

# Foundations of GTK Development

Foundations of GTK Development Korean  
Edition



FoundationsofGTKDevelopment:AbouttheAuthor	1
FoundationsofGTKDevelopment:Acknowledgments	1
FoundationsofGTKDevelopment:Introduction	2
FoundationsofGTKDevelopment:Chapter 01	5
FoundationsofGTKDevelopment:Chapter 02	3
FoundationsofGTKDevelopment:Chapter 03	7
FoundationsofGTKDevelopment:Chapter 04	6
FoundationsofGTKDevelopment:Chapter 05	9
FoundationsofGTKDevelopment:Chapter 06	10
FoundationsofGTKDevelopment:Chapter 07	3
FoundationsofGTKDevelopment:Chapter 08	8
FoundationsofGTKDevelopment:Chapter 08	8
FoundationsofGTKDevelopment:Chapter 09	4
FoundationsofGTKDevelopment:Chapter 09	1
FoundationsofGTKDevelopment:Chapter 10	5
FoundationsofGTKDevelopment:Chapter 10	2
FoundationsofGTKDevelopment:Chapter 11	7
FoundationsofGTKDevelopment:Chapter 12	6
FoundationsofGTKDevelopment:Chapter 13	9
FoundationsofGTKDevelopment:Chapter 13	3
FoundationsofGTKDevelopment:Appendix A	3
FoundationsofGTKDevelopment:Appendix B	3
FoundationsofGTKDevelopment:Appendix C	8
FoundationsofGTKDevelopment:Appendix C	5
FoundationsofGTKDevelopment:Appendix D	3
FoundationsofGTKDevelopment:Appendix D	0
FoundationsofGTKDevelopment:Appendix E	1
FoundationsofGTKDevelopment:Appendix E	1
FoundationsofGTKDevelopment:Appendix F	0



문서 출처 및 기여자	3
그림 출처, 라이선스와 기여자	1
	2
	0

# FoundationsofGTKDevelopment:AbouttheAuthor

---

저자 이력

저자 이력



**ANDREW KRAUSE** 는 C, C++, GTK+ 프로젝트에 중점을 둔 통합 개발 환경, OpenLDev의 개발자다. 현재 펜실베이니아 주립대학교에서 컴퓨터 공학을 전공하고 있다. 1998년부터 Andrew는 C, C++, Perl, PHP를 포함해 많은 컴퓨터 및 웹 프로그래밍 언어 뿐만 아니라 GTK+, Gtkmm, Qt와 같은 그래픽 디자인 라이브러리를 이용해 개발을 해오고 있다. 그는 또한 펜실베이니아주에서 저 전리층 측정 인공위성(Low Ionosphere Measurement Satellite) 프로젝트를 위한 항공기 하드웨어를 설계하였다. Andrew에 관한 정보는 <http://www.andrewkrause.net>에서 찾을 수 있다.

net에서 찾을 수 있다.

## Notes

# FoundationsofGTKDevelopment:Acknowledgments

---

감사의 말

감사의 말

이 책이 출판되기까지 도와주신 많은 분들께 감사를 표하고 싶다. 우선 이 책에 많은 오류를 포착하고 없애는데 도움을 주신 Josh Hoy와 Aaron Sebold에게 고마움을 전한다. 또 뛰어난 기술적 검토 기술을 겸비한 Christiana Johnson과 Micah Carrick에게도 감사함을 전하는 바이다. 필자가 집필한 단락마다, 그리고 필자가 코딩한 예제마다 엄격하게 살펴봐주었지만, 그들의 수고 덕분에 오늘 더 완성도가 높은 책을 출판하게 되었다.

또, 많은 시간 수고해 준 Apress의 직원분들께도 감사드린다. 다른 출판업체와 일하는 모습은 상상할 수조차 없다. 집필 과정을 재미있게 만들어준 멋진 단체다. 그 중에서도 나의 수많은 질문에 인내하고, 필요할 때마다 즉각 도와주신 Matt Wade, Jason Gilmore, Richard Dal Porto, Heather Lang, Katie Stence에게 감사의 마음을 표한다.

마지막으로 책을 쓰는 과정에서 끊임없이 나를 지원해준 내 가족에게 영광을 돌리고 싶다. 그들이 없었다면 오늘의 나도 없었을 것이므로 감사하다고 말하고 싶다.

## Notes

# Foundations of GTK Development: Introduction

---

서론

## 서론

애플리케이션에서 가장 중요한 측면 중 하나는 사용자와 상호작용하기 위해 제공되는 인터페이스다. 오늘날 사회에서 컴퓨터가 전례 없는 인기를 누리면서 사람들은 그러한 사용자 인터페이스가 그래픽할 것으로 기대하는 한편 개발자는 어떤 그래픽 툴킷을 재빠르게 사용해야 하는지에 대한 문제에 직면한다. 대다수의 경우 크로스 플랫폼의 GTK+ 라이브러리가말로 분명한 해결책으로 보인다.

GTK+의 학습이 때로는 벽찬 일임에도 불구하고 많은 기능들의 경우 문서가 부족한 것이 현실이며, 더군다나 API 문서로만 이해하기는 더욱더 힘들다. GTK+로 개발하기는 학습 곡선을 감소시키고 본인만의 방법으로 크로스 플랫폼의 그래픽 사용자 인터페이스를 생성하도록 만드는 데에 목적이 있다.

본 책에 실린 각 장(chapter)은 독자의 이해를 돕는 여러 예제를 포함하고 있다. 예제에 더불어 마지막 장에는 앞에서 소개한 주제를 합쳐 5개의 애플리케이션을 제공한다. 이러한 애플리케이션들은 학습한 내용을 모아 다양한 프로젝트를 달성하는 방법을 보여줄 것이다.

각 장의 앞 부분에서는 무엇을 다룰 것인지 개요를 제공하므로 원한다면 건너뛰어도 좋다. 대부분의 장에는 내용에 대한 이해도를 시험하기 위한 연습문제도 포함되어 있다. GTK+를 학습하는 최고의 방법은 직접 사용해 보는 것이기 때문에 다음 장으로 넘어가기 전에 연습문제를 모두 완료할 것을 권한다.

본 서적의 마지막에는 GTK+의 다양한 측면들에 대한 참조 역할을 하는 부록들이 여러 개 포함되어 있다. 부록에는 GTK+의 모든 위젯마다 시그널, 스타일, 프로퍼티를 열거하는 표를 비롯해 스톱 항목과 GError 타입의 전체 리스트가 실려 있다. 부록은 책을 다 읽고 나서 자신만의 애플리케이션을 생성할 때 참고해도 유용할 것이다. 부록 F는 본문에 걸쳐 소개된 모든 연습문제의 해답 설명을 포함한다.

## 누구를 위한 책인가

이 책은 기본 내용부터 시작해 좀 더 까다로운 개념으로 넘어가기 때문에 책 내용을 소화하기 위해 GTK+ 개발에 대한 사전 지식을 필요로 하지는 않는다. 본문에서는 C 프로그래밍 언어에 대해 적당히 이해하고 있을 것으로 가정한다. 또 Linux terminal에서 명령어를 실행하고 애플리케이션을 종료하는 (Ctrl+C) 데에 익숙해야 한다.

C 프로그래밍 언어에 대한 이해 뿐만 아니라 본문의 일부분은 일반적으로 Linux용 프로그래밍에 대한 추가 지식이 없이는 이해하기가 어려울 수도 있다. 기본 객체 지향의 개념을 이미 이해하고 있다면 이 책에서 더 많은 것을 얻게 될 것이다. 또 Linux가 프로세스를 처리하는 방식을 이해한다면 더 유용하겠다.

Linux에서 프로세스를 관리하거나 객체 지향을 구현하는 방식을 알지 못하더라도 이 책을 사용할 수는 있지만 하나 이상의 온라인 자원을 보완하여 사용하는 방법도 있겠다. 이 책의 웹 사이트 <http://www.gtkbook.com> 를 방문하면 유용한 링크와 지침서 목록을 찾을 수 있다. 책과 관련된 정보는 <http://www.apress.com> 에서 더 찾을 수 있다.

## 책의 구성

GTK+로 개발하기는 총 13개의 장으로 구성된다. 각 장은 한 가지 주제에 대한 전반적인 이해를 제공할 것이다. 가령 제 3장은 컨테이너 위젯에 대해 다루며, GtkContainer 클래스로부터 파생된 가장 중요한 위젯들을 다수 소개할 것이다.

이러한 구조 때문에 몇몇 장들은 내용이 길지도 모른다. 그렇다고 해서 앉은 자리에서 하나의 장을 모두 이해해야 할 필요는 없는데, 제시된 내용을 모두 이해하기가 힘들 수 있기 때문이다. 게다가 많은 예제들이 여러 페이지에 걸쳐 소개되므로 한 번에 몇 가지 예제에만 초점을 두어 어떻게 그들의 구문과 목적을 이해할 것인지 고려해보길 바란다.

각 장은 전문적인 GTK+ 개발자가 되도록 도와주는 중요하고도 독특한 관점들을 제공하는데, 이는 다음과 같다.

제 1장은 Linux 시스템에 GTK+ 라이브러리와 의존하는 패키지를 설치하는 방법을 알려준다. 또한 GLib, GObject, GDK, GdkPixbuf, Pango, ATK를 포함해 각 GTK+ 라이브러리의 개요를 제공한다. 제 2장은 두 개의 "Hello World" 애플리케이션을 처음부터 살펴본다. 첫 번째는 모든 GTK+ 애플리케이션에서 필요로 하는 기본 요점을 보여준다. 두 번째는 첫 번째 애플리케이션을 확장시켜 시그널, 콜백 함수, 이벤트, 자식 위젯을 다룬다. 이후 위젯 프로퍼티와 GtkWidget 위젯에 대해 학습할 것이다. 제 3장은 GtkWidget 구조의 소개로 시작된다. 다음으로 수평 및 수직 박스, 테이블, 고정 컨테이너, 수평 및 수직 패인, 노트북, 이벤트 박스를 교육한다. 제 4장은 프로그래머가 사용자와 상호작용하는 방법을 제공하는 기본 위젯을 다룬다. 여기에는 토글 버튼, 특수화된 버튼, 텍스트 엔트리, 스피너 버튼이 포함된다. 제 5장은 독자가 이용할 수 있는 다양한 범위의 내장된 대화상자(dialog)를 소개한다. 뿐만 아니라 본인만의 커스텀 대화상자를 생성하는 방법을 교육한다. 제 6장은 GLib에서 가장 유용한 기능들을 전반적으로 요약한다. 독자가 이용 가능한 데이터 타입을 다수 다룬다. idle 함수, time-out, 프로세스 띄우기, 동적 모듈 로딩, 파일 유틸리티 함수, 타이머, 기타 일반적인 유틸리티 함수 등을 소개한다. 제 7장은 스크롤을 이용한 창을 소개한다. 또한 텍스트 뷰 위젯을 이용하는 것과 관련해 상세한 설명을 제공한다. 그 외에 클립보드와 GtkSourceView 라이브러리에 관한 주제도 포함된다. 제 8장은 GtkTreeModel 객체를 이용하는 두 가지 타입의 위젯을 다룬다. 트리 뷰 위젯의 개요를 상세히 제공하고, 트리 모델 또는 문자열과 함께 콤보박스를 이용하는 방법을 보여준다. 제 9장은 두 가지 메뉴 생성 방법, 즉 메뉴얼 방식과 다이내믹 방식을 제공한다. 메뉴, 툴바, 팝업 메뉴, 키보드 접근자, 상태 바 위젯을 다룬다. 제 10장은 Glade 사용자 인터페이스 빌더(User Interface Builder)를 이용해 사용자 인터페이스를 디자인하는 방법에 대해 짧게 소개한다. 또 Libglade를 이용해 독자의 사용자 인터페이스를 동적으로 로딩하는 방법도 보여줄 것이다. 제 11장은 다른 위젯으로부터 자신만의 커스텀 GTK+ 위젯을 파생시켜 생성하거나, 처음부터 새로 생성하는 방식을 가르친다. 뿐만 아니라 인터페이스의 구현과 사용을 설명한다. 제 12장은 앞에서 다루지 못한 나머지 위젯을 다수 소개한다. 여기에는 최근 파일과 트레이 아이콘 지원 등 GTK+ 2.10에 도입된 여러 위젯이 포함된다. 제 13장은 실제 세계에서 발견할 수 있는 좀 더 긴 예제를 제공한다. 본문을 통해 학습한 개념을 어떻게 조합해 사용할 수 있는지를 보여준다.

여기까지 내용에 더해 위젯 프로퍼티, 시그널, 스타일, 스톡 항목, GError 타입, 연습문제 해답에 대한 설명의 참조로서 6개의 부록이 제공된다.

## 규칙

이 책은 GTK+ 코드와 일반 영어 문장을 구별하기 위해 다양한 활자체를 활용한다. 실제 코드는 고정폭 글꼴로 표시된다. 문단에서 코드의 전체 행이나 함수명, 시그널, 프로퍼티가 모두 이에 해당된다.

그 외에 준수하는 규칙은 아래와 같다.

### 연습 0-0. 연습문제 예제

이러한 박스는 해당 절의 주제에 관한 독자의 이해도를 검사하기 위한 연습문제를 보여준다. 질문, 코드 도전 과제, 주제에 대한 다양한 유형을 포함한다.

다음으로 넘어가기 전에 각 연습문제를 완료해야만 각 장에서 학습한 개념을 실습하고 앞의 장에서 학습한 개념을 합하도록 도와줄 것이다.

■ **Note** 해당 상자는 중요한 주석, 조인, 주의사항을 제공한다. 자신의 애플리케이션을 개발 시 필요하게 될 정보를 제공하므로 주의를 기울여야 한다.

GTK+는 그래픽을 하기 때문에 대부분의 출력은 이미지 형태겠지만 terminal 내 텍스트 출력은 행간 고정폭 글꼴로 표시된다.

## 준비 사항

진행 전에 몇 가지 준비해야 할 사항이 있다. 컴파일러, 텍스트 에디터, 터미널 에뮬레이터(terminal emulator), GTK+ 라이브러리, pkg-config 애플리케이션, 그리고 이 책을 준비한다.

본문에서 제공된 모든 컴파일러 명령은 <http://gcc.gnu.org> 또는 자신의 패키지 매니저를 통해 이용 가능한 GCC 컴파일러용이다. 대부분의 표준 C 또는 C++ 컴파일러도 작동하겠지만 GCC 이외의 컴파일러를 사용하면 제공된 것과는 다른 명령 집합을 사용해야 할 것이다.

어떤 텍스트 에디터도 괜찮으므로 자신에게 맞는 것을 선택해야 한다. 몇 가지 고려해 볼만한 인기 있는 텍스트 에디터로 Vim, Emacs, Leafpad, GEdit가 있겠다. Vim과 Emacs는 terminal 기반의 에디터인 반면 Leafpad와 GEdit는 그래픽 텍스트 에디터다.

GTK+ 라이브러리와 pkg-config 애플리케이션의 설치에 관한 지침서는 제 1장 마지막 절에 제공된다.

## 공식 웹 사이트

본 서적의 공식 웹 사이트, <http://www.gtkbook.com> 에서 추가 자원을 찾을 수 있을 것이다. 해당 웹 사이트에는 최신 문서, 유용한 자원의 링크, 저서를 통해 학습 시 도움이 되는 논문이 포함되어 있다. 또 서적의 예제마다 소스 코드를 다운로드할 수 있는 링크도 발견할 수 있다. <http://www.apress.com> 에서 찾을 수 있는 Apress 웹 사이트도 본 서적에 관한 추가 정보를 찾는 데 도움이 될 것이다.

웹 사이트에서 받은 소스 코드를 압축 해제하면 각 장에 실린 예제를 포함하는 폴더와 연습문제 해답이 실린 추가 폴더를 발견할 것이다. 현재 폴더 내에서 모든 파일을 빌드하기 위해선 make 를 실행할 수 있겠다. 제 2장에 주어진 컴파일 명령을 이용하거나 make sourcefile을 실행함으로써 단일 파일을 만드는 것도 가능하다. 가령, exercise2-1.c 를 빌드하기 위해서는 make exercise2-1 를 입력해야 한다.

## Notes

# FoundationsofGTKDevelopment:Chapter 01

## 제 1 장 시작하기

### 시작하기

GTK+로 개발하기를 읽게 된 것을 환영한다! 본 저서를 통해 당신은 전문적인 그래픽 프로그래머가 되도록 도와줄 수 있는 GIMP Toolkit(GTK+)에 대한 전반적인 이해를 요한다. 다음으로 넘어가기 전에, 본 저서는 C 프로그래머를 겨냥하였음을 인식해야 하며, 그에 따라 GTK+의 사용으로 바로 넘어갈 것이다. 이미 알고 있는 내용은 별도로 다루지 않을 것이다.

본 저서를 대체적으로 이해하기 위해서는 각 예제를 따라해 보고 각 장의 끝에 실린 연습문제를 시도해야 한다. Linux에서 GTK+를 시작하기란 꽤 간단한데, 현재 배포판 대다수는 일반적으로 필수 라이브러리와 툴이 함께 제공되기 때문이다.

그럼에도 불구하고 GNU 컴파일러 컬렉션(GCC), GTK+ 2.0 라이브러리, 연관된 개발 패키지를 포함해 몇 가지 툴은 미리 설치해두어야 한다. 본 장의 뒷 부분에서는 이러한 애플리케이션을 어떻게 설치하는지 학습할 것이다. 컴파일러가 없더라도 이 책을 이용할 수는 있으며, 연습문제를 실행한다면 더 많은 것을 얻게 될 것이다. GTK+를 학습하는 최선의 방법은 직접 사용해 보는 것이다!

■ **Note** 이 책에 선택된 컴파일러는 GCC로, <http://gcc.gnu.org> 에서 다운로드할 수 있다. 표준 C 또는 C++ 컴파일러라면 어떤 것이든 작동하겠지만 제공된 것과 다른 명령 집합을 이용해야 할 것이다. 그러한 컴파일러 명령은 본문에서 다루지 않을 것이다.

대부분 장의 마지막 부분에는 그 시점까지 학습한 내용을 설명하는 연습문제가 한 두 개 정도 실려 있을 것이다. 다음 장으로 넘어가기 전에는 자신의 지식을 재확인하도록 각 연습문제를 완료하도록 한다. 각 장은 앞의 장들에서 제시한 개념들을 바탕으로 구축하므로 좀 더 복잡한 예제들을 이해하려면 탄탄한 기반이 필요할 것이다.

이번 장에서 학습할 내용은 다음과 같다.

- GTK+와 X Window System의 역사는 두 가지 기술이 개발자들에게 미치는 엄청난 영향력과 관련해 맥락을 제공할 것이다.
- GTK+와 그것이 지원하는 라이브러리는 그래픽 애플리케이션 개발자에게 무엇을 제공하는가
- 이용 가능한 GTK+ 언어 바인딩은 무엇이며 어디서 다운로드할 수 있는가
- 자신의 컴퓨터에 GTK+와 그것이 의존하는 패키지를 어떻게 설치하는가

### GTK+의 간략한 역사

GIMP 툴킷(GTK+)은 본래 GNU 이미지 조작 프로그램(GIMP)이라 불리는 래스터 그래픽 에디터를 대상으로 설계되었다. Peter Mattis, Spencer Kimball, Josh MacDonald는 버클리의 캘리포니아 대학에서 1997년 eXperimental Computing Facility를 작업하면서 GTK+를 만들었다.

Lesser General Public License(LGPL)를 따르는 GTK+는 가장 유명한 Linux 데스크톱 환경인 GNOME과 XFCE의 기본 그래픽 툴킷으로 채택되었다. 본래 GTK+는 Linux 운영체제에서 사용되었는데 이후 Microsoft Windows, BeOS, Solaris, Mac OS X 등 다른 유사 유닉스 운영체제에서도 지원하도록 확장되었다.

■ **Note** 참고. LGPL은 GTK+를 다른 오픈 소스 그래픽 툴킷으로부터 구별하는 사항들 중 하나에 속한다. LGPL은 유명한 다른 오픈 소스 라이선스와 달리 특히 소프트웨어와 함께 사용하기 쉽다. 이러한 점이 바로 GTK+를 활용하는 GNOME 데스크톱 환경이 상업적 사회에서 많이 선택되는 이유이다.

GTK+는 최근 두 번째 안정된 배포판 주기인 GTK+2에 있다. 새로운 기능을 포함시킴과 동시 개발자들이 API 호환성을 중단하는 것이 옳다고 결정함에 따라 원본 branch인 GTK+1에 극적인 변화를 줄 필요가 있었다.

GTK+의 두 가지 branch는 호환되지 않기 때문에 동시에 설치가 가능하다. 애플리케이션을 빌드할 때 첫 번째 branch 대신 두 번째 branch를 사용하고 싶다면 컴파일러를 구별할 필요가 있는데, GCC를 이용하는 방법은 다음 장에서 학습할 것이다.

이 책에서는 모든 코드 예제에 대해 GTK+의 버전 2를 사용한다. GTK+ 2.10이 이미 배포되었지만 대부분의 예제는 두 번째 branch에선 어떤 버전과도 작동할 것이다. GTK+2는 이전 기종과 호환성을 유지하므로, GTK+2의 초기 배포판에서 작동하는 애플리케이션이라면 버전 2의 후반기 배포판과도 작동할 것이다.

## X Window System

1984년 Jim Getty와 Bob Scheifler는 메사추세츠 공과대학에서 Argus 시스템의 디버깅을 위한 플랫폼 의존적 디스플레이 환경으로서 X Window System(X11)을 만들었다. 현재 The X.Org Foundation에서 개발 중인 X11은 Linux 및 다른 유사 유닉스 운영체제에서 표준 디스플레이 매니저에 해당한다. 한 마디로 말하자면, X11은 비트맵 디스플레이를 위한 윈도잉 기능을 제공한다.

X Windows System은 Linux에서는 사용되지만 Microfot Windows와 같은 다른 운영체제에서는 사용되지 않는다. 따라서 GTK+의 또 다른 이점으로, 기본이 되는 렌더링 시스템이 무엇이든 상관없이 그와 상호작용 해야 한다는 의무가 사라진다는 점을 들 수 있다. Linux, Windows, Mac OS X 에 대한 코드를 작성한다면 운영체제가 무엇이든 코드 모양은 동일할 것이다.

Linux로 돌아와, X11은 창을 가장 기본적이고 추상적인 형태로 관리한다. 화면에 창을 그리고 창의 움직임을 처리한다. X11은 마우스나 키보드와 같은 입력 장치를 그래픽 환경에서 제어한다.

X11의 기본 프로그래밍 인터페이스인 Xlib는 그래픽 사용자 인터페이스를 생성하는 데에 필요한 툴을 제공한다. Xlib를 이용한 개발도 가능하지만 대부분 프로그래머들은 GTK+ 와 같은 그래픽 툴킷을 사용하는 편을 선호하는데, 저수준 호출은 모두 라이브러리의 메서드에 의해 숨겨지고 관리되기 때문이다.

X11이 다른 디스플레이 매니저들에 비해 가진 독특한 주요 기능들 중 하나는, 클라이언트와 서버가 서로 관계 없이 처리된다고 가정한다는 점이다. 이로 인해 클라이언트는 서버와 상관없이 원격 위치에 존재할 수 있다.

**Note** X Windows System 내 클라이언트와 서버에 대한 정의는 전형적인 정의와 다르다. 클라이언트는 애플리케이션이 실행되는 머신이다. 서버는 사용자의 원격 머신이 아니라 로컬 디스플레이를 의미한다.

X Window System의 또 다른 장점은 사용자 인터페이스를 의무적으로 지시하지 않는다는 것이다. 따라서 윈도 관리자(window manager)의 그래픽 사용자 인터페이스(GUI)에 대한 맞춤설정이 매우 유연하다. 윈도 관리자가 그렇게 상이한 인터페이스와 테마를 제공할 수 있는 이유이기도 하다. 이는 Linux 사용자가 오늘날 선택의 자유를 즐기도록 해준다.

아이러니하게도 이러한 자유는 X11에 쏟아지는 비평 중 가장 큰 부분을 차지하기도 한다. 많은 사람들은 이것이 Linux 개발자 공동체 내에서 분열을 조장할 것이라고 두려워한다. 하지만 지금으로선 자신의 요구에 가장 알맞은 윈도 관리자를 선택할 수 있는 기능을 즐기면 되겠다.

GTK+ 라이브러리는 프로그래머로서 X Window System과 직접 상호작용하지 않아도 되도록 생성되었다. 창과 위젯을 생성하고, 위젯과 사용자 간 상호작용을 처리할 수는 있지만 Xlib 함수 호출과 화면으로의 모든 직접 렌더링은 자동으로 처리된다.

따라서 본 저서는 X Window System을 더 이상 다루지 않을 것이며, 대신 GTK+ 라이브러리에 초점을 둘 것이다. X11과 X.Org Foundation에 관한 추가 정보는 <http://www.x.org> 에서 찾을 수 있다.

