each of the directories in the source distribution and create template makefiles that are customized for your operating system. Each template file will be named `Makefile.in`. The following is a sample configure command that you can use:

```
./configure --prefix=/usr
```