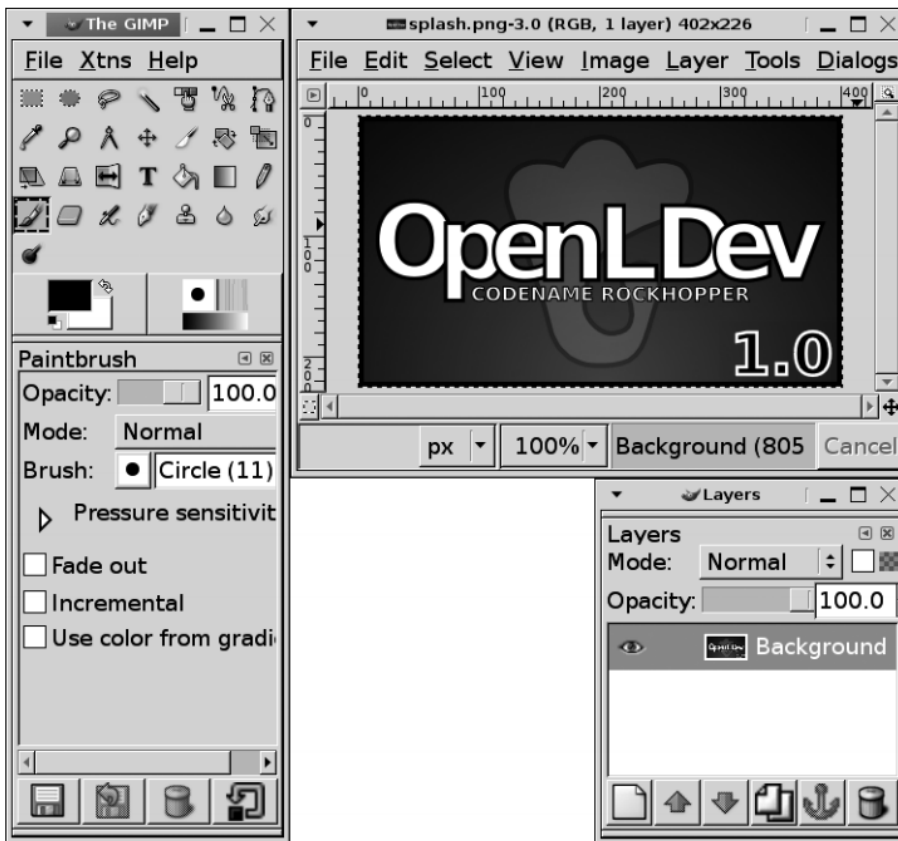


### GTK+와 지원 라이브러리

GTK+는 다수의 라이브러리에 의존하며, 각 라이브러리는 그래픽 애플리케이션 개발자에게 기능에 대한 구체적인 클래스를 제공한다.

GTK+는 C 프로그래밍 언어로 작성된 객체 지향 애플리케이션 프로그래밍 인터페이스(API)다. 이는 염두에 둔 클래스의 개념을 이용해, 자체에서 빌드되는 확장 가능 시스템을 생성하도록 구현된다. 사용된 객체 지향 프레임워크는 본래 GTK+ 라이브러리 자체의 일부로서 개발되었지만 이후 GTK+에서 분리되어 GObject라 불리는 다른 지원 라이브러리로서 GLib에 추가되었다. GObject는 객체 상속, 다형성, C에서 허용하는 한계인 데이터 은닉(data hiding)까지 포함해 C에서 모든 객체 지향 개발을 가능하게 한다.

GTK+ 라이브러리는 고유의 API를 통해 다른 라이브러리의 수많은 기능을 만들 수도 있다는 사실이 분명하지만 그보다는 그래픽 사용자 인터페이스를 빌드해야 하는 필요성을 제공하는 데에 중점을 둔다. GTK+ 자체에서 구현된 요소로는 버튼, 라벨, 텍스트 박스, 창과 같은 위젯이 있다. 이는 애플리케이션 레이아웃에 사용된 추상적 구성요소를 더 제공하고, 이벤트 캡처링(event capturing) 기능을 확장한다. 예를 들어, 그림 1-1은 GTK+를 사용하는 GIMP 애플리케이션의 스냅샷이다.



그 외에 동기식 및 비동기식 이벤트 프로세싱과 같이 GUI 개발에서 덜 시각적인 기본사항은 주로 다른 라이브러리에 의해 지원된다. 여전히 GTK+는 고유의 API를 통해 그 중 많은 기능으로 접근성을 제공한다.

Cairo라 불리는 2-D 벡터 그래픽 렌더링 라이브러리는 버전 2.8의 배포판부터 GTK+에 렌더링 기능을 제공해 오고 있다. Cairo는 모든 플랫폼과 시스템에 걸쳐 일관되게 벡터 그래픽을 렌더링하도록 생성되었다. 또 윈도 관리자 하에 가능한 위치에서 하드웨어 가속의 이용을 허용한다.

Cairo 자체는 이 책에서 다루지 않을 것이지만 GTK+의 인쇄 API와 어떻게 연관되는지는 살펴볼 것인데 그 호출들은 당신이 상호작용하게 될 GTK+의 계층 아래에 존재하기 때문이다. 후에 GTK+ 소스 코드를 해킹하기로 결정한다면 살펴보고 싶을 법한 중요한 내용이다. Cairo에 대한 추가 정보는 <http://www.cairographics.org> 를 방문한다.

**GLib**

GLib는 다목적 유틸리티 라이브러리로, 많은 유용한 비그래픽 기능의 구현에 사용된다. GTK+에서 필요로 하지만 독자적으로 사용할 수도 있다. 이로 인해 일부 애플리케이션은 다른 GTK+ 라이브러리 없이 GLib가 제공하는 많은 기능 덕분에 GLib만 사용한다.

GLib를 이용 시 주요 장점은 코드를 재작성하지 않고 지원하는 운영체제에서 본인의 코드를 실행하도록 허용하는 크로스 플랫폼 인터페이스를 제공한다는 점이다! 또 다른 GLib의 장점으로는 개발자에게 제공하는 엄청난 수의 데이터 타입을 들 수 있다. GLib가 제공하는 데이터 타입 몇 가지를 열거할 것인데, 상세한 내용은 제 6장에서 다룰 것이다.

- GLib는 C 프로그래머들에게 보통 다른 언어에 기본 값으로 포함되어 있는 데이터 타입을 다수 제공하는데, 단일 연결 리스트와 이중 연결 리스트를 예로 들 수 있겠다. 그 외 기본 데이터 타입으로는 양단 (double-ended) 큐, 자체적으로 균형을 이룬(self-balancing) 이진 트리, 불균형 n-ary 트리가 있다.
- 해시 테이블은 데이터에 대한 포인터의 리스트를 생성하도록 해준다. 이는 연결 리스트와는 차이가 있는데, 정수 참조에 의한 요소 접근 대신 두 번째 포인터를 키로서 명시하기 때문이다.
- GLib에서 문자열은 C++에서 문자열과 유사한데, 데이터가 추가되면 자동으로 증가하는 텍스트 버퍼이기 때문이다. 이들은 또 printf() 함수 계열로의 호출과 쉽게 통합된다.
- 메모리 슬라이스는 크기가 모두 동일한 메모리 청크를 효율적으로 생성하는 방법이다. 메모리 슬라이스를 이용해 균일한 크기의 요소로 된 배열을 생성할 수 있다. 이러한 구조는 GLib 2.10 배포판에서 메모리 청크를 대신해 도입되었다.
- 캐시는 쉬운 API에서 크고 복잡한 데이터 구조를 공유하도록 해주기 때문에 공간을 절약하는 데 도움이 된다. 이들은 스타일과 그래픽 컨텍스트를 위해 GTK+에 의해 사용되기도 하는데, 두 객체 모두 많은 자원을 소모하기 때문이다.

GLib는 다른 데이터 타입도 제공하는데, 그 중 다수는 제 6장에서 소개할 것이다. 뿐만 아니라 GLib는 데이터 타입 외의 기능들도 구현한다. 그리고 수많은 타입의 유틸리티 함수를 제공하기도 한다. 몇 가지만 예를 들자면 파일 조작, 국제화 지원, 문자열, 경고, 디버깅 플래그, 동적 모듈 로딩, 자동 문자열 완성 등등에 대한 유틸리티 함수를 찾을 수 있을 것이다.

제 6장에서는 idle 함수, time-out 함수, 타이머에 관해 학습할 것인데, 이들은 모두 개발자들에게 흥미롭고 다양한 가능성을 열어준다. idle 함수는 프로세서가 애플리케이션을 위해 어떤 일도 하고 있지 않을 때 함수를 호출하도록 해준다. timeout 함수들은 당신이 제공한 시간 간격으로 함수를 호출할 때 사용된다. 타이머는 그것이 초기화된 후 시간이 얼마나 경과했는지 추적한다. 이러한 함수들은 애플리케이션이 각각 idle 상태일 때 업데이트를 검사하거나, 자동 저장 기능을 구현하거나, 경과 시간을 추적하는 데에 사용되기도 한다.

GLib는 그 크로스 플랫폼 특성 덕분에 프로세스 띄우기, 파일 조작, 메모리 할당, 스레드를 대상으로 사용하기 쉽다. 이들 중 어떤 것이든 다수의 플랫폼을 대상으로 개발을 시도한다면 악몽이 될지도 모른다. GLib는 이러한 귀찮은 일을 알아서 처리하므로 독자는 크로스 플랫폼의 호환성 문제를 염려하지 않아도 된다.

**GObject**

GLib 객체 시스템(GObject)은 본래 GtkObject 클래스의 형태로 GTK+1의 일부였다. GTK+ 2.0이 배포되면서 고유의 라이브러리로 이동되었고 GLib와 함께 배포되었다.

GObject는 그 복잡성 때문에 비평을 받곤 했는데, 그 API들이 극단적으로 길어지는 것처럼 보였기 때문이다. 하지만 본래는 다른 프로그래밍 언어로부터 C 객체로 쉽게 접근하도록 생성된 것이다. 다른 프로그래밍 언어로부터 C 객체로 쉽게 접근하는 기능이 C에서 구현된다 하더라도 다른 프로그래밍 언어에 이용 가능한 바인딩의 범위가 거대해지는 결과를 야기한다.

각 프로그래밍 언어는 데이터 타입에 대해 서로 다른 접근법을 제공하기 때문에 까다로운 일인데, 그러한 차이는 표면에 드러나거나 각 언어의 내부에 나타난다. 가령, C에서는 char, long, integer를 포함한 데이터 타입을 갖는다. Perl을 비롯한 다른 언어들에서는 이와 유사한 데이터 타입을 찾을 수 없는데, 각 객체의 타입은 그것이 어떻게 사용되느냐에 따라 결정되기 때문이다. GObject는 이러한 한계점은 해결했지만 새로운 객체를 파생하는 과정이 대단히 복잡하다는 단점이 있다.

GObject는 C에서 완전한 기능을 갖춘 객체 지향 인터페이스를 구현하기도 하는데, 이는 이번 절과 본 저서의 나머지 부분에 걸쳐 세부적으로 다룰 것이다. 이 시스템은 GTK+ 위젯 계층구조뿐만 아니라 GTK+가 지원하는 라이브러리에 구현된 다수의 객체에도 기반이 된다. GObject의 객체 지향 인터페이스는 GType이라고 불리는, 포괄적이고 동적인 타입의 시스템에 의해 일부 구현된다. GType은 프로그래머들이 단일 상속된 (singly-inherited) 클래스 구조를 통해 서로 다른 다수의 동적 데이터 타입을 구현하도록 허용한다. 단일 상속된 클래스는 객체 계층구조로서, 각 자식(child) 클래스는 하나의 부모 클래스로부터 직접적으로 파생되어야만 한다. 이는 GTK+ 위젯을 소개한 후 제 2장에서 상세히 논할 것이다.

확장 가능한 데이터 타입의 생성 기능 외에도 GObject는 프로그래머들에게 많은 클래스가 아닌 (또는 기본적인) 데이터 타입을 제공한다. 클래스가 아닌 데이터 타입은 루트 클래스로서, 다른 클래스들은 이로부터 파생된다. 루트 클래스 자체는 어떤 클래스로부터도 파생되지 않음을 명심하는 것이 중요하다.

표 1-1 은 가장 중요한 클래스가 아닌 데이터 타입의 목록을 제공한다. GType 매크로, C 변수 디스크립터, 설명이 각각 설명되어 있으며, 해당하는 경우 범위도 표시한다.

GType	C Type	설명
G_TYPE_NONE		void와 동일한 빈 타입.
G_TYPE_CHAR	gchar	표준 C char 타입과 동일.
G_TYPE_INT	gint	표준 C int 타입과 동일. 값은 G_MININT와 G_MAXINT 사이여야 함.
G_TYPE_LONG	glong	표준 C long 타입과 동일. 값은 G_MINILONG과 G_MAXLONG 사이여야 함.
G_TYPE_BOOLEAN	gboolean	표준 Boolean 타입으로, TRUE 또는 FALSE를 보유.
G_TYPE_ENUM	GEnumClass	C enum 타입과 동일한 표준 열거.
G_TYPE_FLAGS	GFlagsClass	Boolean 플래그를 보유하는 비트 필드.
G_TYPE_FLOAT	gfloat	표준 C float 타입과 동일. 값은 음의 G_MAXFLOAT과 G_MAXFLOAT 사이여야 함.
G_TYPE_DOUBLE	gdouble	표준 C double 타입과 동일. 값은 음의 G_MAXDOUBLE과 G_MAXDOUBLE 사이여야 함.
G_TYPE_STRING	gchar*	NULL로 끝나는 C 문자열과 동일.
G_TYPE_POINTER	gpointer	타입이 정해지지 않은 포인터 타입으로, void*와 유사.

표 1-1. 클래스가 아닌 표준 GObject 데이터 타입

GObject는 두 가지 중요한 또 다른 데이터 타입인 GValue와 GObject도 GTK+에 포함하여 제공한다. GValue는 일반적 컨테이너로서, 시스템이 이미 인식하는 구조라면 무엇이든 보유할 수 있다. 이는 함수가 임의 타입의 데이터 조각을 리턴하도록 허용한다. GValue가 없이는 GTK+의 객체 지향 특성이 불가능할 수 있다.

G\_TYPE\_GOBJECT 또는 GObject는 GTK+의 위젯 클래스 상속 구조가 기반으로 하는 기본적인 타입이다. 이는 위젯들로 하여금 그들 부모의 프로퍼티를 상속하도록 허용하는데, 스타일 프로퍼티와 시그널도 이에 포함된다.

GObject는 단일 상속 시스템으로서, 각 자식 클래스는 하나의 부모 클래스만 가질 수 있다. 파생된 자식은 부모의 모든 특성을 상속 받는데, 어떤 점에서든 자식은 또 부모에 "해당하기" 때문이다. 이러한 시스템을 이용해 커스텀 GTK+ 위젯을 파생하는 방법은 제 11장에서 학습할 것이다.

GObject는 또한 위젯에게 시그널 시스템, 객체 프로퍼티 시스템, 메모리 관리를 제공하기도 한다. 이 세 가지 개념은 다음 장에서 살펴볼 것이다.

**GDK**

GIMP 드로잉 키트(GDK)는 본래 X Windows System용으로 설계된 컴퓨터 그래픽 라이브러리로서, 저수준 드로잉과 윈도우 함수를 다룬다. GDK 는 Xlib와 GTK+의 매개체 역할을 한다.

이는 모든 GTK+ 애플리케이션에서 드로잉, 래스터 그래픽, 커서, 글꼴을 렌더링한다. 또 모든 GTK+ 프로그램에서 구현되기 때문에 GDK는 드래그 앤 드롭 지원과 윈도우 이벤트도 제공한다.

GDK는 GTK+ 위젯에게 화면에 그림을 그릴 수 있는 기능을 제공한다. 이를 위해선 모든 위젯이 연관된 GdkWindow 객체를 가져야 하지만 몇 가지 예외에 속하는 위젯이 있는데 이는 후에 다른 장에서 논할 것이다. GdkWindow는 기본적으로 위젯이 그려지는 화면에 위치한 직사각형 모양의 영역이다. GdkWindow 객체는 위젯이 X Windows System 이벤트를 감지하도록 허용하기도 하는데, 이와 관련된 내용도 다음 장에서 다룰 것이다.

GDK는 Windows와 Mac OS X로 이식되었다. GTK+ 2.8 배포판 이후로는 Cairo를 지원하기도 한다.

**GdkPixbuf**

GdkPixbuf는 작은 라이브러리로, 클라이언트 측 이미지 조작 함수를 제공한다. 이것은 GNOME 이미징 모델 (Imlib)을 대신해 생성되었다. 이미지는 파일로부터 로딩될 수도 있고, 이미지 데이터가 라이브러리 함수로 직접 입력될 수도 있다. 후에 이미지를 트리 뷰로 추가하거나 새로운 GtkImage 위젯을 생성할 때도 해당 라이브러리를 사용할 것이다.

GdkPixbuf 이미지의 한 가지 장점으로, 이미지를 참조 계수(reference-counted)할 수 있다는 점을 들 수 있다. 이는 GdkPixbuf 이미지를 메모리에 한 번만 보관하면서 여러 위치에 표시할 수 있음을 의미한다. 모든 참조 계수가 감소할 경우에만 메모리 해제될 것이다.

GdkPixbuf 라이브러리는 GNOME과 함께 배포되는 2-D 드로잉 라이브러리인 Libart의 장점을 취해 이미지로 의 변형(transformation)을 지원한다. 이 때문에 프로그래머는 이미지를 원하는 대로 기울이고(shear), 크기를 조정하며(scale), 회전시킬 수 있는 것이다. 이후 이미지는 GdkRGB 라이브러리와 드로잉 가능한 영역을 이용해 렌더링이 가능하다. GdkPixbuf는 이렇게 다양한 범위의 특수 툴을 이용함으로써 상당히 높은 클래스의 이미지 렌더링을 제공할 수 있는 것이다.

GdkPixbuf는 소규모 라이브러리임에도 불구하고 이미지의 조작과 표시에 다양한 범위의 함수를 제공한다. 라이브러리는 본문에서 가장 기초적으로만 사용될 것이다. GdkPixbuf 의 고급 주제에 관한 정보는 그에 대한 API 문서를 참조하도록 한다.

**Pango**

GDK는 창과 이미지 렌더링을 처리하는 반면 Pango는 GTK+ 버전에 따라 Cairo 또는 Xft와 함께 텍스트 및 글꼴 출력을 제어한다. 또 부수적 라이브러리를 사용하지 않고 in-memory 버퍼로 직접 렌더링이 가능하다.

■Note Pango는 "모두"를 뜻하는 그리스어 pan과, "언어"를 뜻하는 일본어 go에서 기원한 것이다. 이러한 용어가 선택된 이유는 완전히 국제화된 글꼴 렌더링 시스템을 생성함으로써 모든 언어를 지원하는 것이 Pango의 디자인 목표 중 하나에 해당하기 때문이다.

Linux에서 Pango는 클라이언트 측 글꼴에 대해 FreeType과 fontconfig 라이브러리를 사용한다. Pango를 다른 것과 비교할 때 두드러지는 점은 광범위한 언어 집합을 지원한다는 것이다. 사실상 세계 거의 모든 언어를 지원하므로 애플리케이션에서 국제화된 텍스트를 렌더링하는 것은 문제가 되지 않는다.

Pango 내의 모든 텍스트는 UTF-8 인코딩으로 내부적으로 표현된다. UTF-8은 유닉스 플랫폼에서 일반적으로 볼 수 있는 8 비트 소프트웨어와 호환된다. UTF-8에서 오프셋은 비트가 아니라 문자를 기반으로 계산되는데, 각 문자가 1 바이트 이상 차지할 수도 있기 때문이다. 이는 GtkTextView를 사용하는 방식을 학습하게 될 제 7장에서 중요해지는데, 문자 오프셋마다 비교해야 하나 항상 1 바이트인 것은 아니기 때문이다.

Pango는 광범위한 텍스트 속성을 지원한다. 이러한 속성으로는 언어, 폰트 패밀리, 스타일, 굵기(weight), 폭(stretch), 크기, 전면색, 배경색, 밑줄, 취소선, rise, 모양(shape), 크기조정(scale)을 들 수 있지만 이에 국한되지 않는다. 이러한 속성들 중 다수는 자체적으로 다중 옵션을 지원한다.

편의상 Pango Text Markup Language는 HTML과 유사한 형태로 텍스트 속성을 나타내는 간단한 태그 집합을 제공한다. 이러한 markup 언어를 이용하면 위젯 내 임의의 텍스트 부분에 대해 글꼴 스타일을 쉽게 변경할 수 있다. 이는 특히 Glade 사용자 인터페이스 빌더(Glade User Interface Builder)로 사용자 인터페이스를 생성 시 유용한데, 위젯의 텍스트 내용 필드에 직접 태그를 입력할 수 있기 때문이다.

후에 여러 장에 걸쳐 위젯의 글꼴을 사용자의 기본값 외의 글꼴로 변경하는 다수의 예제에서는 Pango를 사용할 것이다. PangoFontDescription 객체 또는 Pango Text Markup Language를 이용하면 되겠다.

**ATK**

애플리케이션을 디자인할 경우에는 자신이 경험할 수 있는 장애를 고려하는 것이 중요하다. 그에 따라 접근성 툴킷(ATK)은 GTK+ 위젯에게 접근성 문제를 처리하기 위한 내장된 방식을 제공한다.

그 외에 ATK가 지원하는 몇 가지 예로는 시각적으로 장애가 있는 이들을 위한 스크린 리더와 고대비 시각 테마, 그리고 운동 제어가 안 되는 사람들을 위해 고정키(sticky key)와 같은 키보드 행위 수정자(behavior modifiers)가 있다.

이는 생산용 애플리케이션의 디자인에 있어 중요한 부분이긴 하나 이번 책에서는 ATK를 다루지 않을 것이다. 우리는 GTK+ 위젯을 생성하는 방법과 ATK를 사용하기 전에 자신만의 커스텀 위젯을 생성하는 방법을 학습할 필요가 있다. 따라서 필자는 GTK+와 다른 필수 요소에 중점을 두고자 한다.

접근성을 항상 유념하고, 자신의 애플리케이션에서 ATK를 처리할 준비가 되었을 때 라이브러리를 다시 찾는 것이 중요하겠다.

**언어 바인딩**

원본 형태로 된 GTK+는 C 프로그래밍 언어와 함께 사용이 가능하지만 바인딩은 다수의 다른 목적을 위해 생성되었다. 가장 유명한 언어 바인딩을 열거하자면 아래와 같지만 전체 목록은 <http://www.gtk.org/bindings.html> 에서 찾을 수 있다.

- Gtkmm은 공식 C++ 바인딩 집합이다. 기존 버전과 호환이 가능하기 때문에 C++와 함께 GTK+를 이용 가능하지만 Gtkmm은 일련의 클래스에서 모든 GTK+ 기능을 제공하며, 그 스타일은 모든 C++ 프로그래머에게 익숙할 것이다. Gtkmm, GLibmm, Libglademmm, 기타 의존 패키지에 대한 소스는 <http://www.gtkmm.org> 에서 이용할 수 있다.
- <http://www.pygtk.org> 에서 이용 가능한 PyGTK는 GTK+ 라이브러리에 대해 Python 바인딩을 제공한다. PyGTK를 이용 시 장점은 메모리 관리와 타입 캐스팅을 처리한다는 점이다. 이는 다른 언어 바인딩을 사용하는 프로그래머에게 불편할 수 있는 문제들을 완화시켜준다.
- <http://gtk2-perl.sf.net> 에서 이용 가능한 Gtk2-perl은 객체 지향 Perl 툴킷에 모든 GTK+ 라이브러리를 제공한다. 각 라이브러리는 Glib, Gtk2, Gtk2::GladeXML 이라 불리는 모듈로 나뉜다. 스크립팅 언어에 대한 대부분의 GTK+ 바인딩과 마찬가지로 메모리 관리는 언어의 기능들에 의해 처리된다.
- PHP-GTK는 GTK+에 대한 PHP 언어 바인딩의 처리를 허용한다. PHP 바인딩은 클라이언트 측 크로스 플랫폼 GUI 애플리케이션을 생성하도록 해준다. PHP-GTK는 <http://gtk.php.net> 에서 이용할 수 있다. 이 주제는 Apress에서 출판한 Scott Mattocks의 저서 Pro PHP-GTK 에서도 다룬다 (Berkeley, 2006).
- Gtkmm 과 많이 비슷한 Java-Gnome은 GTK+ 라이브러리를 위해 진정한 객체 지향의 플랫폼을 제공한다. <http://java-gnome.sf.net> 에서 이용 가능하며, Java에서 GTK+ 애플리케이션을 개발 시 필요한 모든 라이브러리를 제공한다.
- Gtk#은 광범위한 운영체제에서 C# 애플리케이션에 대한 GTK+ 바인딩을 제공한다. 이는 <http://www.mono-project.com> 에서 Mono Project에 의해 제공된다.

## GTK+ 설치하기

프로그래밍을 시작하기 전에 우선 자신의 시스템에 GTK+와 그것이 의존하는 패키지를 설치해야 한다. 이번 절은 Linux와 유사 유닉스 운영체제에서 GTK+를 설치하는 것을 다룬다.

Ubuntu, Debian, Fedora Core 등 다수를 포함해 패키지 매니저가 있는 Linux 배포판을 사용 중이라면 제공되는 사전 컴파일된 바이너리를 설치해야 한다. GTK+2 라이브러리, pkg-config, 그리고 그들이 의존하는 패키지들이 필요할 것이다.

GTK+의 개발 패키지, 그리고 각 패키지가 의존하는 패키지도 필요하다. Debian과 Debian을 기반으로 한 배포판에서 이러한 패키지들은 -dev에서 끝이 난다. RedHat Package Manager(RPM)를 사용하는 Fedora Core와 다른 배포판에서는 -devel에서 끝날 것이다. GTK+의 개발 패키지를 설치할 경우 필요로 하는 의존 패키지들은 가장 최신 패키지 매니저가 모두 자동으로 처리할 것이다. 배포된 패키지의 설치에 관한 상세한 정보는 자신의 Linux 배포판 문서를 참조해야 한다.

소스 아카이브에서 GTK+와 그것이 의존하는 패키지들을 설치하려면 지금부터 잘 살펴보도록 한다. GTK+는 컴파일에 표준 GNU 툴을 사용하는데, autoconfig는 설정을 비롯해 이식성 문제의 처리에 사용되고, automake는 makefiles의 빌드, libtool은 공유 라이브러리의 빌드, make는 바이너리의 컴파일과 설치 시 사용할 수 있다.

가장 최신 GTK+ 소스는 <http://www.gtk.org/download> 에서 찾을 수 있다. ATK, GLib, GTK+, Pango의 최신 버전도 다운로드 해야 한다. 의존하는 패키지 디렉터리의 Cairo, JPEG, libpng, pkg-config, tiff도 필요할 것이다.

Linux의 오래된 버전을 사용할 것이라면 libiconv를 설치해야 한다. 대부분의 시스템은 이미 해당 패키지를 갖고 있으므로 이것이 없이 시작하여 향후 문제에 직면하면 라이브러리를 설치하는 편이 안전하다. 대부분 현대 Linux 배포판에서 표준으로 제공되긴 하지만 libintl, fontconfig, FreeType 패키지도 설치해야 할 것이다.

이러한 패키지들은 정확히 다음과 같은 순서대로 설치"해야만" 작동함을 주목해야 한다. GTK+ FTP 사이트에서 의존 패키지 디렉터리로부터 모든 패키지를 설치했다면 GLib, Pango, ATK, GTK+ 순으로 설치해야 한다.

아래 프로시저는 한 번에 하나의 소스 패키지에서 사용되어야 한다. 다음으로 넘어가기 전에 각 라이브러리가 성공적으로 설치되어야 하며, 그렇지 않으면 프로시저가 작동하지 않을 것이다.

GTK+를 설치할 준비가 되면 이제 시작해보자. GTK+ FTP 사이트에서 패키지를 다운로드 했다면 자신이 다운로드한 아카이브 유형에 따라 파일을 압축해제(extract)하는 다음 명령 중 하나를 사용할 수 있다.

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

압축해제된 아카이브의 디렉터리로 이동시키면 configure라 불리는 셸 스크립트를 확인할 것이다. 해당 스크립트는 소스 배포판 내 각 디렉터리를 통해 재귀적으로 파싱할 것이며, 자신의 운영체제에 맞춤설정된 템플릿 makefiles를 생성할 것이다. 각 템플릿 파일은 Makefile.in으로 명명될 것이다. 아래는 당신이 사용할 수 있는 설정 명령의 예제이다.

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

설정 스크립트는 옵션 개수로 전달할 수 있다. --prefix=/usr을 이용하면 앞의 예제는 make에게 /usr이 있는 패키지를 루트 디렉터리로서 설치할 것을 알린다. GTK+ 설정 스크립트로 전달할 수 있는 옵션에는 그 외에도 여러 가지가 있다.

표 1-2는 구체적으로 GTK+ 설정에 전달 가능한 매개변수의 목록을 짧게 보여준다. 어떤 패키지든 매개변수의 전체 목록을 확인하려면 ./configure --help를 이용할 수 있다.

옵션	설명
--enable-debug	no로 설정 시 디버깅과 asserts가 비활성화된다. yes로 설정 시 런타임 디버깅이 활성화된다. 기본값은 minimum으로, cast 검사만 비활성화한다.
--enable-shm	이용 가능할 경우 공유 메모리를 켜고, --disable-shm을 이용해 비활성화한다.
--enable-xkb	X Window System 키보드 확장을 지원하고, --disable-xkb를 이용해 비활성화한다.
--disable-rebuilds	모든 소스 자동생성 규칙을 비활성화하고, --enable-rebuilds를 이용해 활성화한다.
--enable-visibility	ELF 시각성 속성을 이용하고, --disable-visibility를 이용해 비활성화한다.
--with-xinput	yes를 이용해 자신의 애플리케이션에 XInput 확장을 지원하고, no를 이용해 비활성화한다.
--with-gdktarget=	non-default GDK 대상을 선택한다. 해당 매개변수에 대한 옵션으로 x11, linux-fb, win32, quartz, directfb가 있다.
--disable-shadowfb	linux-fb에서 shadowfb에 대한 지원을 비활성화하거나 --enable-shadowfb를 이용해 활성화한다.
--enable-fbmanager	GtkFB를 통해 프레임 버퍼 매니저 지원을 활성화한다.
--disable-modules	이는 GdkPixbuf에 대한 모든 이미지 파일 포맷 로더들이 GTK+ 라이브러리로 정적으로 빌드되어야 함을 나타낸다. --enable-modules를 이용해 이들을 공유 라이브러리로 빌드할 수 있다.
--with-include-loaders	이는 PNG나 JPEG와 같이 포함해야 할 이미지 로더를 명시하도록 해준다.

표 1-2. GTK+ 설정 옵션

패키지를 설정하고 나면 다음과 같은 명령 집합을 이용해 패키지를 빌드 및 설치할 수 있는데, make install과 ldconfig는 루트 사용자로서 실행될 필요가 있음을 인지하는 것이 중요하다.

```
make
make install
ldconfig
```

ldconfig 명령은 모든 시스템에서 필수적인 것은 아니지만 안전한 측에서 실행시키도록 해야 한다. 그래야만 다음 패키지를 컴파일하기 전에 본인이 설치한 라이브러리를 시스템이 확실히 인식할 수 있을 것이다.

**연습 1-1. 설치 검증하기**

소스 패키지로부터 GTK+ 라이브러리를 설치할 경우 성공적인 설치를 검증하는 간단한 방법이 제공된다. 이를 위해서는 /usr/bin에 설치된 gtk-demo 애플리케이션을 실행시켜야 한다. terminal에서 아래 명령을 실행하거나 실행 파일을 더블 클릭한다.

```
/usr/bin/gtk-demo
```

설치가 성공하면 "GTK+ Code Demos"라는 제목으로 된 창이 표시될 것이다. 그 창에서는 열거된 각 위젯에 대한 정보와 소스 코드를 확인할 수 있다. 이는 앞으로 학습하게 될 많은 위젯을 연구할 수 있는 기회를 제공한다.

애플리케이션을 시작 시 문제에 직면할 경우 terminal에 표시된 오류에 집중한다. 이는 어떤 라이브러리가 문제를 야기하는지를 알려줄 것이다.

GTK+ 라이브러리와 그것이 의존하는 패키지를 모두 설치했다면 다음 장으로 넘어갈 준비가 되었으며, 이제 모든 GTK+ 애플리케이션이 필요로 하는 가장 기본적인 요소를 보여주는 간단한 예부터 시작하겠다.

## 요약

이번 장에서는 GTK+ 라이브러리와 X Windows System의 역사와 그들의 사용 용도를 학습하였다.

또 그래픽 위젯 라이브러리뿐만 아니라 그 지원 라이브러리로서의 GTK+도 소개하였다. 이러한 라이브러리들은 다음과 같다.

- GLib는 다용도의 유틸리티 라이브러리로서, 데이터 타입, 파일 관리, 파이프, 스레드 등을 포함해 수많은 유용한 비그래픽 기능들을 구현하는 데 사용된다.
- GLib 객체 시스템(GObject)은 객체 지향 GType 시스템을 구현하며, 시그널 및 프로퍼티 시스템을 제공하기도 한다.
- GIMP 드로잉 키트(GDK)는 본래 X Window System용으로 설계된 컴퓨터 그래픽 라이브러리로서, 저수준 드로잉과 윈도우 함수를 다룬다.
- GdkPixbuf는 작은 라이브러리로, 클라이언트 측 이미지 조작 함수를 제공한다. 이는 Imlib를 대신해 생성되었다.
- Pango는 글꼴 렌더링에 사용된다. 이는 UTF-8 인코딩을 사용하므로 모든 국제화 텍스트 형태를 지원할 수 있다.
- 접근성 툴킷(ATK)은 모든 GTK+ 위젯에게 접근성을 처리하는 내장된 방식을 제공한다.

본 장의 마지막 두 절은 다른 프로그래밍 언어에서 GTK+를 구현하는 데 이용 가능한 모든 언어 바인딩과, GTK+ 라이브러리를 설치하는 방법을 보였다. 언어 바인딩은 본래부터 GObject가 설계된 방식 덕분에 가능하다.

제 2장에서는 창, 라벨, 버튼 위젯뿐만 아니라 위젯 계층구조 시스템도 함께 소개하고, 기본 GTK+ 애플리케이션에서 이러한 위젯을 사용하는 방법을 학습할 것이다.

## Notes



# FoundationsofGTKDevelopment:Chapter 02

## 제 2 장 자신의 첫 GTK+ 애플리케이션

### 자신의 첫 GTK+ 애플리케이션

제 1장에서는 그래픽 애플리케이션 개발자로서 자신의 GTK+ 라이브러리에서 어떤 것들을 이용할 수 있는지에 대한 개요를 제시하였다. 이번 장에서는 자신만의 GTK+ 애플리케이션을 작성하는 방법을 학습할 것이다. 간단한 예제로 시작하지만 이번 장에서는 중요한 개념들을 많이 제시하고 있다. 향후 본인이 작성하는 GTK+ 애플리케이션마다 의존하게 될 주제를 다룰 것이다. 따라서 여느 장과 마찬가지로 다음으로 넘어가기 전에 아래의 여러 페이지에 걸쳐 제시된 개념을 이해하도록 한다.

이번 장에서 학습하게 될 내용은 다음과 같다.

- 모든 GTK+ 애플리케이션에서 요하는 기본 함수 호출
- GCC를 이용해 GTK+ 코드를 컴파일하는 방법
- GTK+ 위젯 시스템의 객체 지향적 특성
- 자신의 애플리케이션에서 시그널, 콜백, 이벤트가 하는 역할
- Pango Text Markup Lanaguage를 이용해 텍스트 스타일을 변경하는 방법
- 본 장에 제시된 위젯에 제공되는 다른 다양하고 유용한 함수들
- GtkButton 위젯을 이용해 클릭 가능한 GtkLabel을 만드는 방법
- GObject 메서드를 이용해 객체의 프로퍼티를 얻고 설정하는 방법

### Hello World

필자가 지금까지 읽은 프로그래밍 저서는 하나같이 "Hello World" 애플리케이션을 예로 시작한다. 그리고 필자도 그 전통을 깨고 싶지는 않다.

예제를 바로 시작하기 전에 알아두어야 할 점이 있는데, 모든 예제에 대한 소스 코드는 본 책의 웹 사이트 [www.gtkbook.com](http://www.gtkbook.com)에서 다운로드가 가능하다는 것이다. 본 장의 후반부에 제시된 방법을 이용해 각 예제를 컴파일하거나 패키지의 base 폴더에 발견되는 지침을 따라도 된다.

리스팅 2-1은 이번 책에서 처음으로 소개하는 가장 간단한 GTK+ 애플리케이션이다. 이는 GTK+를 초기화하고, 창을 생성하여 사용자에게 표시하며, 프로그램이 종료될 때까지 기다린다. 이것은 매우 기본적인지만 당신이 생성하는 모든 GTK+ 애플리케이션마다 가져야 하는 기본 코드를 보여준다!

**■Note** 리스팅 2-1의 애플리케이션은 애플리케이션의 종료 방법은 제공하지 않는다. 창의 모서리에 X를 클릭하면 창은 닫히지만 애플리케이션은 여전히 실행될 것이다. 따라서 애플리케이션을 강제로 나가려면 terminal 창에서 Ctrl+C를 눌러야 할 것이다.

리스팅 2-1. World 와 인사하기 (helloworld.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window;

    /* Initialize GTK+ and all of its supporting libraries. */
    gtk_init (&argc, &argv);

    /* Create a new window, give it a title and display it to the user.
```

```

*/
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "Hello World");
gtk_widget_show (window);

/* Hand control over to the main loop. */
gtk_main ();
return 0;
}
    
```

위의 내용에서 <gtk/gtk.h> 파일은 GTK+ 에서 이용 가능한 모든 위젯, 변수, 함수, 구조를 포함할 뿐만 아니라 <glib/glib.h>와 <gdk/gdk.h>와 같이 GTK+가 의존하는 다른 라이브러리로부터의 헤더 파일도 포함한다. 자신의 애플리케이션들 중 대부분은 <gtk/gtk.h>가 GTK+ 개발에 포함시켜야 할 유일한 GTK+ 헤더 파일에 해당하겠지만 좀 더 향상된 애플리케이션의 경우라면 더 많은 대상을 포함시켜야 할 것이다.

리스트 2-1은 GTK+를 이용해 생성할 수 있는 가장 간단한 애플리케이션에 속한다. 이는 기본 너비와 높이가 200 픽셀인 최상위 수준 GtkWindow 위젯을 생성한다. 애플리케이션을 시작한 terminal에서 죽이는(kill) 방법 말고는 애플리케이션을 나갈 방법이 없다. 필요 시 시그널을 이용해 애플리케이션을 나가는 방법은 다음 예제에서 학습할 것이다.

이 예제는 간단한 편이지만 당신이 생성하는 모든 GTK+ 애플리케이션에 필요한 기본사항을 보여준다. "Hello World" 애플리케이션을 이해하기 위한 첫 단계는 main() 함수의 내용을 살펴보는 것이다.

**GTK+ 초기화하기**

GTK+ 라이브러리의 초기화는 대부분 애플리케이션에서 매우 간단하게 이루어진다. gtk\_init()를 호출하면 모든 초기화 작업이 자동으로 실행될 것이다.

GTK+ 환경의 설정부터 시작하는데, 이 과정에는 GDK 디스플레이를 열고 GLib 메인 이벤트 루프와 기본 시그널 처리를 준비하는 과정이 포함된다. gtk\_init()가 필요 이상으로 작업한다면 gdk\_init() 또는 g\_main\_loop\_new()와 같이 적은 수의 함수를 호출하는 작은 initialization 함수를 본인이 직접 생성하는 방법도 택할 수 있으나 대부분의 애플리케이션에서는 그럴 필요가 없다.

오픈 소스 라이브러리를 이용 시 가장 큰 장점 중 하나는 어떻게 일이 진행되는지 코드를 직접 읽을 수 있다는 데에 있다. GTK+ 소스를 손쉽게 확인하여 gtk\_init()가 호출하는 모든 내용을 확인하고 자신의 애플리케이션이 무엇을 실행해야 하는지를 선택할 수 있다. 하지만 각 라이브러리가 어떻게 사용되고 상호 연관되는지에 대해 더 학습하기 전에는 우선 gtk\_init()를 사용하도록 한다.

표준 main() 인자 매개변수인 argc와 argv를 gtk\_init()로 전달하였음을 눈치챘을 것이다. GTK+ 초기화 함수는 모든 인자를 파싱하고 그것이 인식하는 것은 무엇이든 제거한다. 그것이 사용하는 매개변수는 모두 리스트에서 제거될 것이므로 gtk\_init()를 호출한 다음에 인자 파싱은 개발자가 직접 실행해야 한다. 즉, 매개변수의 표준 리스트는 개발자 당사자가 추가로 작업할 필요 없이 모든 GTK+ 애플리케이션이 전달 및 파싱할 수 있다는 뜻이다.

GTK+ 라이브러리에 다른 함수를 호출하기 전에 먼저 gtk\_init()를 호출하는 것이 중요하다. 이를 어길 경우 개발자의 애플리케이션은 적절하게 기능하지 못해 결국 충돌할 가능성이 크다.

gtk\_init() 함수가 만일 GUI를 초기화할 수 없거나 해결할 수 없는 큰 문제에 직면할 경우 애플리케이션을 종료시킬 것이다. GUI 초기화가 실패할 때 본인의 애플리케이션이 텍스트 인터페이스에 의존하길 원한다면 gtk\_init\_check()를 이용해야 한다.

```

gboolean gtk_init_check (int *argc,
                        char ***argv);
    
```

초기화가 실패하면 FALSE가 리턴된다. 성공한다면 `gtk_init_check()`는 TRUE를 리턴할 것이다. 해당 함수는 의지할 텍스트 인터페이스가 있을 경우에만 사용하길 바란다!

위젯 계층구조

위젯 계층구조는 GTK+를 학습할 때 가장 중요한 논의 주제들 중 하나라고 생각한다. 이해하기 힘든 주제는 아니지만 이것이 없다면 오늘날 위젯은 존재할 수 없었을 것이다.

주제를 이해하기 위해 새로운 `GtkWindow` 객체를 생성하는 데에 사용되는 함수 `gtk_window_new()`를 살펴볼 것이다. 아래 행을 통해 우리는 새 `GtkWindow` 객체를 생성하길 원하는데 `gtk_window_new()`는 `GtkWidget`를 향하는 포인터를 리턴함을 눈치챌 것이다. 이는 GTK+ 에서 모든 위젯은 사실상 `GtkWidget` 자체이기 때문이다.

```
GtkWidget* gtk_window_new (GtkWindowType type);
```

GTK+의 위젯은 GObject 계층구조 시스템을 이용하므로, 이미 존재하는 위젯으로부터 새 위젯을 파생하는 것이 가능하다. 자식 위젯은 사실상 그들의 조상(ancestor) 위젯의 implementation이기 때문에 그들의 부모, 조부 위젯 등으로부터 프로퍼티, 함수, 시그널을 상속받는다.

GTK+의 위젯 계층구조는 단일 상속된 시스템으로, 각 자식은 하나의 직계 부모만 가질 수 있다는 의미이다. 이는 모든 위젯이 구현하는 단순한 선형적 관계를 생성한다. 개발자 고유의 자식 위젯을 파생시키는 방법은 제 11장에서 학습할 것이다. 그때까지 우리는 위젯 계층구조를 이용해 상속된 메서드와 프로퍼티를 활용할 것이다.

그림 2-2에는 `GtkWindow` 클래스의 위젯 계층구조에 대한 단순한 개요가 설명되어 있다.

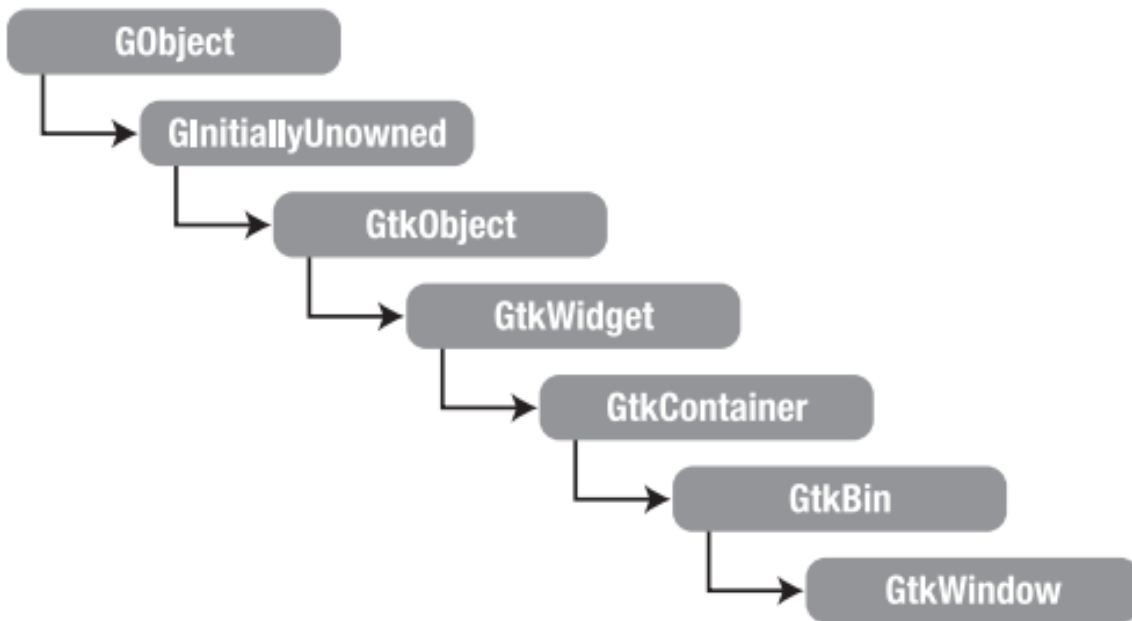


그림 2-1은 처음에는 복잡해 보이지만 이해가 쉽도록 클래스 타입을 하나씩 살펴보도록 하자. GObject는 GTK+와 Pango를 포함해 그것을 기반으로 하는 모든 라이브러리에 대해 공통 속성을 제공하는 기본적인 타입이다. 이는 그로부터 파생된 객체들이 구성, 소멸, 참조, 참조해제되도록 허용한다. 또 시그널 시스템과 객체 프로퍼티 함수를 제공하기도 한다. `G_OBJECT()`를 이용해 객체를 GObject로서 형변환(cast)할 수 있다. GObject가 아닌 객체나 GObject로부터 파생되지 않은 객체를 `G_OBJECT()`로 형변환할 경우 GLib는 임계 오류를 던지고 형변환은 실패할 것이다. 이는 다른 어떤 GTK+ 형변환 함수와도 발생할 것이다.

GInitiallyUnowned는 프로그래머가 접근해선 절대로 안 되는데, 그 모든 멤버들이 private하기 때문이다. 이는 참조가 floating할 수 있도록 존재한다. floating 참조는 누구도 소유하지 않은 참조다.

- GtkWidget는 모든 GTK+ 객체의 기반 클래스이다. GTK+ 2.0에서는 모든 객체의 절대적 기반 클래스로서 대체되었지만 GtkWidget는 GtkAdjustment와 같은 비위젯(nonwidget) 클래스의 이전 기종 호환성을 위해 유

지되었다. GTK\_OBJECT()를 이용해 객체를 GtkWidget로서 형변환할 수 있다.

- GtkWidget은 모든 GTK+ 위젯에 대한 추상적 기반 클래스다. 이는 모든 위젯이 필요로 하는 스타일 프로퍼티와 표준 함수를 소개한다. 모든 위젯을 GtkWidget로서 보관하는 것이 표준 관행이며 이는 리스팅 2-1에서 확인할 수 있다. 따라서 객체의 형변환에 GTK\_WIDGET()을 사용해야 하는 경우는 드물 것이다.
- GtkContainer는 하나 또는 그 이상의 위젯을 포함하는 데에 사용되는 추상적 클래스다. 이것이 없이는 창에 다른 어떤 위젯도 추가할 수가 없기 때문에 매우 중요한 구조다. 따라서 제 3장에서는 이 클래스로부터 파생된 위젯을 중심으로 다루겠다. GTK\_CONTAINER()를 이용해 객체를 GtkContainer로서 형변환할 수 있다.
- GtkBin은 또 다른 추상적 클래스로서, 위젯이 하나의 자식만 포함하도록 허용한다. 이는 코드를 재생성할 필요 없이 다수의 위젯이 해당 기능을 갖도록 허용한다. GTK\_BIN()을 이용해 객체를 GtkBin으로서 형변환할 수 있다.
- GtkWindow는 리스팅 2-1에서 살펴본 표준 window 객체다. GTK\_WINDOW()를 이용해 객체의 형변환이 가능하다.

본문에 실린 모든 위젯은 비슷한 위젯 계층구조를 사용할 것이다. API 문서가 있다면 자신이 사용하는 위젯의 계층구조를 참조할 수 있으므로 유용하다. API 문서를 라이브러리와 함께 설치하지 않았다면 [www.gtk.org/api](http://www.gtk.org/api) 에서 이용할 수 있다.

현재로서는 객체를 어떻게 형변환하고 어떤 기본 추상적 타입이 사용되는지만 알아도 충분하다. 제 11장에서는 자신만의 위젯을 생성하는 방법을 학습할 것이다. 그 때 GObject 시스템이 하는 일도 자세히 알아볼 것이다.

### GTK+ Windows

리스팅 2-1의 코드는 기본 너비와 높이가 200 픽셀로 설정된 GtkWindow 객체를 생성한다. 기본 크기를 200으로 선택한 이유는 너비와 높이가 0 픽셀인 창은 크기 조정이 불가능하기 때문이다. 제목 표시줄과 창 테두리는 총 크기에 포함되어 있으므로 사실상 창의 작업 영역은 200 x 200 픽셀보다 작다는 사실을 주목해야 한다.

GTK\_WINDOW\_TOPLEVEL을 gtk\_window\_new()로 전달하였다. 이는 GTK+에게 새로운 최상위 수준 창을 생성하라고 말한다. 최상위 수준 창은 윈도 관리자 decorations를 사용하고, 테두리 프레임을 가지며, 윈도 관리자가 그들을 위치시킬 수 있도록 허용한다. 따라서 개발자는 창의 위치와 관련해 절대적인 제어를 "갖지 않으며" 그러한 가정도 해선 안 된다.

```
GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

GTK+ 가 제어하는 것과 윈도 관리자가 제어하는 것을 구별하는 것이 중요하다. 최상위 수준 위젯의 크기와 위치에 대한 제안과 요청을 보낼 수는 있다. 하지만 이러한 기능의 최종적 제어는 윈도 관리자에게 있다.

한편 GTK\_WINDOW\_POPUP을 이용해 팝업 창을 생성할 수도 있지만 GTK+에서는 오해의 소지가 있는 이름이다. 팝업 창은 툴팁이나 메뉴와 같이 창으로 간주하지 않는 대상에 사용된다.

팝업 창은 윈도 관리자가 무시하므로 decorations 또는 테두리 프레임이 없다. 팝업 창을 최소화하거나 최대화하는 방법도 없는데, 윈도 관리자가 그에 대해 알지 못하기 때문이다. 크기조정 그립(grip)이 표시되지 않고, 기본 키 바인딩은 작동하지 않을 것이다.

GTK\_WINDOW\_TOPLEVEL과 GTK\_WINDOW\_POPUP은 GtkWidgetType 목록(enumeration)에서 이용 가능한 유일한 요소들이다. 대부분의 경우 타당한 이유가 있지 않는 한 GTK\_WINDOW\_TOPLEVEL을 사용할 것이다.

**Note** 창에서 윈도 관리자 decorations만 끄고자 할 때는 GTK\_WINDOW\_POPUP을 사용해선 안 된다. 창의 decorations를 끄려면 그 대신 gtk\_window\_set\_decorated(GtkWindow \*window, gboolean show)를 사용해야 한다.

아래 함수는 제목 표시줄과 작업표시줄로 "Hello World!"를 창의 제목으로 표시할 것을 요청한다.

gtk\_window\_set\_title()은 GtkWidget 객체를 그 첫 번째 매개변수로서 요하기 때문에 GTK\_WINDOW() 함수를 이용해 창의 형변환을 실행해야 한다.

```
void gtk_window_set_title (GtkWindow *window,
    const gchar *title);
```

gtk\_window\_set\_title()의 두 번째 매개변수는 창이 표시하게 될 제목이다. 이는 gchar이라 불리는 GLib의 char 구현을 이용한다. gchar\*로 열거된 매개변수가 보이던 const char\*도 수용할 것인데, gchar\*는 표준 C 문자열 객체의 typedef로서 정의되기 때문이다.

이번 절에서 마지막으로 살펴볼 흥미로운 함수는 gtk\_widget\_show()인데, 이는 명시된 위젯을 visible로 설정할 것을 GTK+로 알린다. gtk\_widget\_show()를 호출한 직후엔 위젯이 표시되지 않을 수도 있는데, 화면으로 가져오기 전에 GTK+는 모든 처리가 완료될 때까지 위젯을 대기시키기(queue) 때문이다.

gtk\_widget\_show()는 그것이 호출되는 위젯만 표시할 것이라는 사실을 주목하는 것이 중요하다. 위젯이 이미 visible로 설정되지 않은 자식들을 가진 경우 자식들은 화면에 표시되지 않을 것이다. 또 위젯의 부모가 visible로 설정되지 않은 경우 부모는 화면에 표시되지 않을 것이다. 대신 그 부모도 visible로 설정될 때까지 대기할 것이다.

위젯의 표시 뿐만 아니라 gtk\_widget\_hide()를 이용해 사용자의 뷰에서 위젯을 숨기는 것도 가능하다.

```
void gtk_widget_hide (GtkWidget *widget);
```

이는 뷰에서 모든 자식 위젯을 숨기겠지만 주의해서 사용해야 한다. 해당 함수는 명시된 위젯을 hidden으로 설정할 뿐이다. 후에 위젯을 표시하면 그 자식들은 hidden으로 표시된 적이 없기 때문에 visible 상태일 것이다. 다수의 위젯을 한 번에 표시하고 숨기는 방법을 학습 시 이러한 구별이 중요해질 것이다.

### 메인 루프 함수

초기화가 완료되고 필요한 시그널이 GTK+ 애플리케이션에서 연결되면 GTK+ 메인 루프가 제어를 담당하고 이벤트 처리를 시작하길 원할 때가 올 것이다. 이를 위해 gtk\_main()을 호출할 것인데, 해당 호출은 당신이 gtk\_main\_quit() 를 호출하거나 애플리케이션이 종료될 때까지 계속 실행된다. 이것은 main()에서 호출되는 마지막 GTK+ 함수여야 한다.

gtk\_main()을 호출한 후에는 콜백 함수가 초기화되기 전까지는 프로그램의 제어를 다시 얻기가 불가능하다. GTK+에서 시그널과 콜백 함수는 버튼 클릭, 비동기식 입-출력 이벤트, 프로그램 가능 timeout 등의 사용자 액션에 의해 트리거된다. 다음 예제에서는 시그널, 이벤트, 콜백 함수를 살펴봄으로써 시작할 것이다.

**Note** 특정 시간 간격으로 호출되는 함수의 생성도 가능하며 이를 timeout이라 부른다. 콜백 함수의 또 다른 타입으로 idle 함수라는 것이 있는데, 이는 운영체제가 다른 작업의 처리로 바쁘지 않을 때 호출된다. 이러한 두 가지 기능은 GLib의 일부이며 제 6장에서 상세히 다루겠다.

이런 몇 가지 경우를 제외하곤 애플리케이션의 제어는 gtk\_main()이 호출되고 나면 시그널, timeout 함수, 그 외 다양한 콜백 함수에 의해 관리된다. 본 장의 후반부에서는 자신의 애플리케이션에서 시그널과 콜백을 사용하는 방법을 살펴볼 것이다.

## GCC와 pkg-config를 이용해 컴파일하기

이제 리스팅 2-1이 어떻게 작동하는지 이해했으니 코드를 실행 파일(executable)로 컴파일할 때가 되었다. 이를 위해 terminal에서 아래 명령을 실행한다.

```
gcc -Wall -g helloworld.c -o helloworld `pkg-config --cflags gtk+-2.0` \
    `pkg-config --libs gtk+-2.0`
```

해당 명령은 libglade도 필요로 하는 제 10장의 예제들만 제외하고 모든 예제에서 사용 가능하다. 필자는 Linux에서 표준 C 컴파일러인 GCC 컴파일러를 사용하기로 결정했는데, 대부분의 C와 C++ 컴파일러도 작동할 것이다. 단, 다른 컴파일러를 사용할 것이라면 문서의 참조가 필요할 것이다.

앞의 컴파일 명령은 제공되는 여러 옵션을 이용해 과잉된다. `-Wall` 옵션은 모든 타입의 컴파일러 경고를 가능하게 한다. 항상 바람직한 것은 아니지만 GTK+로 프로그래밍을 시작할 때 단순한 프로그래밍 오류를 감지하는 데 도움이 되기도 한다. 디버깅은 `-g`를 이용해 가능하므로, GDB 또는 자신이 선택한 디버거를 이용해 컴파일된 애플리케이션을 사용할 수 있을 것이다.

다음 명령 집합인 `helloworld.c -o helloworld`는 명시된 파일을 컴파일하고 이를 `helloworld`라는 실행 파일로 출력한다. GCC에 의한 컴파일에 대해 하나 또는 그 이상의 소스 파일을 명시할 수 있다.

**Caution** 컴파일 명령에 사용된 기울어진 작은 따옴표 부호는 역따옴표로, 대부분 키보다 상단 좌측 모서리 쪽의 키에서 찾을 수 있다. 이것은 따옴표 사이의 명령은 나머지 행이 실행되기 전에 출력에 의해 실행 및 대체되어야 함을 단말기로 알려준다.

GCC 컴파일러에 더해 명시된 라이브러리나 경로의 리스트를 리턴하는 `pkg-config` 애플리케이션도 사용할 필요가 있다.

첫 번째 예인 `pkg-config --cflags gtk+-2.0`은 컴파일러의 `include` 경로로 디렉터리명을 리턴한다. 이는 컴파일러에서 GTK+ 헤더 파일을 이용 가능하도록 확보할 것이다. 자신의 단말기에서 `pkg-config --cflags gtk+-2.0`을 실행하면 컴파일러로 출력되는 내용의 예를 살펴볼 수 있을 것이다.

두 번째 호출인 `pkg-config --libs gtk+-2.0`은 링커가 사용하는 명령행으로 옵션을 추가하는데, 실행 파일로 연결하는 데에 필요한 라이브러리 리스트와 디렉터리 경로 확장자가 포함된다. 표준 Linux 환경에서 리턴되는 라이브러리는 아래와 같다.

- `GTK+(-lgtk)`: 그래픽 위젯
- `GDK(-lgdk)`: 표준 그래픽 렌더링 라이브러리
- `GdkPixbuf(-lgdk_pixbuf)`: 클라이언트 측 이미지 조각
- `Pango(-lpango)`: 글꼴 렌더링과 출력
- `GObject(-lgobject)`: 객체 지향의 타입 시스템
- `GModule(-lgmodule)`: 동적으로 로딩되는 라이브러리
- `GLib(-lglib)`: 데이터 타입과 유틸리티 함수
- `Xlib(-lX11)`: X Windows System 프로토콜 라이브러리
- `Xext(-lXext)`: X 확장 라이브러리 루틴
- `GNU math library(-lm)`: GNU 라이브러리로서, GTK+는 여기서 비롯된 다수의 루틴을 사용

위에서 확인할 수 있듯이 `pkg-config`는 개발자가 수동으로 GTK+ 애플리케이션을 컴파일할 때마다 `includes` 및 라이브러리 리스트의 하드코딩을 피하는 편리한 방법을 제공한다.

리스트 2-1은 GTK+를 이용해 생성할 수 있는 가장 간단한 애플리케이션에 해당한다. 그림 2-2와 같이 너비와 높이가 200 픽셀인 최상위 수준의 `GtkWindow` 위젯이 생성된다.



창은 제목 표시줄의 오른쪽에 표준 X를 포함하고 있지만 X를 클릭 시 창이 사라지는 효과만 나타남을 눈치챌 것이다. 애플리케이션은 계속해서 이벤트를 기다리며, `Ctrl+C`를 누를 때까지 제어는 시작 단말기(launching terminal)로 돌아가지 않을 것이다. 시그널을 이용해 `shutdown` 콜백을 구현하는 방법을 다음 예제에서 살펴볼 것이다.

## "Hello World" 확장하기

개발자가 작성하는 모든 GTK+ 애플리케이션은 리스팅 2-1에 표시된 함수 호출을 필요로 하지만 예제 자체는 그다지 유용하지 못하다. 이제 시작하는 방법을 알았으니 좀 더 멋지게 세상에 "hello"를 외쳐보자.

리스팅 2-2는 "Hello World" 애플리케이션을 두 가지 방법으로 확장한다. 첫째, 콜백 함수를 윈도우 시그널로 연결하여 애플리케이션이 스스로 종료할 수 있다. 둘째, 이 예제는 GtkContainer 구조를 도입하여 위젯이 하나 또는 이상의 다른 위젯을 포함할 수 있도록 해준다.

리스팅 2-2. World 와 다시 인사하기 (helloworld2.c)

```
#include <gtk/gtk.h>

static void destroy (GtkWidget*, gpointer);
static gboolean delete_event (GtkWidget*, GdkEvent*, gpointer);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Hello World!");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 200, 100);

    /* Connect the main window to the destroy and delete-event signals.
    */
    g_signal_connect (G_OBJECT (window), "destroy",
                     G_CALLBACK (destroy), NULL);
    g_signal_connect (G_OBJECT (window), "delete_event",
                     G_CALLBACK (delete_event), NULL);

    /* Create a new GtkLabel widget that is selectable. */
    label = gtk_label_new ("Hello World");
    gtk_label_set_selectable (GTK_LABEL (label), TRUE);

    /* Add the label as a child widget of the window. */
    gtk_container_add (GTK_CONTAINER (window), label);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* Stop the GTK+ main loop function when the window is destroyed. */
static void
destroy (GtkWidget *window,
```

```

        gpointer data)
{
    gtk_main_quit ();
}

/* Return FALSE to destroy the widget. By returning TRUE, you can
cancel
 * a delete-event. This can be used to confirm quitting the
application. */
static gboolean
delete_event (GtkWidget *window,
              GdkEvent *event,
              gpointer data)
{
    return FALSE;
}

```

그림 2-3은 리스팅 2-2이 실행되는 모습의 스크린샷이다. 이는 GtkWidget가 포함하는 GtkLabel을 표시한다. 이제 해당 예제에서 표시된 새 기능을 살펴보자.



### GtkLabel 위젯

리스팅 2-2에서 GtkLabel이라 불리는 새로운 타입의 위젯이 생성되었다. 이름이 암시하듯 GtkLabel 위젯은 보통 다른 위젯에 라벨을 붙이는 데에 사용된다. 하지만 편집 불가하거나, 포매팅되거나, 래핑된 텍스트의 큰 블록을 생성하는 등의 작업에 사용되기도 한다.

gtk\_label\_new()를 호출함으로써 새 라벨 위젯을 생성할 수도 있다. gtk\_label\_new()에 NULL을 전달하는 것은 빈 문자열을 전달하는 것과 동일하다. 이런 경우 어떠한 텍스트도 없이 라벨만 표시될 것이다.

```
GtkWidget* gtk_label_new (const gchar *str);
```

사용자가 키보드나 마우스를 이용해 일반적인 GtkLabel을 편집하기란 (즉, 프로그래머의 추가 작업 없이 편집하기란) 불가능하지만 gtk\_label\_set\_selectable()을 이용하면 사용자가 텍스트를 선택하고 복사할 수 있도록 해줄 것이다. 위젯은 커서 포커스를 허용 가능하므로, 라벨과 다른 위젯들 간에 Tab 키를 이용해 이동할 수 있다.

```
void gtk_label_set_selectable (GtkLabel *label,
                              gboolean selectable);
```

라벨을 선택하는 기능은 기본적으로 꺼져 있는데, 이러한 기능은 사용자가 특정 정보를 계속 유지해야 할 필요가 있을 때에만 사용되어야 하기 때문이다. 가령 오류 메시지는 selectable(선택 가능하게) 설정되어야만 웹 브라우저와 같은 다른 애플리케이션으로 쉽게 복사가 가능하다.

GtkLabel의 텍스트는 당신이 생성 시 명시한 텍스트 문자열과 동일한 상태로 유지될 필요가 없다.

gtk\_label\_set\_text()를 이용해 쉽게 변경할 수 있다. 연상기호(mnemonics)는 물론이고 라벨이 최근에 포함한 텍스트는 무엇이든 오버라이드될 것이다.



```
void gtk_label_set_text (GtkLabel *label,
                        const gchar *str);
```

■ **Note** 연상기호는 사용자가 누르면 특정 유형의 액션을 실행하는 키의 조합이다. GtkLabel에 연상기호를 추가하면 키를 누를 시 지정된 위젯의 활성화가 가능하다.

현재 라벨에 의해 표시된 문자열은 `gtk_label_get_text()`를 이용해 검색할 수 있다. 리턴된 문자열은 markup이나 연상기호에 대한 정보는 포함하지 않을 것이다. 라벨은 또 이를 내부적으로 사용하기 때문에 리턴된 문자열은 절대 수정해선 안 된다!

마지막으로 알아야 하는 GtkLabel 메서드는 `gtk_label_set_markup()`으로, 표시된 텍스트에 대한 커스텀 스타일을 정의하도록 해준다. Pango Text Markup Language가 제공하는 태그의 수는 많은데, 본 서적의 뒷부분에 실린 부록 C에서 찾을 수 있겠다.

```
void gtk_label_set_markup (GtkLabel *label,
                          const gchar *str);
```

Pango Text Markup Language는 두 가지 타입의 스타일 메서드를 제공한다. `<span>` 태그는 글꼴 타입, 크기, 선 굵기, 전경색 등과 같은 속성들과 함께 사용 가능하다. 또 감싼 텍스트를 볼드체, 고정폭 간격, 이탤릭체로 만드는 `<b>`, `<tt>`, `<i>`와 같은 다양한 태그도 제공한다.

### 컨테이너 위젯과 레이아웃

이번 장에서 처음으로 소개한 예제를 다시 상기하면, GtkWindow 구조는 GtkContainer로부터 간접적으로 파생된다. 즉, GtkWindow는 GtkContainer이고, GtkContainer의 함수, 시그널, 프로퍼티를 모두 상속받는다.

`gtk_container_add()`를 이용하면 위젯을 컨테이너의 자식으로서 추가할 수 있다. 컨테이너가 이제 위젯의 부모가 되는 것이다. 해당 컨테이너, 그리고 포함된 관계를 설명하는 데에 "부모와 자식"이란 용어가 자주 사용되는데, 부모는 포함하는 위젯이고 자식은 부모 안에 포함된다.

```
void gtk_container_add (GtkContainer *container,
                       GtkWidget *child);
```

아쉽게도 이러한 용어는 오해를 불러일으키곤 하는데 그 이유는 GTK+가 모든 면에서 객체 지향적이기 때문이다. 이러한 이유로 인해 GTK+를 사용하고 논할 때는 "부모"와 "자식"이 사용되는 맥락을 인지해야만 한다. 해당 용어들은 컨테이너 위젯 관계를 이야기할 때와 위젯 파생(derivation) 관계를 논할 때 모두 사용된다.

GtkContainer 클래스의 목적은 부모 위젯이 하나 또는 그 이상의 자식을 갖도록 허용하는 데에 있다.

GtkWindow는 GtkBin이라 불리는 컨테이너 타입에서 파생된다. GtkBin은 부모가 하나의 자식만 갖도록 한다. 따라서 컨테이너로서 창은 직접적으로 하나의 자식만 포함하도록 제한한다. 다행히 하나의 자식은 그 자체가 조금 더 복잡한 컨테이너 위젯이 될 수 있으므로 또 하나 이상의 자식 위젯을 포함할 수도 있다.

예제로 소개한 창은 더 이상 기본 크기 200 x 200 픽셀이 아니며 사각형 비율(square aspect ratio)이 유지되지 않음을 주목해야 한다. 이것은 GTK+가 보통 자동적이고 다이내믹하게 크기가 조정되는 레이아웃 시스템을 사용하기 때문이다. 이렇게 다이내믹한 크기 조정의 원인은 컨테이너 객체 존재의 이면에서 발생한다. 크기 조정 체계는 컨테이너 위젯을 다루는 다음 장에서 세부적으로 논할 것이다.

우리가 생성한 창은 GtkContainer이므로 `gtk_container_set_border_width()` 함수를 이용해 창의 안쪽 테두리 주위로 10 픽셀 테두리를 위치시킬 수 있다. 테두리는 자식 위젯의 사면에 모두 설정된다.

```
void gtk_container_set_border_width (GtkContainer *container,
                                     guint border_width);
```

레이아웃 관리자는 테두리를 추가할 필요 없이 GtkLabel 위젯의 기본 크기로 창을 축소시킬 수 있도록 해준다. 리스팅 2-1에서 창은 200 픽셀 너비와 100 픽셀 높이로 설정되었다. 이 크기라면 대부분의 시스템에서 라벨 주위에 10 픽셀 이상의 테두리가 표시될 것이다. 테두리는 사용자로 하여금 테두리와 위젯이 할당된 크기보

다 작게 창을 조정하지 못하도록 한다.

그리고 나서 창에 `gtk_widget_show_all()`을 호출한다. 해당 함수는 창, 그의 자식, 자식의 자식 등을 재귀적으로 그린다. 해당 함수가 없다면 자식 위젯마다 `gtk_widget_show()`를 호출해야 할 것이다. 대신 `gtk_widget_show_all()`을 이용하면 GTK+는 화면에 각 위젯이 모두 나타날 때까지 하나씩 표시하는 일을 알아서 해준다.

```
void gtk_widget_show_all (GtkWidget *widget);
```

비재귀적 `gtk_widget_show()`와 같이, 부모가 `visible`로 설정되지 않은 위젯에서 위의 함수를 호출할 경우 표시되지 않을 것이다. 위젯은 그 부모가 `visible`로 설정될 때까지 대기할 것이다.

GTK+는 `gtk_widget_hide_all()`을 제공하기도 하는데, 이는 명시된 위젯과 그 모든 자식을 숨겨진 상태로 설정할 것이다. 컨테이너가 숨겨지면 포함된 위젯들도 표시되지 않으므로, 포함하는 객체에서 `gtk_widget_hide()`를 호출하면 `gtk_widget_hide_all()`을 호출할 때와 동일한 효과를 보이는데, 둘 다 컨테이너와 그 자식들을 모두 숨길 것이기 때문이다. 하지만 둘 사이에는 중요한 차이점이 있다. `gtk_widget_hide()`를 호출하면 하나의 위젯에만 `visible` 프로퍼티를 `FALSE`로 설정하는 반면 `gtk_widget_hide_all()`을 호출하면 전달된 위젯 상의 프로퍼티를 비롯해 그에 포함된 모든 위젯에서 프로퍼티를 재귀적으로 변경한다.

```
void gtk_widget_hide_all (GtkWidget *widget);
```

`gtk_widget_show()`와 `gtk_widget_show_all()` 함수 집합은 같은 관계를 갖는다. 따라서 같은 위젯에서 `gtk_widget_hide_all()`을 사용하되 `gtk_widget_show()`를 호출할 경우 그 자식들은 모두 `invisible` 상태로 유지될 것이다.

컨테이너 위젯과 애플리케이션 레이아웃 관리는 다음 장에서 상세히 다룰 것이다. `GtkContainer`를 이해하는데 필요한 정보는 리스팅 2-2에서 충분히 실고 있으니 시그널과 콜백 함수를 계속하겠다.

### 시그널과 콜백

GTK+는 시그널과 콜백 함수에 의존하는 시스템이다. 시그널은 사용자가 어떤 액션을 실행했다는 사실을 개발자의 애플리케이션에게 전하는 알림이다. 개발자는 시그널이 방출되면 함수를 실행하도록 GTK+에게 알릴 수 있다. 이를 콜백 함수라고 부른다.

**Caution** GTK+ 시그널은 POSIX 시그널과 동일하지 않다! GTK+의 시그널은 X Windows System으로부터 이벤트에 의해 전파된다. 각 시그널은 별개의 메서드를 제공하고, 이러한 두 가지 타입의 시그널은 교대(*interchangeably*)해서 사용해선 안 된다.

사용자 인터페이스를 초기화하고 나면 제어는 `gtk_main()` 함수로 주어지고, 해당 함수는 시그널이 방출될 때까지 `sleep` 상태로 유지된다. 이 시점에서 제어는 콜백 함수라고 불리는 다른 함수로 전달된다.

프로그래머로서 당신은 `gtk_main()`을 호출하기 전에 그들의 콜백 함수로 시그널을 연결한다. 콜백 함수는 액션이 발생하여 시그널이 방출될 때나, 개발자가 명시적으로 시그널을 방출할 때 호출될 것이다. 시그널이 절대 방출되지 못하도록 막는 기능도 있다.

**Note** 자신의 애플리케이션에서 시그널은 언제든 연결할 수 있다. 예를 들어, 새로운 시그널은 콜백 함수 내부에서 연결 가능하다. 하지만 `gtk_main()`을 호출하기 전에 임무에 결정적인(*mission-critical*) 콜백을 초기화하도록 한다.

함수와 마찬가지로 시그널에도 많은 유형이 있으며, 부모 구조로부터 상속된다. `hide` 또는 `grab-focus` 등 모든 위젯에 일반적인 시그널도 많이 존재하며, `GtkRadioButton` 시그널 `group-changed`와 같이 위젯에 특정한 시그널도 있다. 어떤 경우든 클래스로부터 파생된 위젯은 그의 모든 조상이 이용할 수 있는 시그널이라면 그들도 이용할 수 있다.

시그널 연결하기

처음으로 직면하게 될 시그널의 예제는 리스팅 2-2에 소개되어 있다. GtkWidget는 destroy() 콜백 함수로 연결되어 있었다. 해당 함수는 destroy 시그널이 방출될 때 호출될 것이다.

```
g_signal_connect (G_OBJECT (window), "destroy",
                 G_CALLBACK (destroy), NULL);
```

GTK+는 위젯에 gtk\_widget\_destroy()가 호출되거나 delete\_event() 콜백 함수로부터 FALSE가 리턴되면 destroy 시그널을 방출한다. API 문서를 참조한다면 destroy 시그널이 GObject 클래스에 속함을 확인할 수 있을 것이다. 즉, GTK+ 내 모든 클래스는 시그널을 상속받고, 개발자는 어떤 GTK+ 구조든 소멸되면 그러한 사실을 통지받을 것이다.

모든 g\_signal\_connect() 호출에는 4개의 매개변수가 있다. 첫 번째는 시그널을 감시해야 하는 위젯이다. 그 다음 본인이 추적하길 원하는 시그널의 이름을 명시한다. 각 위젯마다 가능한 시그널이 여러 개 존재하는데, 모두 API 문서에서 찾을 수 있다. 각 위젯은 구조적으로 사실상 각 조상들의 구현에 해당하기 때문에 위젯은 그 조상들의 시그널을 마음대로 사용할 수 있음을 기억하라. 부모 클래스를 참조하려면 API에서 "객체 계층구조(Object Hierarchy)" 부분을 이용할 수 있겠다.

```
gulong g_signal_connect (gpointer object,
                        const gchar *signal_name,
                        GCallback handler,
                        gpointer data);
```

시그널명을 입력할 때 밑줄과 대시 문자는 서로 바꿔 사용할 수 있다. 해당 문자들은 동일한 문자로 파싱될 것이기 때문에 둘 중 무엇을 이용하는지는 별 차이가 없다. 필자는 본문에서 밑줄 문자로 통일해 사용할 것이다.

g\_signal\_connect()에서 세 번째 매개변수는 시그널이 방출되면 호출될 콜백 함수로서, G\_CALLBACK()을 이용해 형변환된다. 콜백 함수의 포맷은 각 구체적인 시그널의 함수 프로타 타입 요구조건에 따라 좌우된다. 콜백 함수의 예는 다음 절에서 소개할 것이다.

g\_signal\_connect()에서 마지막 매개변수는 콜백 함수로 포인터를 전송하도록 해준다. 리스팅 2-2에서 우리는 NULL을 전달하였기 때문에 포인터가 void였지만, 콜백 함수로 GtkWidget를 전달하는 상황을 가정해보자.

g\_signal\_connect()의 예제에서 GtkWidget는 gpointer로서 형변환되어 후에 콜백 함수로 전달될 것이다. gpointer는 그저 void 포인터의 타입 정의에 해당한다. 이는 콜백 함수에서 다시 형변환(recast)할 수 있지만 g\_signal\_connect()는 gpointer 타입을 필요로 한다.

```
g_signal_connect (G_OBJECT (window), "destroy",
                 G_CALLBACK (destroy),
                 (gpointer) label);
```

g\_signal\_connect()에 대한 리턴값은 시그널의 핸들러 식별자이다. 이는 g\_signal\_handler\_block(), g\_signal\_handler\_unblock(), g\_signal\_handler\_disconnect()와 함께 사용 가능하다. 이러한 함수들은 각각 콜백 함수가 호출되지 못하도록 막고, 콜백 함수를 재활성화하며, 메모리로부터 시그널 핸들러를 제거한다. 이에 관한 추가 정보는 API 문서에서 찾을 수 있다.



### 이벤트

이벤트는 X Windows System에 의해 방출되는 특수 타입의 시그널이다. 이러한 시그널들은 처음에 X Window System에 의해 방출되고, 이후 윈도 관리자로부터 개발자의 애플리케이션으로 전송되어 GLib이 제공하는 시그널 시스템에 의해 해석된다. 예를 들어 destroy 시그널은 위젯에서 방출되지만 delete-event 이벤트는 위젯의 기본이 되는 GdkWindow에서 먼저 인식한 후 위젯의 시그널로서 방출된다.

처음으로 만나게 될 이벤트의 예는 리스팅 2-2의 delete-event이다. delete-event 시그널은 사용자가 창 닫기를 시도할 때 방출된다. 창은 제목 표시줄에 위치한 닫기 버튼을 클릭하거나, 작업표시줄에서 팝업 메뉴 항목을 이용하거나, 윈도 관리자가 제공하는 다른 방법들을 이용해 닫을 수 있다.

이벤트를 콜백 함수로 연결하는 것은 다른 GTK+ 시그널과 마찬가지로 g\_signal\_connect()의 사용을 통해 이루어진다. 하지만 당신의 콜백 함수는 약간 다른 방식으로 준비할 것이다.

```
static gboolean
callback_function (GtkWidget *widget,
                  GdkEvent *event,
                  gpointer data);
```

콜백 함수에서 첫 번째 차이점은 gboolean 리턴값이다. 이벤트 콜백으로부터 TRUE가 리턴되면 GTK+는 이벤트가 이미 처리되었으며 계속되지 않을 것이라고 가정한다. FALSE를 리턴하면 개발자는 GTK+에게 계속해서 이벤트를 처리하라고 알리는 셈이다. FALSE는 함수에 대한 기본값이므로 대부분의 경우 delete-event 시그널을 사용할 필요가 없다. 이는 기본 시그널 핸들러를 오버라이드하고자 할 경우에만 유용하다.

예를 들어, 많은 애플리케이션에서 당신은 프로그램이 꺼졌는지 확인하길 원할 것이다. 아래 코드를 이용하면 사용자가 애플리케이션을 끄고 싶지 않을 경우 애플리케이션의 꺼짐을 방지할 수 있다.

```
static gboolean
delete_event (GtkWidget *window,
              GdkEvent *event,
              gpointer data)
{
    gboolean answer = /* Ask the user if exiting is desired. */

    if (answer)
        return FALSE;
    else
        return TRUE;
}
```

delete-event 콜백 함수에서 FALSE를 리턴하면 gtk\_widget\_destroy()가 위젯에서 자동으로 호출된다. 앞에서 언급한 바와 같이 이 시그널은 자동으로 액션을 계속할 것이므로, 기본값의 오버라이드를 원치 않는 한 따로 연결할 필요가 없다.

뿐만 아니라 콜백 함수는 GdkEvent 매개변수를 포함한다. GdkEvent는 GdkEventType 목록과 이용 가능한 모든 이벤트 구조에 대한 C 공용체(union)다. 우선 GdkEventType 목록을 살펴보자.

이벤트 타입

GdkEventType 목록은 이용할 수 있는 이벤트 타입의 리스트를 제공한다. 무슨 일이 일어났는지 항상 알 수는 없기 때문에 이러한 이벤트 타입을 이용하면 발생한 이벤트의 타입을 알아낼 수 있다.

가령 button-press-event 시그널을 위젯으로 연결할 경우 시그널의 콜백 함수를 실행시킬 수 있는 이벤트 타입에는 GDK\_BUTTON\_PRESS, GDK\_2BUTTON\_PRESS, GDK\_3BUTTON\_PRESS 세 가지가 있다. 이를 두 번 또는 세 번 클릭하면 GDK\_BUTTON\_PRESS를 두 번째 이벤트로서 방출하므로 이벤트 타입을 구별할 줄 알아야 한다.

부록 B에서는 개발자가 이용 가능한 이벤트의 전체 리스트를 확인할 수 있다. 이는 g\_signal\_connect()로 전달된 시그널명, GdkEventType 목록 값, 이벤트의 설명을 보여준다.

리스트 2-2에 실린 delete-event 콜백 함수를 살펴보자. delete-event가 GDK\_DELETE 타입이라는 사실을 이미 알고 있지만 잠시만 모른다고 가정하자. 이는 아래 조건문을 이용해 간단히 검사할 수 있다.

```
static gboolean
delete_event (GtkWidget *window,
             GdkEvent *event,
             gpointer data)
{
    if (event->type == GDK_DELETE)
        return FALSE;

    return TRUE;
}
```

위의 예제에서 이벤트 타입이 GDK\_DELETE라면 FALSE가 리턴되고 gtk\_widget\_destroy()가 위젯 상에서 호출될 것이다. 그 외의 경우 TRUE가 리턴되고 추가 액션은 취하지 않는다.

특정 이벤트 구조 이용하기

때로는 어떤 이벤트 타입이 방출되었는지 이미 알고 있는 경우가 있다. 아래 예제에서는 key-press-event가 항상 방출될 것을 알고 있다.

```
g_signal_connect (G_OBJECT (widget), "key-press-event"
                G_CALLBACK (key_press), NULL);
```

이런 경우 이벤트 타입은 항상 GDK\_KEY\_PRESS가 될 것이며 콜백 함수도 그와 같이 선언될 수 있다고 가정하는 것이 안정하겠다.

```
static gboolean
key_press (GtkWidget *widget,
          GdkEventKey *event,
          gpointer data)
```

이벤트 타입이 GDK\_KEY\_PRESS라는 사실을 이미 알고 있기 때문에 GdkEvent 내의 모든 구조로 접근할 필요가 없을 것이다. 콜백 함수에서 GdkEvent 를 대신해 사용 가능한 GdkEventKey 만 이용할 수 있을 것이다. 이벤트는 이미 GdkEventKey로서 형변환되었으므로 해당 구조 내의 요소로만 직접 접근이 가능할 것이다.

```
typedef struct
{
    GdkEventType type; // GDK_KEY_PRESS or GDK_KEY_RELEASE
    GdkWindow *window; // The window that received the event
    gint8 send_event; // TRUE if the event used XSendEvent
```

```

guint32 time; // The length of the event in milliseconds
guint state; // The state of Control, Shift, and Alt
guint keyval; // The key that was pressed <gdk/gdkkeysyms.h>
gint length; // The length of string
gchar *string; // A string approximating the entered text
guint16 hardware_keycode; // Raw code of the key that was pressed
or released
guint8 group; // The keyboard group
guint is_modifier : 1; // Whether hardware_keycode was mapped
(since 2.10)
} GdkEventKey;
    
```

GdkEventKey 구조에는 유용한 프로퍼티가 많이 존재하며 본 저서에 걸쳐 사용될 것이다. 어느 시점에서는 API 문서에서 GdkEvent 구조를 어느 정도 살펴보는 것이 유용할 것이다. 본문에서는 GdkEventKey와 GdkEventButton을 포함해 가장 중요한 구조를 몇 가지 다룰 것이다.

모든 이벤트 구조에서 이용 가능한 유일한 변수는 이벤트 타입으로서, 이는 발생한 이벤트의 타입을 정의한다. 올바르게 않은 방식으로 처리되는 것을 막기 위해서는 항상 이벤트 타입을 확인하는 것이 좋겠다.

### 그 외 GTK+ 함수

더 많은 예제를 살펴보기 전에 다른 장들을 학습하거나 자신만의 GTK+ 애플리케이션을 설치할 때 편리하게 사용할 수 있는 함수를 몇 가지 집중해서 살펴보고자 한다.

#### GtkWidget 함수

GtkWidget 구조는 어떤 위젯과도 사용 가능한 유용한 함수를 많이 포함한다. 이번 절에서는 개발자의 수많은 애플리케이션에서 필요로 하게 될 몇 가지 함수를 간략하게 살펴보겠다.

객체 상에서 gtk\_widget\_destroy()를 명시적으로 호출하면 위젯을 소멸시키는 것이 가능하다.

gtk\_widget\_destroy()를 호출하면 참조 계수(reference count)를 위젯과 그 모든 자식들에게 재귀적으로 떨어뜨린다(drop on). 따라서 위젯과 그 자식들은 소멸되고, 모든 메모리는 해제된다.

```

void gtk_widget_destroy (GtkWidget *widget);
    
```

일반적으로 위의 함수는 최상위 위젯에서만 호출된다. 보통은 대화창을 소멸시키고, 애플리케이션을 중단하는 메뉴 항목을 구현할 때에만 사용된다. 본 장에서 다음으로 소개할 예제에서는 버튼을 클릭하면 애플리케이션이 중단되도록 사용하였다.

위젯의 최소 크기를 설정 시에는 gtk\_widget\_set\_size\_request()를 이용할 수 있다. 이는 위젯을 강제로 정상 크기보다 작거나 크게 만들 것이다. 하지만 기능할 수 없거나 화면에 그려지지 않을 정도로 작게 조정되지는 않을 것이다.

```

void gtk_widget_set_size_request (GtkWidget *widget,
    gint width,
    gint height);
    
```

어떤 매개변수로든 -1를 전달하면 개발자는 GTK+에게 위젯의 본래(natural) 크기를 사용하도록 알린다는 뜻이며, 개발자가 만일 커스텀 크기를 정의하지 않았다면 일반적으로 위젯에 할당되는 크기를 사용하도록 알리는 것이다. 위젯을 본래 크기로 리셋할 수도 있다.

위젯의 높이나 너비는 1 픽셀 미만으로 설정할 수 없지만 매개변수 중 하나로 0을 전달할 경우 GTK+는 위젯을 가능한 한 작게 만들 것이다. 다시 말하지만, 기능을 할 수 없거나 화면에 그려지지 않을 정도로 작게 조정되진 않을 것이다.

국제화로 인해 위젯의 크기를 설정 시 위험이 발생한다. 개발자의 컴퓨터에선 텍스트가 너무 크게 보이는 반면 애플리케이션에 독일어 번역을 사용하는 컴퓨터에서는 텍스트에 비해 위젯이 너무 작거나 크게 나타날 수도 있다. 테마 또한 위젯의 크기 조정과 관련된 문제를 제시하는데, 위젯은 테마에 따라 서로 다른 크기로 기본 설정되기 때문이다. 따라서 대부분의 경우는 GTK+가 창과 위젯의 크기를 선택하도록 허용하는 방법이 최선이겠다.

위젯이 강제로 키보드 포커스를 잡도록 하기 위해서는 `gtk_widget_grab_focus()`를 이용할 수 있다. 이는 키보드 상호작용을 처리할 수 있는 위젯 상에서만 작동할 것이다. `gtk_widget_grab_focus()`의 사용 예로, 검색 툴바가 Firefox에 나타날 때 텍스트 엔트리로 커서를 전송하는 경우를 들 수 있다. 선택 가능한 `GtkLabel`에 포커스를 주는 데에도 사용될 수 있겠다.

```
void gtk_widget_grab_focus (GtkWidget *widget);
```

종종 우리는 위젯을 비활성화(`inactive`)로 설정하길 원할 것이다. `gtk_widget_set_sensitive()`를 호출하면 명시된 위젯과 그 자식들이 비활성화/활성화된다. 위젯을 비활성화로 설정하면 사용자는 위젯과 상호작용할 수 없을 것이다. 대부분의 위젯은 비활성화로 설정 시 회색으로 표시될 것이다.

```
void gtk_widget_set_sensitive (GtkWidget *widget,
                               gboolean sensitive);
```

위젯과 그 자식들을 재활성화시키고 싶다면 동일한 위젯 상에서 해당 함수를 호출하기만 하면 된다. 자식들은 그 부모 위젯의 감도(`sensitivity`)의 영향을 받지만 자신들의 프로퍼티를 변경하는 대신 부모의 설정을 반영할 뿐이다.

**GtkWindow 함수**

`GtkWindow` 구조의 두 가지 사용 예를 살펴보았다. 그리고 창의 안쪽 테두리와 창의 자식 사이에 테두리 패딩(`border padding`)을 추가하는 방법도 학습하였다. 뿐만 아니라 창의 제목을 설정하고 자식 위젯을 추가하는 방법도 배웠다. 이제 창의 맞춤설정(`customize`)을 가능하게 하는 함수를 몇 가지 살펴보도록 한다.

모든 창은 기본적으로 크기 조정이 가능하다. 대부분의 애플리케이션에서는 바람직한 일인데, 각 사용자가 선호하는 크기가 다를 것이기 때문이다. 하지만 특정한 이유가 있다면 사용자가 창의 크기를 조정하지 못하도록 `gtk_window_set_resizable()`을 이용할 수 있겠다.

```
void gtk_window_set_resizable (GtkWindow *window,
                               gboolean resizable);
```

**Caution** 크기 조정의 기능은 윈도 관리자가 제어하므로 모든 경우에서 환영받는 일은 아니라는 사실을 주목해야 한다.

바로 위에 실린 주의사항은 요점을 상기시킨다. GTK+가 하는 일은 대부분 윈도 관리자가 제공한 기능과 상호작용하는 일이다. 이 때문에 윈도우 설정이라고 해서 모든 윈도 관리자에서 따르려는 법은 없다. 개발자의 설정은 힌트에 불과하므로 이후에 사용되거나 무시되거나 둘 중 하나에 해당하기 때문이다. GTK+로 애플리케이션을 개발 시에는 개발자의 요청이 수락되기도, 거부되기도 한다는 점을 유념해야 한다.

`GtkWindow`의 기본 크기는 `gtk_window_set_default_size()`로 설정 가능하지만 해당 함수를 이용 시에는 조심해야 할 점이 몇 가지 있다. 창의 최소 크기가 당신이 명시한 크기보다 큰 경우 GTK+는 해당 함수를 무시할 것이다. 개발자가 이전에 더 큰 크기의 요청을 설정한 경우에도 무시될 것이다.

```
void gtk_window_set_default_size (GtkWindow *window,
                                  gint width,
                                  gint height);
```

`gtk_widget_set_size_request()`와 달리 `gtk_window_set_default_size()`는 창의 처음 크기만 설정할 뿐, 사용자가 크기를 더 크게 또는 작게 조정할 수는 없게 만든다. 높이 또는 너비 매개변수를 0으로 설정할 경우 창의 높이



나 너비가 가능한 최소 크기로 설정될 것이다. 매개변수 중 하나로 -1을 전달할 경우 창은 본래 크기로 설정될 것이다.

gtk\_window\_move()를 이용하면 윈도 관리자에게 창을 명시된 위치로 이동시킬 것을 "요청"(request)할 수 있다. 하지만 윈도 관리자는 이러한 요청을 무시하기도 한다. 윈도 관리자로부터 액션을 요하는 "요청(request)" 함수라면 모두 마찬가지다.

```
void gtk_window_move (GtkWindow *window,
    gint x,
    gint y);
```

화면에서 창의 크기는 기본적으로 화면의 좌측 상단 모서리를 기준으로 계산되지만 gtk\_window\_set\_gravity()를 이용하면 기준을 변경할 수 있다.

```
void gtk_window_set_gravity (GtkWindow *window,
    GdkGravity gravity);
```

해당 함수는 위젯의 gravity를 정의하는데 이는 레이아웃 계산이 (0,0)을 고려하게 될 것이라는 의미다.

GdkGravity 목록에 가능한 값으로는 GDK\_GRAVITY\_NORTH\_WEST, GDK\_GRAVITY\_NORTH, GDK\_GRAVITY\_NORTH\_EAST, GDK\_GRAVITY\_WEST, GDK\_GRAVITY\_CENTER, GDK\_GRAVITY\_EAST, GDK\_GRAVITY\_SOUTH\_WEST, GDK\_GRAVITY\_SOUTH, GDK\_GRAVITY\_SOUTH\_EAST, GDK\_GRAVITY\_STATIC가 있다.

North, south, east, west는 화면에서 상단, 하단, 우측, 좌측의 변(edge)을 나타낸다. 이들은 다수의 gravity 타입을 구성하는 데에 사용된다. GDK\_GRAVITY\_STATIC은 창 자체의 좌측 상단 모서리를 나타내며 창 decorations는 무시한다.

애플리케이션이 하나 이상의 창을 가진 경우 gtk\_window\_set\_transient\_for()를 이용해 하나의 창을 부모로 설정할 수 있다. 이로 인해 윈도 관리자는 부모 위젯 위에 자식을 중앙 정렬하거나 하나의 특정 창을 다른 창들 보다 항상 위에 표시하게 만들 수 있다. 다중 창이나 일시적(transient) 관계에 대한 개념은 대화상자를 논하면서 제 5장에서 살펴볼 것이다.

```
void gtk_window_set_transient_for (GtkWindow *window,
    GtkWindow *parent);
```

창의 작업 표시줄이나 제목 표시줄에 표시될 아이콘은 gtk\_window\_set\_icon\_from\_file()을 이용해 설정할 수 있다. 아이콘의 크기는 중요하지 않은데, 알맞은 크기가 알려지면 조정될 것이기 때문이다. 그래야만 최상의 질로 크기가 조정된 아이콘이 가능해진다.

```
gboolean gtk_window_set_icon_from_file (GtkWindow *window,
    const gchar *filename,
    GError **err); // NULL
```

아이콘이 성공적으로 로딩되어 설정되면 TRUE가 리턴된다. 따라서 아이콘 로딩이 왜 실패했는지에 대한 세부적인 정보를 원하지 않는 한 현재로선 NULL을 세 번째 매개변수로 전달하는 편이 안전하다. GError 구조는 제 4장에서 논할 것이다.

보류 이벤트 처리

이따금씩 애플리케이션에서 모든 보류(pending) 이벤트를 처리하길 원할 때가 있다. 이는 처리에 오랜 시간이 소요되는 코드 조각을 실행 중인 경우 매우 유용하다. 애플리케이션이 멈춘(freeze) 것처럼 보일 것인데, 다른 프로세스가 CPU를 차지하는 경우 위젯이 그려지지 않기 때문이다. 가령 필자가 생성한 통합 개발 환경인 OpenLDev에서는 build 명령이 처리되는 동안 필자가 사용자 인터페이스를 업데이트해야 한다. 그렇지 않으면 창이 잠기면서 build가 완료될 때까지 어떤 build 출력도 표시되지 않을 것이다.

아래 루프는 이러한 문제에 대한 해답이 되는데, 신규 GTK+ 프로그래머들이 제기할 법한 수많은 질문에 해답이 되기도 한다.

```
while (gtk_events_pending ())
    gtk_main_iteration ();
```

루프는 `gtk_main_iteration()`을 호출하는데 이는 자신의 애플리케이션에 대한 첫 번째 보류 이벤트를 처리할 것이다. 이는 `gtk_events_pending()`이 TRUE를 리턴하여, 처리되어야 할 이벤트가 기다리고 있는지 알려주는 동안에도 계속된다.

루프는 freezing 문제를 해결하는 데에 쉬운 해답이 되지만 문제를 피할 수 있는 코딩 전략들을 사용하는 편이 더 나을 것이다. 예를 들어, 처리가 필요한 더 중요한 액션이 없을 때에만 함수를 호출하기 위해서는 `idle` 함수를 사용할 수 있는데, 이는 제 6장에서 다룰 것이다.

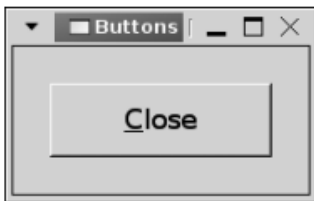
### 버튼

`GtkButton` 위젯은 특별한 타입의 컨테이너로서 그 자식을 클릭 가능한 개체로 바꾼다. 해당 위젯은 하나의 자식만 보유할 수 있다. 하지만 자식 위젯은 자체가 컨테이너가 될 수 있으므로 버튼은 이론적으로 수많은 자식을 가진 조상(ancestor)이 될 수 있다. 이는 버튼이 라벨과 이미지를 동시에 보유하도록 허용한다.

`GtkButton` 위젯의 목적은 자식을 클릭 가능하게 만드는 것이기 때문에 버튼이 활성화되었다는 알림을 받으면 항상 `clicked` 시그널을 사용해야 한다. 바로 다음 예제에서 해당 시그널을 사용해보겠다.

`GtkButton` 위젯은 `GtkLabel`을 이용해 새 버튼을 자식으로서 생성하는 `gtk_button_new_with_label()`을 이용해 주로 초기화된다. 빈 `GtkButton`를 생성하고 후에 자신만의 자식을 추가하고 싶다면 `gtk_button_new()`를 사용할 수 있지만 대부분의 경우 이것을 사용하고 싶지 않을 것이다.

그림 2-4는 연상기호 기능을 가진 버튼을 보여준다. 연상기호 라벨은 밑줄이 쳐진 문자로 인식할 수 있다. 아래 버튼의 경우 `Alt+C`를 누르면 버튼이 클릭될 것이다.



`gtk_button_new_with_mnemonic()` 함수는 연상기호 라벨을 지워하는 새 버튼을 초기화할 것이다. 사용자가 명시된 문자 키와 함께 `Alt` 키를 누르면 버튼이 활성화될 것이다. 가속기(accelerator)는 사전 정의된 액션을 활성화하는 데에 사용할 수 있는 키 또는 키의 집합에 해당한다.

**Note** 일정 타입의 사용자 인터페이스를 제공하는 위젯에서 연상기호 옵션이 이용 가능하다면 그러한 기능을 활용할 것을 권한다. 키보드 단축키가 없더라도 일부 사용자는 마우스 대신 키보드를 사용해 사용자 인터페이스를 살펴보는 편을 선호한다.

리스팅 2-3은 `clicked` 시그널을 이용하는 `GtkButton`의 단순한 예제다. 버튼을 누르면 창이 소멸되고 애플리케이션이 중단될 것이다. 해당 예제의 버튼도 연상기호와 키보드 가속기 기능을 이용한다. 이 예제의 스크린샷은 그림 2-4에 실려 있다.

리스팅 2-3. `GtkButton` 위젯 (buttons.c)

```
#include <gtk/gtk.h>

static void destroy (GtkWidget*, gpointer);

int main (int argc,
          char *argv[])
```

```

{
    GtkWidget *window, *button;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 25);
    gtk_widget_set_size_request (window, 200, 100);

    g_signal_connect (G_OBJECT (window), "destroy",
        G_CALLBACK (destroy), NULL);

    /* Create a new button that has a mnemonic key of Alt+C. */
    button = gtk_button_new_with_mnemonic ("_Close");
    gtk_button_set_relief (GTK_BUTTON (button), GTK_RELIEF_NONE);

    /* Connect the button to the clicked signal. The callback function
    receives the
    * window followed by the button because the arguments are swapped.
    */
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
        G_CALLBACK (gtk_widget_destroy),
        (gpointer) window);

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}
/* Stop the GTK+ main loop function. */
static void
destroy (GtkWidget *window,
        gpointer data)
{
    gtk_main_quit ();
}

```

리스팅 2-3에서 버튼이 클릭되면 메인 창에서 `gtk_widget_destroy()`가 호출된다. 이는 매우 간단한 예제지만 대부분의 애플리케이션에서 유용하게 사용된다. GNOME Human Interface Guidelines는 <http://developer.gnome.org/projects/gup/hig> 에서 확인하고 다운로드할 수 있는데, 개인설정 대화상자는 설정이 변경된 직후 설정을 적용시켜야 한다고 명시된다.

따라서 개인설정 대화상자를 설정하면 하나의 버튼만 필요할 가능성이 높다. 버튼을 포함하고 변경내용을 저장하는 창을 소멸시키는 것이 버튼의 목적일 것이다.

버튼을 생성하고 나면 `gtk_button_set_relief()`를 이용해 `GtkButton` 주위에 특정 양감(relief) 정도를 추가할 수 있다. 양감은 주위 위젯으로부터 버튼을 구별시키는 3-D 타입의 테두리다. `GtkReliefStyle` 목록 값은 아래를 준

수한다.

- `GTK_RELIEF_NORMAL`: 버튼의 모든 변 주위에 양감을 추가한다.
- `GTK_RELIEF_HALF`: 버튼의 절반에만 양감을 추가한다.
- `GTK_RELIEF_NONE`: 버튼 주위에 양감을 추가하지 않는다.

리스트 2-3은 새로운 시그널 연결 함수인 `g_signal_connect_swapped()`를 소개한다. 해당 함수는 콜백 함수가 실행될 때 데이터 매개변수와 시그널이 방출되는 객체의 위치를 바꾼다(`swap`).

```
g_signal_connect_swapped (G_OBJECT (button), "clicked",
                          G_CALLBACK (gtk_widget_destroy),
                          (gpointer) window);
```

이는 콜백 함수에서 `gtk_widget_destroy()`의 사용을 허용하여 `gtk_widget_destroy (window)`를 호출할 것이다. 콜백 함수가 하나의 매개변수만 수신한다면 객체는 무시될 것이다.

### 위젯 프로퍼티

`GObject`는 프로퍼티 시스템을 제공하여 위젯이 그 사용자와 어떻게 상호작용하는지와 화면에 어떻게 그려지는지를 맞춤 설정하도록 해준다. 이번 절에서는 스타일, 리소스 파일, `GObject`의 프로퍼티 시스템을 사용하는 방법을 학습할 것이다.

`GObject` 클래스로부터 파생된 모든 클래스는 원하는 수만큼 프로퍼티를 설치할 수 있다. `GTK+`에서 이러한 프로퍼티는 위젯이 어떻게 행동할 것인지에 관한 정보를 저장한다. 가령 `GtkButton`은 버튼이 사용하는 양감 스타일을 정의하는 `relief`라는 프로퍼티를 갖고 있다.

아래 코드에서는 `g_object_get()`을 이용해 버튼의 `relief` 프로퍼티가 보관한 현재 값을 검색한다. 해당 함수는 `NULL`로 종료되는 프로퍼티와 변수의 리스트를 수락하여 리턴 값을 보관한다.

```
g_object_get (button, "relief", &value, NULL);
```

각 객체에는 수많은 프로퍼티가 있기 때문에 본문에 전체 리스트를 실지는 않을 것이다. 특정 위젯에 이용 가능한 프로퍼티에 관한 정보는 API 문서를 참조해야 한다.

### 위젯 프로퍼티 설정하기

`g_object_set()`를 이용하면 프로퍼티에 대한 새 값을 쉽게 설정할 수 있다. 예를 들어, 버튼의 `relief` 프로퍼티는 `GTK_RELIEF_NORMAL`로 설정되었다.

```
g_object_set (button, "relief", GTK_RELIEF_NORMAL, NULL);
```

각 위젯의 많은 프로퍼티를 설정 및 검색하도록 함수가 제공된다. 하지만 모든 프로퍼티가 그러한 선택을 할 수 있는 것은 아니다. 이러한 함수들은 제 8장에서 `GtkTreeView` 위젯을 학습할 때 매우 유용해지는데, 그 장에서 사용되는 많은 객체들은 어떤 프로퍼티에 대해서도 `get` 또는 `set` 함수를 제공하지 않기 때문이다.

`GObject`의 `notify` 시그널을 이용해 특정 프로퍼티를 감시하는 수도 있다. 프로퍼티를 `notify::property-name` 시그널로 연결하면 감시가 가능하다. 리스트 2-4의 예제는 `relief` 프로퍼티가 변경되면 `property_changed()`를 호출한다.

### 리스트 2-4. Notify 프로퍼티 사용하기

```
g_signal_connect (G_OBJECT (button), "notify::relief",
                 G_CALLBACK (property_changed), NULL);
```

...

```
static void
```

```
property_changed (GObject *button,
                  GParamSpec *property,
                  gpointer data)
{
    /* Handle the property change ... */
}
```

**Caution** 시그널명을 입력 시에는 대시나 밑줄의 사용이 모두 허용되지만 notify 시그널을 사용 시에는 항상 대시를 사용해야 한다. 가령, GtkWidget의 can-focus 프로퍼티를 감시하고 싶다면 notify::can\_focus 는 허용되지 않는다는 말이다! notify는 시그널명이므로 can-focus는 위젯 프로퍼티명이다.

콜백 함수는 GParamSpec이라는 새로운 타입의 객체를 수신하는데, 이것은 프로퍼티에서 무엇이 변경되었는지에 대한 정보를 보유한다. 현재로서는 property->name을 이용해 변경된 프로퍼티명을 검색할 수 있다는 사실만 알면 된다. GParamSpec 구조에 관해서는 제 11장에서 자신만의 커스텀 위젯에 프로퍼티를 추가하는 방법을 학습할 때 더 자세히 알아볼 것이다.

모든 GObject는 프로퍼티 시스템에 더해 문자열 리스트를 포인터 리스트와 연관시키는 테이블을 갖고 있다. 이는 쉽게 접근할 수 있는 데이터를 객체로 추가할 수 있도록 해주므로 시그널 핸들러에게 추가 정보를 전달해야 하는 경우 특히 유용하다. 객체에 새 데이터 필드를 추가하려면 g\_object\_set\_data()를 호출하기만 하면 된다. 해당 함수는 data를 가리키는 데에 사용될 유일한 문자열을 수락한다. 동일한 키 이름으로 된 연관 (association)이 이미 존재하는 경우, 새 데이터가 기존 데이터를 대체할 것이다.

```
void g_object_set_data (GObject *object,
                       const gchar *key,
                       gpointer data);
```

데이터로 접근해야 한다면 key와 연관된 포인터를 리턴하는 g\_object\_get\_data()를 호출할 수 있다. g\_signal\_connect()를 이용해 임의의 데이터 조각을 전달하는 대신 위의 데이터 전달 방법을 사용해야 한다.

### 자신의 이해도 시험하기

제 2장에서는 창과 라벨 위젯에 관해 학습하였다. 이제 지식을 실제로 사용해볼 때가 되었다. 아래 두 개의 연습문제를 통해 당신은 GTK+ 애플리케이션의 구조, 시그널, GObject 프로퍼티 시스템에 대한 본인의 지식을 활용할 것이다.

#### 연습문제 2-1. 이벤트와 프로퍼티 사용하기

이 연습문제는 본 장의 첫 부분에 실린 두 예제를 확장시켜 스스로를 소멸시키는 능력이 있는 GtkWindow를 생성할 것이다. 우선 개발자의 이름을 창의 제목으로 설정해야 한다. 그리고 개발자의 성을 기본 텍스트 문자열로 가진, 선택 가능한 GtkLabel 을 창의 자식으로 추가해야 한다.

그 외에 창의 크기가 조정이 불가능하고, 최소 크기는 300 x 100 픽셀이어야 한다는 특성이 있다. 이러한 작업을 실행하기 위한 함수는 본 장의 내용에서 찾아볼 수 있다.

다음으로, API 문서를 살펴보고 창에 key-press-event 시그널을 연결하라. key-press-event 콜백 함수에서 창 제목과 라벨 텍스트를 바꿔본다. 예를 들어, 콜백 함수를 처음으로 호출하면 창 제목은 당신의 성으로 설정되고 라벨은 당신의 이름으로 설정되어야 한다.

아래 함수가 도움이 될 것이다.

```
gint g_ascii_strcasecmp (const gchar *str1, const gchar *str2);
```

g\_ascii\_strcasecmp() 에서 두 개의 문자열이 동일하면 0이 리턴된다. str1이 str2보다 작으면 음수가 리턴된다. 그 외의 경우 양수가 리턴된다.

연습문제 2-1을 완료했다면 부록 F에서 해답에 대한 설명을 찾을 수 있으며, [www.gtkbook.com](http://www.gtkbook.com)에서 해답의 전체 소스 코드를 다운로드할 수 있다.

**연습문제 2-2. GObject 프로퍼티 시스템**

이번에는 연습문제 2-1을 확장시키되 창의 제목, 높이, 너비는 GObject에서 제공하는 함수를 이용해 설정할 것이다. 또 콜백 함수 내에서 창 제목과 라벨 텍스트를 수반하는 모든 연산은 GObject가 제공하는 함수를 이용해 실행되어야 한다. 게다가 notify 시그널을 이용해 창의 제목을 감시해야 한다. 제목이 변경되면 terminal 출력을 통해 사용자에게 통지해야 한다.

힌트: GLib가 제공하는 함수 `g_message()`를 이용해 terminal로 메시지를 출력할 수 있다. 해당 함수는 `printf()`가 지원하는 것과 동일한 포맷팅을 준수한다.

두 연습문제를 완료했다면 이제 다음 장으로 넘어가 컨테이너 위젯을 다룰 준비가 되었다. 컨테이너 위젯은 메인 창이 하나 이상의 위젯을 포함하도록 허용하는데, 이번 장에 실린 모든 예제가 이에 해당한다.

하지만 다음으로 넘어가기 전에 GTK+로 개발하기의 내용을 보완할 수 있는 [www.gtkbook.com](http://www.gtkbook.com)에 대해 알아야겠다. 해당 웹 사이트는 다운로드, 추가 GTK+ 정보의 링크, C refresher 지침서, API 문서 등을 포함한다. 본문을 읽으면서 웹 사이트를 참고한다면 GTK+를 학습 시 도움이 될 것이다.

**요약**

이번 장에서는 가장 기본적인 GTK+ 위젯과 애플리케이션에 대해 학습하였다. 첫 번째 애플리케이션은 "Hello World" 예제로, 모든 GTK+ 애플리케이션이 필요로 하는 기본 호출을 보였는데 이러한 호출은 다음과 같다.

- `gtk_init()`를 이용해 GTK+ 초기화하기
- 최상위 수준의 `GtkWindow` 생성하기
- `GtkWindow` 표시하기
- `gtk_main()`을 이용해 메인 루프로 이동하기

두 번째 예제에서는 GTK+ 애플리케이션 내에서 시그널, 이벤트, 콜백 함수의 목적에 대해 배웠다.

`GtkContainer` 구조는 `GtkWindow`와 연관된 것으로 소개되었다. 또 GObject 라이브러리에 의해 구현되는 위젯 계층구조 시스템의 목적을 살펴보았다.

다음으로 `GtkWidget`, `GtkWindow`, `GtkLabel`에 연관된 유용한 함수들을 살펴보았다. 그 중 다수는 본 저서를 통해 사용될 것이다. 두 연습문제에서 사실 몇 가지 함수들을 실제로 사용해야 했을 것이다.

또 마지막 예제를 통해서는 `GtkButton` 위젯을 소개하였다. `GtkButton`은 자식 위젯을 클릭 가능한 버튼으로 만드는 컨테이너 타입이다. 이를 이용해 라벨, 연상기호, 또는 임의의 위젯을 표시할 수 있다. 버튼은 제 4장에서 더 상세히 다루겠다.

다음 장에서는 `GtkContainer` 구조에 대해, 그리고 이를 마음대로 광범위한 컨테이너 위젯 집합에 관계시키는 방법을 학습할 것이다.

**Notes**

# FoundationsofGTKDevelopment:Chapter 03

## 제 3 장 컨테이너 위젯

### 컨테이너 위젯

제 2장을 통해 개발자가 생성하는 GTK+ 애플리케이션마다 필요로 하는 기본적인 것들을 제시하였다. 그 외에 시그널, 이벤트, 콜백 함수, GtkLabel 위젯, GtkButton 위젯, GtkContainer 클래스에 대한 개념도 소개하였다. 이번 장에서는 컨테이너 위젯의 두 가지 타입을 다룰 것인데, decorator와 레이아웃 컨테이너가 그것이다. 그 다음으로 박스, 노트북, 핸들 박스, 이벤트 박스를 포함해 많은 중요한 컨테이너 위젯에 대한 지식을 습득할 것이다.

마지막으로 다루게 될 위젯인 GtkEventBox가 없다면 위젯은 GDK 이벤트를 활용할 수 없을 것이다.

이번 장에서 학습하게 될 내용은 다음과 같다.

- GtkContainer 클래스와 그 자식 클래스(descendents)의 목적
- 박스, 페인(pane), 테이블을 포함해 레이아웃 컨테이너를 사용하는 방법
- 고정 컨테이너의 사용 시 장·단점
- 여러 페이지로 된 노트북 컨테이너를 생성하는 방법
- 이벤트 박스를 이용해 모든 위젯에게 이벤트를 제공하는 방법

### GtkContainer

GtkContainer 클래스는 앞의 절에서 간략하게 다룬 바 있지만 심도 있게 살펴보기 위해서는 우선 숙련된 GTK+ 개발자가 되어야 한다. 따라서 이번 절에서는 해당 추상 클래스의 모든 중요한 측면들을 다룬다.

컨테이너 클래스의 주요 목적은 부모 위젯이 하나 또는 그 이상의 자식들을 포함하도록 허용하는 데에 있다. GTK+에는 두 가지 타입의 컨테이너 위젯이 있는데, 하나는 자식들과 decorator들을 배치하는 데에 사용되는 것이고 나머지 하나는 위치지정을 넘어서 특정 종류의 기능까지 자식에게 추가하는 것이다.

### Decorator 컨테이너

제 2장에서는 GtkBin에서 파생된 위젯 GtkWindow를 소개하였다. GtkBin은 하나의 자식 위젯만 보유하는 기능을 가진 컨테이너 클래스 타입이다. 해당 클래스에서 파생된 위젯을 decorator 컨테이너라고 부르는데, 자식 위젯에게 특정 타입의 기능을 추가하기 때문이다.

가령 GtkWindow는 최상위 수준의 위젯에 위치시킬 수 있는 추가 기능을 자식 위젯에게 제공한다. decorator의 예제로는 자식 위젯의 주변에 프레임을 그리는 GtkFrame 위젯, 자식을 클릭 가능한 버튼으로 만드는 GtkButton, 사용자에게 자식을 숨기고 표시하는 GtkExpander가 있다. 이러한 위젯들은 모두 자식 위젯을 추가할 때 gtk\_container\_add()를 사용한다.

GtkBin은 gtk\_bin\_get\_child()라는 하나의 함수만 제공하는데, 이것은 컨테이너의 자식 위젯에 대한 포인터를 검색하도록 해준다. GtkBin 클래스의 실제 목적은 하나의 자식 위젯만 요구하는 모든 서브클래스들이 파생될 수 있는 실체화 가능한(instantiable) 위젯을 제공한다. 이것은 공통 기반에 사용되는 중심(central) 클래스다.

```
GtkWidget* gtk_bin_get_child (GtkBin *bin);
```

GtkBin으로부터 파생된 위젯으로는 창(windows), 정렬(alignments), 프레임, 버튼, 항목, 콤보 박스, 이벤트 박스, 익스펜더(expanders), 핸들 박스, 스크롤을 이용한 창, 툴 항목이 있다. 이러한 컨테이너들 중 다수는 이번 장과 이후 여러 장에서 다룰 것이다.

레이아웃 컨테이너

GTK+가 제공하는 또 다른 컨테이너 위젯 타입은 레이아웃 컨테이너라고 불린다. 이는 다수의 위젯을 배열하는 데에 사용된다. 레이아웃 컨테이너는 `GtkContainer`에서 직접 파생된다는 사실을 특징으로 한다.

이름에서 짐작할 수 있듯이 레이아웃 컨테이너는 사용자의 개인설정, 개발자의 명령어, 내장된 규칙에 따라 그 자식들을 올바르게 배열하는 데에 목적이 있다. 사용자 개인설정에는 테마 및 글꼴 개인설정의 사용도 포함된다. 이러한 값을 오버라이드할 수도 있지만 대부분의 경우 사용자의 개인설정을 존중하도록 한다. 모든 컨테이너 위젯에 적용되는 크기조정(resizing) 규칙도 있는데, 이는 다음 절에서 다루겠다.

레이아웃 컨테이너로는 박스, 고정 컨테이너, 패인(paned) 위젯, 아이콘 뷰, 레이아웃, 메뉴 셸(menu shell), 노트북, 소켓, 테이블, 텍스트 뷰, 툴바, 트리 뷰가 있다. 이들 중 대부분은 이번 장을 통해, 나머지는 다른 장에서 다룰 것이다. 본문에서 찾을 수 없는 정보는 API 문서를 참고하면 될 것이다.

자식 위젯의 크기조정

컨테이너는 자식 위젯을 배열하고 꾸미는 것 외에 자식 위젯의 크기도 조정해야 한다. 크기조정은 두 가지 단계로 진행되는데, `size requisition`과 `size allocation`이 그것이다. 짧게 말하자면, 두 가지 단계는 위젯에 이용 가능한 크기를 협상한다. 따라서 이는 위젯과 그 부모들과 자식들 간에 이루어지는 재귀적 통신 과정으로 볼 수 있겠다.

`Size requisition`은 자식 위젯의 바람직한 크기를 의미한다. 프로세스는 최상위 수준 위젯에서 시작하고, 자식 위젯들에게 선호하는 크기를 질문한다. 자식들은 또 그들의 자식들에게 질문하는 식으로 마지막 자식들로 도달할 때까지 지속된다.

이 시점에서 마지막 자식은 화면에 올바르게 표시되기 위해 필요한 공간과 프로그래머가 요청한 크기를 기반으로 자신이 원하는 크기를 결정한다. 예를 들어 `GtkLabel` 위젯은 그 텍스트를 화면에 완전히 표시하기에 충분한 공간을 요청하고, 개발자가 만일 그보다 더 큰 크기를 요청하였다면 더 많은 공간을 요청할 것이다.

이후 자식 위젯은 조상 위젯들에게 그 크기를 전달하는데 이는 최상위 수준의 위젯이 자식의 `requisition`을 바탕으로 필요한 공간을 수신할 때까지 지속된다.

**typedef struct**

```
{
    gint width;
    gint height;
} GtkRequisition;
```

각 위젯은 `GtkRequisition` 객체에 너비와 높이 값으로서 자신의 크기 개인설정(size preferences)을 보관한다. `requisition`은 요청에 불과하므로 부모 위젯은 이 값은 굳이 이행할 필요가 없음을 명심한다.

최상위 수준 위젯이 원하는 공간의 양을 스스로 결정하면 `size allocation`이 시작된다. 최상위 수준 위젯을 `nonresizable`(크기조정 불가)로 설정했다면 위젯의 크기는 절대로 조정되지 않을 것이며, 어떤 액션도 발생하지 않고 `requisition`은 무시될 것이다. 그 외의 경우는 최상위 수준 위젯이 스스로 바람직한 크기로 조정할 것이다. 이후 이용 가능한 공간을 자식 위젯으로 전달할 것이다. 이러한 프로세스는 모든 위젯이 스스로 크기를 조정할 때까지 반복된다.

**typedef struct**

```
{
    gint x;
    gint y;
    gint width;
    gint height;
} GtkAllocation;
```



모든 위젯에 대한 size allocation은 각 자식 위젯마다 GtkAllocation 구조의 하나의 인스턴스에 보관된다. 이러한 구조는 gtk\_widget\_size\_allocate()를 이용해 크기조정을 위해 자식 위젯으로 전달된다. 해당 함수는 프로그래머에 의해 직접 호출되기도 하지만 대부분의 경우는 권하지 않는다.

대개의 경우 자식 위젯에게 자신들이 요청한 공간이 제공되지만 그렇지 않은 특정 상황들이 있다. 가령 최상위 수준의 위젯의 크기 조정이 불가능한 경우 requisition은 고려되지 않을 것이다.

반대로 위젯이 부모에 의해 크기가 할당된 경우 위젯은 새로운 크기의 자신을 다시 그리는 수 밖에 없다. 따라서 gtk\_widget\_size\_allocate()를 호출할 때는 주의해야 한다. 대부분의 경우 위젯의 크기조정 시 gtk\_widget\_set\_size\_request()가 최선의 선택이다.

### 컨테이너 시그널

GtkContainer 클래스는 현재 네 가지 시그널, add, check\_resize, remove, set\_focus\_child를 제공한다.

- add: 자식 위젯이 컨테이너로 추가 또는 패키징된다. 당신이 gtk\_container\_add()를 명시적으로 호출하지 않고 위젯의 내장된 패키징 함수를 사용하더라도 해당 시그널은 방출될 것이다.
- check\_resize: 추가 액션을 취하기 전에 컨테이너가 자식 위젯의 크기조정이 필요한지 검사한다.
- remove: 컨테이너로부터 자식 위젯이 제거된다.
- set\_focus\_child: 컨테이너의 자식 위젯이 윈도우 관리자로부터 포커스를 받는다.

이제 GtkContainer 클래스의 목적을 알게 되었으니 컨테이너 위젯의 다른 타입으로 넘어갈 것이다. 창, 특히 GtkBin 위젯의 타입에 관해 이미 학습한 바 있으니 GtkBox라 불리는 레이아웃 컨테이너를 살펴볼 것이다.

### 수평 및 수직 박스

GtkBox는 1차원의 직사각형 영역에 다수의 자식 위젯을 넣을 수 있도록 해주는 추상 컨테이너 위젯이다. 박스에는 두 가지 유형이 있는데, GtkVBox는 자식 위젯을 하나의 열에 패키징하고, GtkHBox는 하나의 행에 패키징한다.

**Note** 본 서적의 나머지 부분에 실린코드 리스팅은 각 절에서 중요한 텍스트의 부분만 포함할 것이다. 따라서 전체 예제를 확인하기 위해서는 소스 코드를 다운로드 해야 할 것이다. 가령, destroy 콜백 함수는 지금쯤이면 사용법을 익혔을 것이기 때문에 이후 예제에서는 포함하지 않을 것이다. 하지만 [www.gtkbook.com](http://www.gtkbook.com)에서 그 소스 코드를 다운로드할 수 있다.

리스팅 3-1. 기본 패키징의 수직 박스 (boxes.c)

```
#include <gtk/gtk.h>

#define NUM_NAMES 4
const gchar* names[] = { "Andrew", "Joe", "Samantha", "Jonathan" };

int main (int argc,
          char *argv[])
{
    gint i;
    GtkWidget *window, *vbox;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Boxes");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 200, -1);
```

```

vbox = gtk_vbox_new (TRUE, 5);

/* Add four buttons to the vertical box. */
for (i = 0; < NUM_NAMES; i++)
{
    GtkWidget *button = gtk_button_new_with_label (names[i]);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), button);

    g_signal_connect_swapped (G_OBJECT (button), "clicked",
        G_CALLBACK (gtk_widget_destroy),
        (gpointer) button);
}

gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window);

gtk_main ();
return 0;
}

```

리스트링 3-1은 GtkBox 위젯의 간단한 모습이다. 애플리케이션의 그래픽 출력은 그림 3-1에 표시되어 있다. 이름의 시작 위치는 다르게 패킹되었으나 배열에 추가된 순서대로 표시되었음을 명심한다.



리스트링 3-1을 분석해보면 GtkVBox와 GtkHBox 위젯이 동일한 함수 집합을 사용함을 눈치챌 것인데, 이는 둘 다 GtkBox 클래스로부터 파생되기 때문이다. 각 함수의 매개변수는 동일하지만, 수직 박스는 gtk\_vbox\_new()를 이용해 생성되고, 수평 박스는 gtk\_hbox\_new()를 이용해 생성된다는 점이 유일하게 다르다.

어떤 위젯에서든 마찬가지로 객체를 사용하기 전에 GtkVBox를 초기화할 필요가 있다. gtk\_vbox\_new()에서 첫 번째 매개변수는 박스 내 모든 자식이 동질적인지(homogeneous) 여부를 나타낸다. TRUE로 설정되면 모든 위젯에 들어맞는 최소의 공간이 모든 자식 위젯에게 주어질 것이다.

```

GtkWidget* gtk_vbox_new (gboolean homogeneous,
    gint spacing);

```

두 번째 매개변수는 각 자식 위젯과 그 주변 위젯 간 공간의 기본 픽셀 값을 명시한다. 박스가 동일한 간격으로 설정되지 않을 경우 자식 위젯이 추가되면서 각 셀마다 해당 값이 변경될 수 있다.

리스트링 3-1에서 GtkBox 위젯으로 우선 라벨이 추가되고 나면 더 이상 라벨로 접근할 필요가 없으므로 애플리케이션은 객체마다 별도의 포인터를 보관하지 않는다. 부모 위젯이 소멸되면 자동으로 제거될 것이다. 이후 각 버튼은 패킹(packing)이라 불리는 방법을 이용해 박스로 추가된다.

gtk\_box\_pack\_start\_defaults()를 이용해 박스로 위젯을 추가함으로써 자식 위젯은 자동으로 설정된 세 개의 프로퍼티를 가지는데, Expanding은 TRUE로 설정되어 박스로 할당된 추가 공간을 셀(cell)에 자동으로 제공할 것이다. 이러한 공간은 공간을 요청한 모든 셀에게 균등하게 배분된다. fill 프로퍼티도 TRUE로 설정되는데, 이는 위젯이 패딩(padding)을 이용해 채우는 대신 제공된 추가 공간으로 확장될 것을 의미한다. 마지막으로 셀과 그 주변 위젯 사이에 위치한 패딩의 양은 0 픽셀로 설정된다.

```
void gtk_box_pack_start_defaults (GtkBox *box,
    GtkWidget *widget);
```

패킹 박스는 함수의 명명 규칙 때문에 이해하기가 약간 힘들다. 손쉽게 이해하려면 패킹이 시작되는 장소와 관련시키는 방법을 이용한다. 시작 위치에서 패킹할 경우 첫 번째 자식은 박스의 가장 상단이나 좌측에 나타나는 방식으로 자식 위젯들이 추가될 것이다. 끝 위치에서 패킹하면 첫 번째 자식은 박스의 하단이나 우측에 표시될 것이다.

다시 말해, 시작에 대한 참조 위치는 당신이 위젯을 추가하면서 이동한다. 위젯을 끝 위치로 추가해도 동일한 프로세스가 발생한다. 따라서 요소를 역순으로 추가하려면 gtk\_box\_end() 또는 gtk\_box\_pack\_end\_defaults()를 이용해야 한다. 이와 관련된 예제는 리스팅 3-2에 발췌된 코드에서 확인할 수 있다.

리스팅 3-2. 패킹 매개변수 명시하기 (boxes2.c)

```
vbox = gtk_vbox_new (TRUE, 5);

/* Add four buttons to the vertical box, packing at the end. */
for (i = 0; i < NUM_NAMES; i++)
{
    GtkWidget *button = gtk_button_new_with_label (names[i]);
    gtk_box_pack_end (GTK_BOX (vbox), button, FALSE, FALSE, 5);

    g_signal_connect_swapped (G_OBJECT (button), "clicked",
        G_CALLBACK (gtk_widget_destroy),
        (gpointer) button);
}
```

그림 3-2는 리스팅 3-2의 그래픽 출력을 나타낸다. 끝에서부터 각 위젯을 패킹하였으므로 역순으로 표시되었다. 박스의 끝에서 패킹이 시작되었고, 각 자식 위젯은 이전(previous) 위젯보다 먼저 패킹되었다. 패킹 함수의 시작과 끝에 호출을 마음대로 배치해도 좋다. GTK+는 두 참조 위치를 모두 파악한다.



expanding, filling, spacing에 기본값을 사용하고 싶지 않다면 gtk\_box\_pack\_end() 또는 gtk\_box\_pack\_start()를 이용해 각 패킹 프로퍼티에 다른 값을 명시할 수 있다.

expand 프로퍼티를 TRUE로 설정하면 셀이 확장되어 위젯이 필요로 하는 박스로 할당된 추가 공간을 차지할 것이다. fill 프로퍼티를 TRUE로 설정하면 위젯 자체가 확장되어 셀에 이용 가능한 추가 공간을 채울 것이다. 표 3-1은 expand와 fill 프로퍼티의 가능한 조합을 간략하게 설명한다.

expand	fill	결과
TRUE	TRUE	셀이 확장되어 박스로 할당된 추가 공간을 차지하고, 자식 위젯은 그 공간을 채우도록 확장될 것이다.
TRUE	FALSE	셀이 확장되어 추가 공간을 차지하지만 위젯은 확장되지 않을 것이다. 대신 추가 공간은 비어 있을 것이다.
FALSE	TRUE	셀과 위젯 모두 추가 공간을 채우기 위해 확장되지 않을 것이다. 두 가지 프로퍼티를 모두 FALSE로 설정할 때도 마찬가지다.
FALSE	FALSE	셀과 위젯 모두 추가 공간을 채우기 위해 확장되지 않을 것이다. 창의 크기를 조정하더라도 셀은 스스로 크기를 조정하지 않을 것이다.

표 3-1. expand 프로퍼티와 fill 프로퍼티

앞의 `gtk_box_pack_end()` 호출에서 각 셀은 셀과 그 주변 셀들 사이의 간격을 5 픽셀로 하도록 요청을 받는다. 또한 표 3-1에 따르면, 셀과 그 자식 위젯 중 어떤 것도 박스가 제공한 추가 공간을 차지하기 위해 확장되지는 않을 것이다.

```
void gtk_box_pack_end (GtkBox *box,
    GtkWidget *child,
    gboolean expand,
    gboolean fill,
    guint padding);
```

**Note** 다른 그래픽 툴킷을 이용해 프로그래밍할 경우 GTK+가 제공하는 크기 협상(size negotiation) 시스템이 이상하게 보일지도 모르겠다. 하지만 그 장점은 빠르게 학습할 수 있을 것이다. 사용자 인터페이스를 변경할 경우 프로그래머가 모든 것을 프로그램적으로 위치를 지정하도록 요구하는 대신 GTK+가 자동으로 크기 조정을 처리한다. 이는 GTK+를 학습하는 동안 큰 장점이 될 것이다.

GtkBox 위젯 내 요소들의 순서를 사용자에게 보여주기 전에 `finalize`를 시도하는 동안 `gtk_box_reorder_child()`를 이용하면 박스 내에 자식 위젯들의 재정렬이 가능하다.

```
void gtk_box_reorder_child (GtkBox *box,
    GtkWidget *child,
    gint position);
```

이 함수를 이용해 자식 위젯을 GtkBox 안에 새로운 위치로 이동시킬 수 있다. GtkBox 컨테이너 내부의 첫 번째 위젯의 위치는 0부터 색인된다. position 값을 -1 또는 자식의 개수보다 큰 값으로 명시할 경우 박스의 마지막에 위젯이 위치될 것이다.

### 수평 및 수직적 패인

GtkPaned는 정확히 두 개의 위젯을 보유하는 특수 형태의 컨테이너 위젯이다. 크기조정 막대가 두 위젯 사이에 위치하는 형태로, 사용자는 막대를 일정 방향으로 드래그하여 두 개의 위젯의 크기를 조정할 수 있다. 막대가 움직이면 사용자 상호작용 또는 프로그램 호출에 의해 두 개의 위젯 중 하나는 크기가 줄어들고 하나는 늘어난다.

패인(paned) 위젯에는 두 가지 타입이 있는데, 하나는 수평적 크기조정을 위한 GtkHPaned이고 나머지는 수직적 크기조정을 위한 GtkVPaned다. 박스와 마찬가지로 수평 및 수직 패인 클래스는 위젯을 생성하기 위한 함수만 제공한다. 그 외의 기능은 GtkPaned라는 공통된 부모 클래스에서 정의된다. 리스팅 3-3은 두 개의 GtkButton 위젯을 수평적 패인의 자식으로서 위치시키는 간단한 예를 보여준다.

리스팅 3-3. 수평적 패인(panes.c)

```

#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *hpaned, *button1, *button2;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Panes");

    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 225, 150);

    hpaned = gtk_hpaned_new ();
    button1 = gtk_button_new_with_label ("Resize");
    button2 = gtk_button_new_with_label ("Me!");

    g_signal_connect_swapped (G_OBJECT (button1), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              (gpointer) window);
    g_signal_connect_swapped (G_OBJECT (button2), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              (gpointer) window);

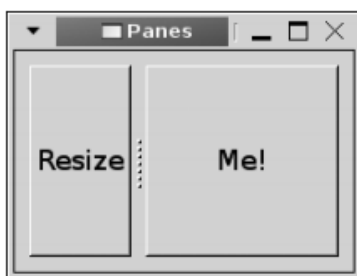
    /* Pack both buttons as the two children of the GtkHPaned widget. */
    gtk_paned_add1 (GTK_PANED (hpaned), button1);
    gtk_paned_add2 (GTK_PANED (hpaned), button2);

    gtk_container_add (GTK_CONTAINER (window), hpaned);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

```

그림 3-3에서 볼 수 있듯이 GtkHPaned 위젯은 두 개의 자식 위젯 사이에 수직막대를 위치시킨다. 막대를 드래그하면 하나의 위젯은 줄어들고 나머지 위젯은 늘어난다. 사실 막대를 이동시켜 하나의 자식을 사용자로부터 완전히 완전히 숨길 수도 있다. `gtk_paned_pack1()`과 `gtk_paned_pack2()`를 이용해 이를 방지하는 법을 학습할 것이다.



리스트 3-3에서 우리는 `gtk_hpaned_new()`를 이용해 `GtkHPaned` 객체를 생성하였다. 수직 패인 위젯을 대신 사용하고자 한다면 `gtk_vpaned_new()`를 호출하면 된다. 그러면 패인 위젯의 타입과 상관없이 `GtkPaned` 함수들이 작동할 것이다.

`GtkPaned`은 두 개의 자식 위젯만 처리할 수 있기 때문에 `GTK+`는 자식 위젯마다 패킹을 위한 함수를 제공한다. 아래 예제에서는 `gtk_paned_add1()`과 `gtk_paned_add2()`를 이용해 두 자식을 모두 `hpaned`로 추가하였다. 해당 함수들은 `GtkPaned` 위젯의 `resize`와 `shrink` 프로퍼티에 기본값을 사용한다.

```
gtk_paned_add1 (GTK_PANED (hpaned), label1);
gtk_paned_add2 (GTK_PANED (hpaned), label2);
```

앞의 `gtk_paned_add1()`과 `gtk_paned_add2()` 호출은 리스트 3-3에서 발췌한 것으로, 아래와 동일하다.

```
gtk_paned_pack1 (GTK_PANED (hpaned), label1, FALSE, TRUE);
gtk_paned_pack2 (GTK_PANED (hpaned), label2, TRUE, TRUE);
```

`gtk_paned_pack1()`과 `gtk_paned_pack2()`에서 세 번째 매개변수는 패인의 크기가 조정되면 자식 위젯이 확장되어야 하는지 여부를 명시한다. `FALSE`로 설정하면 이용 가능한 영역을 얼마나 크게 만들든 자식 위젯은 확장되지 않을 것이다.

마지막 매개변수는 자식을 그것의 `size requisition`보다 작게 만들 수 있는지를 명시한다. 대부분의 경우 `TRUE`로 설정하여 크기조정 막대를 드래그하면 위젯이 사용자로부터 완전히 숨길 수 있길 원할 것이다. 사용자가 이러한 일을 하지 못하도록 방지하려면 네 번째 매개변수를 `FALSE`로 설정하라. 표 3-2는 `resize`와 `shrink` 프로퍼티가 어떻게 밀접하게 연관되는지를 보여준다.

resize	shrink	결과
e	k	
TRUE	TRUE	패인의 크기가 조정되면 위젯은 이용할 수 있는 공간을 모두 차지할 것이며, 사용자는 위젯을 그 <code>size requisition</code> 보다 작게 만들 수 있을 것이다.
TRUE	FALSE	패인의 크기가 조정되면 위젯은 이용할 수 있는 공간을 모두 차지할 것이지만, 위젯이 이용 가능한 공간은 위젯의 <code>size requisition</code> 보다 크거나 같아야 한다.
FALSE	TRUE	위젯은 패인에 이용 가능한 추가 공간을 차지하기 위해 스스로 크기를 조정하지 않을 것이지만 사용자는 위젯을 그 <code>size requisition</code> 보다 작게 만들 수 있을 것이다.
FALSE	FALSE	위젯은 패인에 이용 가능한 추가 공간을 차지하기 위해 스스로 크기를 조정하지 않을 것이며 위젯이 이용 가능한 공간은 그 <code>size requisition</code> 보다 크거나 같아야 한다.

표 3-2. `resize`와 `shrink` 프로퍼티

`gtk_paned_set_position()`을 이용하면 크기조정 막대의 정확한 위치를 쉽게 설정할 수 있다. 위치는 컨테이너 위젯의 상단 또는 좌측면을 기준으로 픽셀로 계산된다. 개발자가 막대의 위치를 0으로 설정하였는데 위젯이 수축(`shrinking`)을 허용하는 경우 막대는 상단 또는 좌측 끝까지 이동할 것이다.

```
void gtk_paned_set_position (GtkPaned *paned,
                             gint position);
```

대부분 애플리케이션은 크기조정 막대의 위치를 기억하는 편을 선호할 것인데, 그래야만 사용자가 다음에 애플리케이션을 로딩했을 때 동일한 위치로 복구할 수 있기 때문이다. 크기조정 막대의 현재 위치는 `gtk_paned_get_position()`을 이용해 검색할 수 있다.

```
gint gtk_paned_get_position (GtkPaned *paned);
```

`GtkPaned`는 많은 시그널을 제공하지만 가장 유용한 시그널 중에서 크기조정 막대가 언제 이동되었는지 알려주는 `move-handle`을 들 수 있겠다. 크기조정 막대의 위치를 기억하길 원하는 경우 해당 시그널은 언제 새 값을

검색해야 하는지 알려줄 것이다. `GtkPaned` 시그널의 전체 리스트는 부록 B에서 찾을 수 있다.

### 테이블

지금까지 필자가 다룬 레이아웃 컨테이너 위젯들은 모두 자식 위젯들을 1 차원으로 패킹하도록 허용하였다. 하지만 `GtkTable` 위젯은 2차원 공간으로 자식 위젯을 패킹하도록 해준다.

`GtkHBox`와 `GtkVBox` 위젯을 여러 개 사용하는 대신 `GtkTable` 위젯을 사용하면 근접한 행과 열에 위치한 자식 위젯들이 자동으로 서로 정렬된다는 장점이 있으나, 박스 내부의 박스는 예외다. 하지만 모든 위젯을 항상 이러한 방식으로 정렬하고 싶지는 않을 것이기 때문에 이러한 장점이 때로는 단점이 되기도 한다.

그림 3-4에는 세 개의 위젯을 포함하는 단순한 테이블이 표시되어 있다. 하나의 라벨이 두 개의 열로 이어짐을 주목하라. 테이블은 영역(region)이 직사각형에 해당하는 한 하나의 위젯이 다수의 열/행으로 이어지는 것을 허용한다.



리스팅 3-4는 그림 3-4에 표시된 `GtkTable`을 생성하여 두 개의 `GtkLabel` 위젯과 하나의 `GtkEntry` 위젯을 2x2 영역으로 삽입한다 (`GtkEntry`의 사용법은 제 4장에서 학습하겠지만 여기서 잠깐 맛보기로 살펴보겠다).

리스팅 3-4. `GtkTable`가 표시하는 이름 (tables.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *table, *label, *label2, *name;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Tables");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 150, 100);

    table = gtk_table_new (2, 2, TRUE);
    label = gtk_label_new ("Enter the following information ...");
    label2 = gtk_label_new ("Name: ");
    name = gtk_enrty_new ();

    /*Attach the two labels and entry widget to their parent container.
    */
    gtk_table_attach (GTK_TABLE (table), label, 0, 2, 0, 1,
                     GTK_EXPAND, GTK_SHRINK, 0, 0);
    gtk_table_attach (GTK_TABLE (table), label2, 0, 1, 1, 2,
                     GTK_EXPAND, GTK_SHRINK, 0, 0);
```

```

gtk_table_attach (GTK_TABLE (table), name, 1, 2, 1, 2,
                 GTK_EXPAND, GTK_SHRINK, 0, 0);

/* Add five pixels of spacing between every row and every column.
*/
gtk_table_set_row_spacings (GTK_TABLE (table), 5);
gtk_table_set_col_spacings (GTK_TABLE (table), 5);

gtk_container_add (GTK_CONTAINER (window), table);
gtk_widget_show_all (window);

gtk_main ();
return 0;
}

```

### 테이블 패킹

`gtk_table_new()`를 이용해 테이블을 생성할 때는 열의 수, 행의 수, 테이블 셀이 동질한지를 명시해야 한다.

```

GtkWidget* gtk_table_new (guint rows,
                          guint columns,
                          gboolean homogeneous);

```

행과 열의 수는 테이블을 생성한 후에 `gtk_table_resize()`를 이용해 변경이 가능하지만 가능하면 초기에 올바른 숫자를 사용해야만 사용자 입장에서 혼동을 피할 수 있다. 전적으로 필요한 경우가 아닌데도 사용자 인터페이스를 고의적으로 변경하는 버릇은 개발자 역시 원치 않을 것이다.

```

void gtk_table_resize (GtkTable *table,
                      guint rows,
                      guint columns);

```

`gtk_table_set_homogeneous()` 함수 또한 테이블의 생성 이후 동질적인 프로퍼티를 리셋하는 데 사용 가능하지만 이 또한 처음에 바람직한 값을 사용해야 한다. 초기 사용자 인터페이스가 설정된 후에는 사용자에게 크기 조정의 제어가 넘어간다.

```

void gtk_table_set_homogeneous (GtkTable *table,
                                gboolean homogeneous);

```

새 위젯의 패킹은 `gtk_table_attach()`를 이용해 실행된다. 두 번째 매개변수 `child`는 개발자가 테이블로 추가하는 자식 위젯을 참조한다.

```

void gtk_table_attach (GtkTable *table,
                      GtkWidget *child,
                      guint left,
                      guint right,
                      guint top,
                      guint bottom,
                      GtkAttachOptions xoptions,
                      GtkAttachOptions yoptions,
                      guint xpadding,
                      guint ypadding);

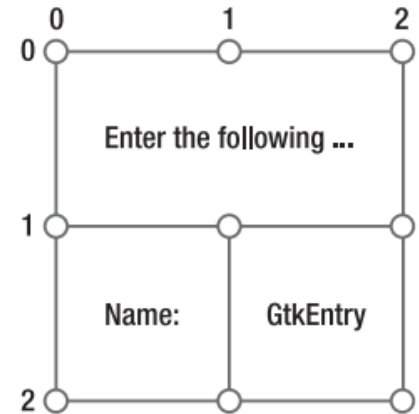
```



left, right, top, bottom 변수들은 테이블 내에서 자식 위젯이 있어야 하는 위치를 설명한다. 가령 리스팅 3-4에서 첫 번째 GtkLabel은 아래 명령을 이용해 추가된다.

```
gtk_table_attach (GTK_TABLE (table), label, 0, 2, 0, 1,
                 GTK_EXPAND, GTK_SHRINK 0, 0);
```

GtkLabel 위젯은 테이블의 첫 열과 첫 행에 바로 붙는데, x 좌표가 추가된 다음 y 좌표가 추가되기 때문이다. 이후 하단에 두 번째 행과 우측의 세 번째 열에 붙는다. 리스팅 3-4의 예제에 표현된 패키징을 그림 3-5에 표시하였다.



두 개의 열을 선택하면 라벨이 붙은 제로 색인(zero-indexed)의 열 부착점(column attach point)이 3개가 생길 것이다. 두 개의 열이 있는 경우 행 부착점에도 마찬가지로 적용된다.

앞서 언급하였듯 위젯이 다수의 셀에 걸친 경우 직사각형 영역을 차지해야 한다. 위젯은 (0, 1, 0, 2)를 이용해 두 개의 행과 하나의 열에 걸치거나, (0, 2, 0, 2)를 이용해 전체 테이블에 걸칠 수도 있다. 부착점을 명시하는 순서는 x 좌표가 먼저 오고 그 다음에 y 좌표가 따라오도록 기억하는 편이 가장 좋다. 부착점을 명시하고 나면 가로와 세로 방향에 대한 부착(attach) 옵션을 제공할 필요가 있다. 우리 예제에서 자식 위젯들은 x 방향으로 확장하고 y 방향으로 축소하도록 설정되었다. GtkAttachOptions 목록(enumeration)에는 3 개의 값이 있다.

- **GTK\_EXPAND**: 위젯은 테이블에 의해 위젯으로 할당된 추가 공간을 차지해야 한다. 이러한 공간은 해당 옵션을 명시하는 모든 자식 위젯들 간 균등하게 할당되어야 한다.
- **GTK\_SHRINK**: 위젯은 렌더링에 충분한 공간만 차지하도록 줄어들어야 한다. 보통 다른 위젯들이 이러한 공간을 차지할 수 있도록 만들기 위해 사용된다.
- **GTK\_FILL**: 추가 공간을 패딩으로 채우는 대신 할당된 공간을 모두 위젯이 채워야 한다.

bitwise 또는 연산자(operator)를 이용함으로써 다수의 부착점 매개변수를 제공하는 것이 가능하다. 예를 들어 **GTK\_EXPAND |GTK\_FILL**을 이용하면 패딩을 추가하는 대신 자식 위젯이 추가 공간을 차지하고 채울 것이다.

gtk\_table\_attach()의 마지막 두 매개변수는 그 자식 위젯과 주변 셀들 사이에 추가되어야 하는 수평 및 수직 패딩의 픽셀을 명시한다.

```
void gtk_table_attach_defaults (GtkTable *table,
                               GtkWidget *child,
                               guint left,
                               guint right,
                               guint top,
                               guint bottom);
```

박스과 마찬가지로 자식 위젯을 추가할 때는 전체 매개변수 집합을 명시하지 않아도 된다. 부착 및 패딩 옵션을 명시하지 않고 gtk\_table\_attach\_defaults()를 이용해 자식을 추가할 수도 있다. 이러한 함수를 이용할 때는 각 부착 옵션마다 **GTK\_EXPAND |GTK\_FILL**이 사용될 것이며, 패딩은 전혀 추가되지 않을 것이다.

## 테이블 간격

`gtk_table_attach()`를 이용하면 행이나 열의 간격을 명시할 수 있지만 GTK+는 자식 위젯을 추가한 후 이러한 수치를 변경하도록 네 가지 방법을 제공한다.

테이블에서 열마다 간격을 설정하고 싶다면 `gtk_table_set_col_spacings()`를 이용하면 된다. 해당 함수는 리스팅 3-4에서 5 픽셀 간격을 추가하는 데에 사용되었다. GTK+는 행 사이에 패딩을 추가하도록 `gtk_table_set_row_spacings()`도 제공한다. 이러한 함수들은 테이블의 기존 설정을 모두 오버라이드할 것이다.

```
void gtk_table_set_col_spacings (GtkTable *table,
                                guint spacing);
```

`gtk_table_set_col_spacing()` 또는 `gtk_table_set_row_spacing()`은 각각 하나의 특정 열과 행의 간격을 설정한다. 이러한 함수들은 자식 위젯과 그 주위 간격을 위젯의 좌우 또는 위아래에 추가한다.

```
void gtk_table_set_col_spacing (GtkTable *table,
                                guint column,
                                guint spacing);
```

## 고정 컨테이너

`GtkFixed` 위젯은 위젯을 픽셀별로 위치시키도록 해주는 타입의 레이아웃 컨테이너다. 해당 위젯을 사용 시에는 많은 문제가 발생할 수 있지만 단점을 확인하기 전에 우선 간단한 예를 살펴보자.

리스팅 3-5는 두 개의 버튼을 포함하는 `GtkFixed` 위젯을 생성하는데, 버튼은 위젯의 상단 좌측 모서리를 기준으로 각각 (0,0)과 (20,30)에서 찾을 수 있다.

리스팅 3-5. 정확한 위치 명시하기 (fixed.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *fixed, *button1, *button2;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Fixed");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    fixed = gtk_fixed_new ();
    button1 = gtk_button_new_with_label ("Pixel by pixel ...");
    button2 = gtk_button_new_with_label ("you choose my fate.");

    g_signal_connect_swapped (G_OBJECT (button1), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              (gpointer) window);
    g_signal_connect_swapped (G_OBJECT (button2), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              (gpointer) window);
```

```

/* Place two buttons on the GtkFixed container. */
gtk_fixed_put (GTK_FIXED (fixed), button1, 0, 0);
gtk_fixed_put (GTK_FIXED (fixed), button2, 20, 30);

gtk_container_add (GTK_CONTAINER (window), fixed);
gtk_widget_show_all (window);

gtk_main ();
return 0;
}

```

gtk\_fixed\_new()로 초기화되는 GtkFixed 위젯은 특정 위치에 특정 크기로 된 위젯을 위치시키도록 해준다. gtk\_fixed\_put()을 이용하면 명시된 가로 및 세로 위치에 위젯을 놓을 수 있다.

```

void gtk_fixed_put (GtkFixed *fixed,
                   GtkWidget *child,
                   gint x,
                   gint y);

```

고정 컨테이너의 상단 좌측 모서리는 위치 (0,0)으로 참조된다. 위치나 위젯에 대한 실제 위치는 양의 공간에 서만 명시할 수 있다. 고정 컨테이너는 자체적으로 크기를 변경할 것이므로 모든 위젯은 완전히 표시된다.

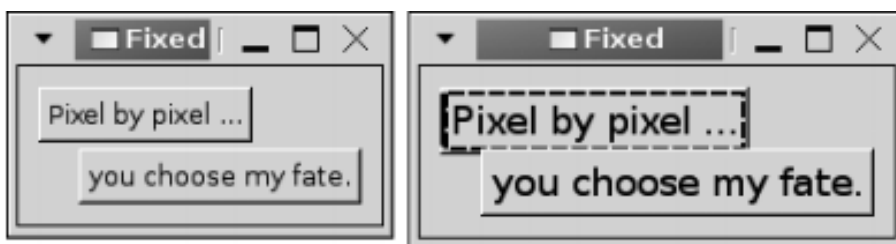
GtkFixed 컨테이너 내에서 위젯을 위치시킨 후 이동해야 하는 경우 gtk\_fixed\_move()를 이용할 수 있다. 이미 위치된 위젯이 겹치지 않도록 주의할 필요가 있겠다. 위젯이 겹치면 GtkFixed 위젯은 알림을 제공하지 않고 대신 창의 렌더링을 시도할 것이므로 그 결과를 예측할 수 없다.

```

void gtk_fixed_move (GtkFixed *fixed,
                    GtkWidget *child,
                    gint x_position,
                    gint y_position);

```

이는 GtkFixed 위젯을 사용 시 불가피한 문제를 야기한다. 첫 번째는 사용자가 원하는 테마를 원하는 대로 사용한다는 점이다. 이는 개발자가 명시적으로 글꼴을 설정하지 않는 한 사용자의 머신에 표시되는 텍스트의 크기가 개발자의 머신에 표시된 텍스트의 크기와 다를 수 있음을 의미한다. 위젯의 크기는 사용자 테마에 따라 다양하다. 따라서 잘못된 정렬(misalignment)과 겹침(overlap)을 야기할 수 있다. 이는 리스팅 3-5의 두 개의 스크린샷을 표시하는 그림 3-6에서 묘사하고 있는데, 하나는 글꼴 크기가 작고 나머지 하나는 크다.



겹침을 피하기 위해 텍스트의 크기와 글꼴을 명시적으로 설정할 수 있지만 대부분의 경우 권하지 않는다. 저시력 사용자를 위해 접근성 옵션이 제공된다. 개발자가 글꼴을 변경하면 일부 사용자는 화면에서 텍스트를 읽을 수 없을 것이다.

GtkFixed 를 사용 시 발생 가능한 또 다른 문제는 애플리케이션을 다른 언어로 해석할 때 발생한다. 사용자 인터페이스가 영어로는 괜찮아 보일지 모르지만 다른 언어로 표시된 문자열은 너비가 일정하지 않아 문제를 표시하기도 한다. 게다가 히브리어와 아랍어와 같이 오른쪽으로 왼쪽으로 읽는 언어는 GtkFixed 위젯을 이용해 적절하게 반영할 수가 없다. 이런 경우 GtkBox 또는 GtkTable과 같은 다양한 크기의 컨테이너를 사용하는 것이 최선의 방법이다.

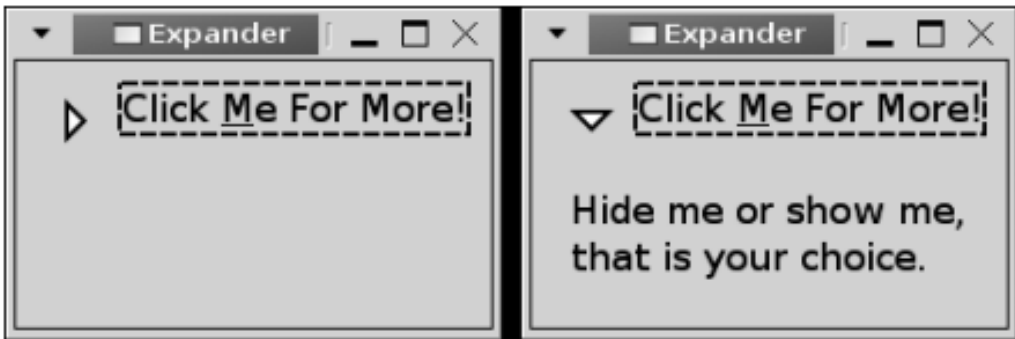
마지막으로, `GtkFixed` 컨테이너를 사용하면 자신의 그래픽 인터페이스에 위젯을 추가하고 그로부터 제거하는 일이 번거롭다. 사용자 인터페이스를 변경하면 자신의 모든 위젯의 위치를 변경해야 할 것이다. 위젯이 많은 애플리케이션을 갖고 있는 경우 이는 장기간 관리 문제를 제시한다.

다른 한편으로는 테이블, 박스, 그 외 자동으로 포맷팅되는 다양한 컨테이너가 있다. 사용자 인터페이스에서 위젯을 추가하거나 제거해야 하는 경우 셀로 추가 및 제거하기가 쉽다. 이는 관리를 훨씬 효율적으로 만들기 때문에 큰 애플리케이션에서 고려할 사항이다.

따라서 위에서 제시한 문제들이 자신의 애플리케이션에 해가 되지 않을 것을 확신할 수 없다면 `GtkFixed` 대신 다양한 크기의 컨테이너를 사용해야 한다. `GtkFixed` 컨테이너는 적절한 상황이 발생할 경우 이용할 수 있음을 알리기 위해 제시되었다. 실사 사용하기에 적절한 상황이라 하더라도 유연한 컨테이너를 사용하는 편이 언제나 더 나은 해결책이자 적절한 작업 처리 방식이 된다.

### 익스펜더

`GtkExpander` 컨테이너는 하나의 자식만 처리할 수 있다. 자식 위젯은 익스펜더의 라벨 왼쪽에 삼각형을 클릭하여 표시/숨김이 가능하다. 해당 액션의 적용 전후 스크린샷을 그림 3-7에서 확인할 수 있다.



리스트 3-6은 그림 3-7을 생성하기 위해 사용되었다. 해당 예제는 가장 중요한 `GtkExpander` 메서드를 소개한다.

리스트 3-6. 위젯 표시하기/숨기기 (`expanders.c`)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *expander, *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Expander");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 200, 100);

    expander = gtk_expander_new_with_mnemonic ("Click _Me For More!");
    label = gtk_label_new ("Hide me or show me, \nthat is your
choice.");

    gtk_container_add (GTK_CONTAINER (expander), label);
    gtk_expander_set_expanded (GTK_EXPANDER (expander), TRUE);
}
```

```

gtk_container_add (GTK_CONTAINER (window), expander);

gtk_widget_show_all (window);

gtk_main ();
return 0;
}
    
```

리스팅 3-6은 GtkExpander를 초기화하기 위해 `gtk_expander_new_with_mnemonic()`을 사용한다. 해당 함수의 initialization 함수에 밑줄을 사용하면 키보드 가속기가 생성될 것이다. 가령, 리스팅 3-6에서는 사용자가 키보드에 Alt+M을 누를 때마다 위젯이 활성화될 것이다. GtkExpander 위젯을 활성화하면 현재 상태에 따라 위젯이 확장 또는 축소될 것이다.

**Tip** 연상기호는 라벨을 표시하는 거의 모든 위젯에서 이용 가능하다. 이용 가능한 곳에서는 항상 이 기능을 사용해야 하는데, 일부 사용자들은 키보드를 이용해 애플리케이션을 훑어보는 편을 선호하기 때문이다.

익스펜더 라벨에 밑줄 문자를 포함시키길 원한다면 두 번째 밑줄과 함께 맨 앞에 붙여야 한다. 연상기호 기능을 활용하고 싶지 않다면 `gtk_expander_new()`를 이용해 표준 문자열을 라벨로 하여 GtkExpander를 초기화할 수 있지만 사용자에게 연상기호를 옵션으로 제공하는 것은 언제나 훌륭한 생각이다. 일반 익스펜더 라벨에서 밑줄 문자는 과잉되진 않지만 다른 문자로서 처리될 것이다.

GtkExpander 위젯 자체는 GtkBin에서 과생되므로 하나의 자식 위젯만 가질 수 있음을 의미한다. 하나의 자식을 보유하는 여느 컨테이너 위젯과 마찬가지로 자식 위젯을 추가하려면 `gtk_container_add()`가 필요하다.

리스팅 3-6에서 필자는 자식 위젯을 기본적으로 visible로 설정하고자 하므로 GtkExpander 위젯을 확장 가능하도록 설정하였다. GtkExpander 컨테이너의 자식 위젯은 `gtk_expander_set_expanded()`를 호출함으로써 표시/숨김이 가능하다.

```

void gtk_expander_set_expanded (GtkExpander *expander,
                                gboolean expanded);
    
```

기본적으로 GTK+는 익스펜더 라벨과 자식 위젯 사이에 간격을 추가하지 않는다. 간격을 픽셀로 추가하려면 `gtk_expander_set_spacing()`을 이용해 패딩을 추가할 수 있다.

```

void gtk_expander_set_spacing (GtkExpander *expander,
                                gint spacing);
    
```

### 핸들 박스

GtkHandleBox 위젯은 또 다른 타입의 GtkBin 컨테이너로, 그 자식들을 마우스로 드래그하여 부모 창에서 제거할 수 있도록 한다.

자식 위젯은 제거되면 decorations가 없는 고유의 창에 위치한다. 위젯이 본래 위치한 곳에는 ghost가 위치한다. 창에 다른 위젯들이 존재한다면 가능할 경우 그들이 빈 공간을 채우도록 크기가 조정될 것이다.

위젯은 보통 툴바와 다른 툴킷 디스플레이를 포함하도록 사용된다. GtkHandleBox 위젯의 예를 그림 3-8에 표시하였다. 이는 창에 추가된 후 제거되는 핸들 박스를 보여준다. 핸들 박스는 본래 위치를 이용해 정렬함으로써 다시 부착할 수 있다.



리스팅 3-7에서는 GtkLabel 자식을 포함하는 GtkHandleBox 위젯을 생성한다. 해당 예제는 GtkHandleBox 클래스를 통해 이용 가능한 모든 프로퍼티를 보여준다.

리스팅 3-7. 분리 가능한 위젯 (handleboxes.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *handle, *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Handle Box");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 200, 100);

    handle = gtk_handle_box_new ();
    label = gtk_label_new ("Detach Me");

    /* Add a shadow to the handle box, set the handle position on the
left and
* set the snap edge to the top of the widget. */
    gtk_handle_box_set_shadow_type (GTK_HANDLE_BOX (handle),
GTK_SHADOW_IN);
    gtk_handle_box_set_handle_position (GTK_HANDLE_BOX (handle),
GTK_POS_LEFT);
    gtk_handle_box_set_snap_edge (GTK_HANDLE_BOX (handle),
GTK_POS_TOP);

    gtk_container_add (GTK_CONTAINER (handle), label);
    gtk_container_add (GTK_CONTAINER (window), handle);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}
```

GtkHandleBox 위젯을 생성할 때는 핸들과 스냅 에지(snap edge)가 어디에 위치할 것인지 결정해야 한다. 핸들은 GtkHandleBox 자식을 그 부모로부터 분리(detach)하기 위해 개발자가 붙잡는 자식 위젯의 면(side)에 있는

영역이다. 핸들 박스를 부모 위젯으로부터 분리하면 본래 위치에 얹은 ghost가 그려진다.

gtk\_handle\_box\_set\_handle\_position() 함수는 핸들의 위치를 설정하는 데에 사용된다. GtkPositionType 목록에는 핸들의 위치와 관련해 네 가지 옵션을 제공한다. 핸들 위치의 기본값은 GTK\_POS\_LEFT로 설정되지만 이는 GTK\_POS\_RIGHT, GTK\_POS\_TOP, GTK\_POS\_BOTTOM을 이용해 어떤 면이든 위치시킬 수 있다.

```
void gtk_handle_box_set_handle_position (GtkHandleBox *handle_box,
                                         GtkPositionType position);
```

핸들 위치를 기반으로 GTK+는 스냅 에지의 위치를 선택하는데, 이 곳에서 핸들 박스는 그 부모로 재부착하기 위해 스스로를 변경한다. 스냅 에지는 분리된 후에 ghost가 표시되는 곳이다.

gtk\_handle\_box\_set\_snap\_edge()를 이용해 스냅 에지에 대한 새 GtkPositionType 값을 명시할 수 있다. 사용자의 혼동을 피하기 위해서는 핸들과 관련된 스냅 에지를 위치시키는 장소에 주의를 기울이는 것이 중요하겠다.

```
void gtk_handle_box_set_snap_edge (GtkHandleBox *handle_box,
                                    GtkPositionType position);
```

예를 들어, 핸들 박스가 GtkVBox 위젯의 위에 있고 핸들이 왼쪽 면에 있을 경우 스냅 에지 위치는 GTK\_POS\_TOP으로 설정해야 한다. 그래야만 ghost가 크기를 조정할 필요 없이 스냅 에지와 같은 장소에 위치할 수 있다.

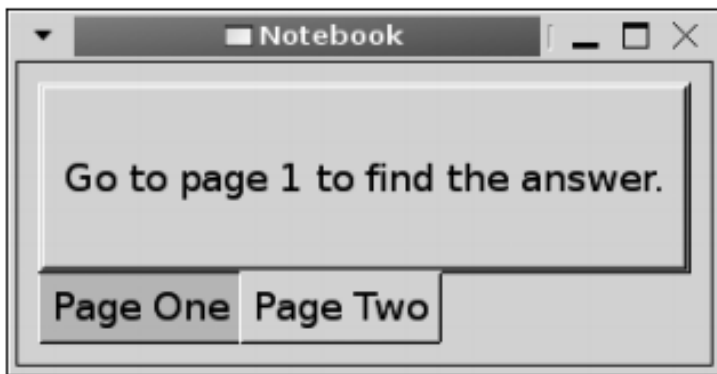
GtkHandleBox는 gtk\_handle\_box\_set\_shadow\_type()이라는 함수도 제공하여 자식 위젯 주변에 위치시킬 테두리 타입을 설정하도록 해준다. GtkShadowType 목록에 대한 값은 아래를 따른다.

- GTK\_SHADOW\_NONE: 자식 위젯 주변에 어떤 테두리도 위치시키지 않을 것이다.
- GTK\_SHADOW\_IN: 테두리가 안쪽으로 치우칠 것이다(skewed inwards).
- GTK\_SHADOW\_OUT: 버튼처럼 테두리가 바깥쪽으로 치우칠 것이다.
- GTK\_SHADOW\_ETCHED\_IN: 테두리가 들어간 3-D 모양일 것이다.
- GTK\_SHADOW\_ETCHED\_OUT: 테두리가 튀어나온 3-D 모양일 것이다.

### 노트북

GtkNotebook 위젯은 자식 위젯을 다수의 페이지에 조직한다. 사용자는 위젯의 한쪽 변에 나타나는 탭을 클릭하여 해당 페이지를 전환할 수 있다.

탭은 기본값으로 상단에 표시되지만 당신이 탭이 표시될 위치를 명시할 수도 있다. 탭을 모두 숨기는 것도 가능하다. 그림 3-9는 리스팅 3-8의 코드로 생성된 두 개의 탭이 있는 GtkNotebook 위젯을 보여준다.



노트북 컨테이너를 생성할 때는 탭마다 탭 라벨 위젯과 자식 위젯을 명시해야 한다. 탭은 앞이나 뒤에서 추가, 삽입, 재정렬, 제거된다.

리스팅 3-8. 다수의 페이지가 있는 컨테이너 (notebooks.c)

```
#include <gtk/gtk.h>

static void switch_page (GtkButton*, GtkNotebook*);
```

```

int main (int argc,
          char *argv[])
{
    GtkWidget *windiw, *notebook;
    GtkWidget *label1, *label2, *child1, *child2;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Notebook");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 250, 100);

    notebook = gtk_notebook_new ();
    label1 = gtk_label_new ("Page One");
    label2 = gtk_label_new ("Page Two");
    child1 = gtk_label_new ("Go to page 2 to find the answer.");
    child2 = gtk_label_new ("Go to page 1 to find the answer.");

    /* Notice that two widgets were connected to the same callback
function! */
    g_signal_connect (G_OBJECT (child1), "clicked",
                     G_CALLBACK (switch_page),
                     (gpointer) notebook);
    g_signal_connect (G_OBJECT (child2), "clicked",
                     G_CALLBACK (switch_page),
                     (gpointer) notebook);

    /* Append to pages to the notebook container. */
    gtk_notebook_append_page (GTK_NOTEBOOK (notebook), child1, label1);
    gtk_notebook_append_page (GTK_NOTEBOOK (notebook), child2, label2);

    gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notebook), GTK_POS_BOTTOM);

    gtk_container_add (GTK_CONTAINER (window), notebook);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* Switch to the next or previous GtkNotebook page. */
static void
switch_page (GtkButton *button,
             GtkNotebook *notebook)
{

```



```

gint page = gtk_notebook_get_current_page (notebook);

if (page == 0)
    gtk_notebook_set_current_page (notebook, 1);
else
    gtk_notebook_set_current_page (notebook, 0);
}
    
```

GtkNotebook을 생성한 후에는 그곳에 탭을 추가하고 나서야 유용해진다. 탭 리스트의 끝이나 시작 부분에 탭을 추가하려면 각각 `gtk_notebook_append_page()` 또는 `gtk_notebook_prepend_page()`를 이용할 수 있다. 각 함수는 자식 위젯인 GtkNotebook과 함께 아래와 같이 탭에 표시되어야 할 위젯을 수락한다.

```

gint gtk_notebook_append_page (GtkNotebook *notebook,
                               GtkWidget *child,
                               GtkWidget *tab_label);
    
```

**Tip** 탭 라벨이 꼭 GtkLabel 위젯일 필요는 없다. 가령, 라벨과 닫기 버튼을 포함하는 GtkHBox 위젯을 사용할 수도 있다. 이는 버튼이나 이미지와 같이 다른 유용한 위젯을 탭 라벨로 내포하도록 허용한다.

각 노트북 페이지는 하나의 자식 위젯만 표시할 수 있다. 하지만 각 자식 위젯은 다른 컨테이너가 될 수도 있으므로 각 페이지는 다수의 위젯을 표시할 수 있다. 사실 GtkNotebook을 다른 GtkNotebook 탭의 자식 위젯으로서 사용하는 것이 가능하다.

**Caution** 노트북 내부에 노트북을 위치시키는 것도 가능하지만 사용자가 쉽게 혼란에 빠질 수 있으므로 주의 를 기울여 실행해야 한다. 꼭 실행해야 한다면 자식 노트북의 탭은 부모 탭과 다른 노트북 면(side)에 위치시키도록 하라. 그래야만 어떤 탭이 어떤 노트북에 속하는지 알아낼 수 있을 것이다.

특정 위치에 탭을 삽입하고자 한다면 `gtk_notebook_insert_page()`를 사용할 수 있다. 해당 함수는 탭의 정수 위치를 명시하도록 허용할 것이다. 삽입된 탭 다음에 위치한 모든 탭의 색인은 1씩 증가할 것이다.

```

gint gtk_notebook_insert_page (GtkNotebook *notebook,
                               GtkWidget *child,
                               GtkWidget *tab_label,
                               gint position);
    
```

GtkNotebook으로 탭을 추가하는 데에 사용된 세 가지 함수 모두 당신이 추가한 탭의 정수 위치를 리턴하고, 액션이 실패하면 -1를 리턴할 것이다.

**GtkNotebook** 프로퍼티

리스트링 3-8에서 GtkNotebook에 대해 `tab-position` 프로퍼티가 설정되었는데, 이는 아래 호출을 통해 이루어진다.

```

void gtk_notebook_set_tab_pos (GtkNotebook *notebook,
                               GtkPositionType position);
    
```

탭 위치는 GtkHandleBox의 핸들 및 스냅 에지 위치를 설정하는 데에 사용했던 GtkPositionType 목록을 이용하여 `gtk_notebook_tab_pos()`에서 설정이 가능하다. 여기에는 `GTK_POS_TOP`, `GTK_POS_BOTTOM`, `GTK_POS_LEFT`, `GTK_POS_RIGHT`가 포함된다.

노트북은 사용자에게 여러 옵션을 제공하길 원하지만 이러한 옵션들을 여러 단계로 표시하길 원할 때 유용하다. 각 탭에 옵션 몇 가지를 위치시키고 `gtk_notebook_set_show_tabs()`를 이용해 탭을 숨기면 사용자 옵션을 왔다갔다 선택할 수 있다. 이러한 개념의 예로 자신의 운영체제를 통해 확인 가능한 수많은 마법사(wizard)를 들 수 있는데, GtkAssistant 위젯이 제공하는 기능과 비슷하겠다.

```
void gtk_notebook_set_show_tabs (GtkNotebook *notebook,
    gboolean show_tabs);
```

어느 시점이 되면 GtkNotebook은 할당된 공간에서 탭을 보관할 공간이 부족하게 될 것이다. 이러한 문제를 해결하기 위해 gtk\_notebook\_set\_scrollable()을 이용해 노트북 탭을 스크롤 가능하게 설정할 수 있다.

```
void gtk_notebook_set_scrollable (GtkNotebook *notebook,
    gboolean scrollable);
```

해당 프로퍼티는 탭을 강제로 사용자에게서 숨길 것이다. 사용자가 탭의 리스트를 스크롤할 수 있도록 화살표가 제공된다. 탭은 하나의 행이나 열에만 표시될 것이기 때문에 꼭 필요하다.

창의 크기를 조정하여 모든 탭이 표시되지 않으면 탭은 스크롤이 가능하게 만들어질 것이다. 또 탭을 모두 그릴 수 없을 만큼 글꼴의 크기가 커도 스크롤이 발생할 것이다. 탭이 할당된 공간 이상을 차지할 가능성이 조금이라도 있다면 해당 프로퍼티는 항상 TRUE로 설정해야 한다.

탭 연산

GTK+는 이미 존재하는 탭과 상호작용할 수 있도록 다수의 함수를 제공한다. 이를 학습하기 전에 우선 이러한 함수들 중 대부분은 change-current-page 시그널을 방출하게 만든다는 사실을 인지하는 것이 유용하다. 해당 시그널은 포커스를 받은 현재 탭이 변경될 때 방출된다.

탭을 추가할 수 있다면 탭을 제거하는 방법도 분명 존재해야 한다. gtn\_notebook\_remove\_page()를 이용해 탭의 색인 참조를 기반으로 탭을 제거할 수 있다. 위젯을 GtkNotebook으로 추가하기 전에 참조 계수를 증가시키지 않았다면 해당 함수는 마지막 참조를 해제(release)하고 자식 위젯을 소멸시킬 것이다.

```
void gtk_notebook_remove_page (GtkNotebook *notebook,
    gint page_number);
```

gtk\_notebook\_reorder\_child()를 호출하여 탭을 수동으로 재정렬할 수도 있다. 이동시키고자 하는 페이지의 자식 위젯과 그것을 이동시킬 위치를 명시해야만 한다. 탭의 개수보다 크거나 음의 숫자를 명시할 경우 탭은 리스트의 끝으로 이동할 것이다.

```
void gtk_notebook_reorder_child (GtkNotebook *notebook,
    GtkWidget *child,
    gint position);
```

현재 페이지를 변경하기 위해 제공하는 방법에는 세 가지가 있다. 확인하길 원하는 페이지의 구체적인 색인을 알고 있다면 gtk\_notebook\_set\_current\_page()를 이용해 해당 페이지도 이동할 수 있다.

```
void gtk_notebook_set_current_page (GtkNotebook *notebook,
    gint page_number);
```

때로는 gtk\_notebook\_next\_page() 또는 gtk\_notebook\_prev\_page()를 호출하여 다음 탭이나 이전 탭으로 전환하길 원할 때도 있다. 두 함수 중 하나로 호출하였는데 현재 탭이 0 보다 작아지거나 현재 탭의 개수보다 많아지는 경우 아무 일도 발생하지 않고 호출은 무시될 것이다.

어떤 페이지로 이동할 것인지 결정할 때는 현재 페이지와 총 탭의 개수를 아는 것이 도움이 되곤 한다. 이러한 값은 각각 gtk\_notebook\_get\_current\_page() 와 gtk\_notebook\_get\_n\_pages()를 이용해 얻을 수 있다.

## 이벤트 박스

GtkLabel을 포함해 다양한 위젯들은 연관된 GDK 창을 갖는다는 이유로 GDK 이벤트에 응답하지 않는다. 이러한 문제를 해결하기 위해 GTK+는 GtkEventBox라고 불리는 컨테이너 위젯을 제공한다. 이벤트 박스는 객체에 대한 GDK 창을 제공함으로써 자식 위젯에 대한 이벤트를 포착(catch)한다.

리스트 3-9는 이벤트 박스를 이용해 button-press-event 시그널을 GtkLabel로 연결한다. 라벨 내 텍스트는 라벨을 더블 클릭할 때 현재 위치를 기반으로 변경된다. 한 번만 클릭하면 시그널은 방출될지언정 눈에 보이는 변화는 발생하지 않는다.

리스트 3-9. GtkLabel로 이벤트 추가하기 (eventboxes.c)

```
#include <gtk/gtk.h>

static gboolean button_pressed (GtkWidget*, GdkEventButton*,
GtkWidget*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *eventbox, *label;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Event Box");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 200, 50);

    eventbox = gtk_event_box_new ();
    label = gtk_label_new ("Double-Click Me!");

    /* Set the order in which widgets will receive notification of
events. */
    gtk_event_box_set_above_child (GTK_EVENT_BOX (eventbox), FALSE);

    g_signal_connect (G_OBJECT (eventbox), "button_press_event",
                     G_CALLBACK
(button_pressed), (gpointer) label);

    gtk_container_add (GTK_CONTAINER (eventbox), label);
    gtk_container_add (GTK_CONTAINER (window), eventbox);

    /* Allow the event box to catch button presses, realize the widget,
and set the
    * cursor that will be displayed when the mouse is over the event
box. */
    gtk_widget_set_events (eventbox, GDK_BUTTON_PRESS_MASK);
    gtk_widget_realize (eventbox);
    gdk_window_set_cursor (eventbox->window, gdk_cursor_new
```

```

(GDK_HAND1));
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* This is called every time a button-press event occurs on the
GtkEventBox. */
static gboolean
button_pressed (GtkWidget *eventbox,
                GdkEventButton *event,
                GtkLabel *label)
{
    if (event->type == GDK_2BUTTON_PRESS)
    {
        const gchar *text = gtk_label_get_text (label);

        if (text[0] == 'D')
            gtk_label_set_text (label, "I Was Double-Clicked!");
        else
            gtk_label_set_text (label, "Double-Click Me Again!");
    }
    return FALSE;
}

```

이벤트 박스를 이용할 때는 이벤트 박스의 `GdkWindow` 가 그 자식 창의 위에 위치할 것인지 아래에 위치할 것인지 결정할 필요가 있다. 이벤트 박스 창이 위에 있는 경우 이벤트 박스 내의 모든 이벤트가 이벤트 박스 위로 이동할 것이다. 창이 아래에 있다면 자식 위젯의 창에 있는 이벤트들이 먼저 해당 위젯으로 이동한 후 그 부모로 이동할 것이다.

■ **Note** 창의 위치를 아래(below)로 설정하면 이벤트는 자식 위젯으로 먼저 이동한다. 하지만 이는 연관된 GDK 창을 가진 위젯에 한한다. 자식이 `GtkLabel` 위젯인 경우 스스로 이벤트를 감지하는 능력이 없다. 따라서 리스팅 3.9에서는 창의 위치를 위로 설정하든 아래로 설정하든 상관없다.

이벤트 박스의 위치는 `gtk_event_box_set_above_child()`를 이용해 그 자식 위젯의 위나 아래로 이동할 수 있다. 해당 프로퍼티의 기본값은 모든 이벤트 박스에 대해 `FALSE`로 설정된다. 즉, 모든 이벤트는 시그널이 처음 방출된 위젯에 의해 처리될 것이란 뜻이다. 그리고 위젯이 완료되고 나면 이벤트는 그 부모로 전달될 것이다.

```

void gtk_event_box_set_above_child (GtkEventBox *event_box,
    gboolean above_child);

```

다음은, 이벤트 마스크를 이벤트 박스로 추가하여 위젯이 수신하게 될 이벤트 타입을 알 수 있도록 해야 한다. 이벤트 마스크를 명시하는 `GdkEventMask` 목록의 값은 표 3-3에 실려 있다. `GdkEventMask` 값에 대한 bitwise 값을 하나 이상 설정해야 하는 경우 그 값을 `gtk_widget_set_events()`로 전달할 수 있다.

값	설명
GDK_EXPOSURE_MASK	위젯이 노출될 때 이벤트를 수락한다.
GDK_POINTER_MOTION_MASK	모든 포인터 모션 이벤트를 수락한다.
GDK_POINTER_MOTION_HINT_MASK	GDK_MOTION_NOTIFY 이벤트 개수를 제한하여 마우스가 움직일 때마다 방출되지 않도록 한다.
GDK_BUTTON_MOTION_MASK	아무 버튼이나 누르는 동안 포인터 모션 이벤트를 수락한다.
GDK_BUTTON1_MOTION_MASK	버튼 1을 누르는 동안 포인터 모션 이벤트를 수락한다.
GDK_BUTTON2_MOTION_MASK	버튼 2을 누르는 동안 포인터 모션 이벤트를 수락한다.
GDK_BUTTON3_MOTION_MASK	버튼 3을 누르는 동안 포인터 모션 이벤트를 수락한다.
GDK_BUTTON_PRESS_MASK	마우스 버튼 누름 이벤트를 수락한다.
GDK_BUTTON_RELEASE_MASK	마우스 버튼 누름해제 이벤트를 수락한다.
GDK_KEY_PRESS_MASK	키보드에서 키 누름 이벤트를 수락한다.
GDK_KEY_RELEASE_MASK	키보드에서 키 누름해제 이벤트를 수락한다.
GDK_ENTER_NOTIFY_MASK	창 근처에 들어갈 때 방출되는 이벤트를 수락한다.
GDK_LEAVE_NOTIFY_MASK	창 근처에서 벗어날 때 방출되는 이벤트를 수락한다.
GDK_FOCUS_CHANGE_MASK	포커스 이벤트의 변경을 수락한다.
GDK_STRUCTURE_MASK	창 설정의 내용이 변경될 때 방출되는 이벤트를 수락한다.
GDK_PROPERTY_CHANGE_MASK	객체 프로퍼티에 대한 변경을 수락한다.
GDK_VISIBILITY_NOTIFY_MASK	시각성(visibility) 이벤트에 대한 변경을 수락한다.
GDK_PROXIMITY_IN_MASK	마우스 커서가 위젯의 근처로 들어갈 때 방출되는 이벤트를 수락한다.
GDK_PROXIMITY_OUT_MASK	마우스 커서가 위젯의 근처에서 벗어날 때 방출되는 이벤트를 수락한다.
GDK_SUBSTRUCTURE_MASK	자식 창의 설정을 변경하는 이벤트를 수락한다.
GDK_SCROLL_MASK	모든 스크롤 이벤트를 수락한다.
GDK_ALL_EVENTS_MASK	모든 타입의 이벤트를 수락한다.

표 3-3. GdkEventMask 값

위젯에서는 `gtk_widget_realize()`를 호출하기 전에 `gtk_widget_set_events()`를 호출"해야만" 한다. 위젯이 이미 GTK+에 의해 실현(realized)된 경우 대신 `gtk_widget_add_events()`를 이용해 이벤트 마스크를 추가할 수 있다. `gtk_widget_realize()`를 호출하기 전에는 개발자의 `GtkEventBox`는 아직 연관된 `GdkWindow`나 다른 GDK 위젯 자원을 갖고 있지 않다. 보통은 부모 위젯이 실현될 때 실현(realization)이 발생하지만 이벤트 박스는 예외다. 위젯에서 `gtk_widget_show()`를 호출하면 자동으로 GTK+에 의해 실현된다. 이벤트 박스는 개발자가 `gtk_widget_show_all()`을 호출할 때 실현되지 않는 것이 정상인데, 이는 `invisible` 상태로 설정되기 때문이다. 이벤트 박스에서 `gtk_widget_realize()`를 호출한다면 이러한 문제를 쉽게 해결할 수 있을 것이다.

개발자는 이벤트 박스를 실현 시 이벤트 박스가 자식으로서 최상위 수준 위젯에 이미 추가되어 있도록 확보해야 하는데, 그렇지 않을 경우 작동하지 않을 것이다. 그 이유는 개발자가 위젯을 실현하면 그 조상 위젯들도 자동으로 실현될 것이기 때문이다. 조상들이 없다면 GTK+는 만족하지 않을 것이며, 실현은 실패할 것이다.

이벤트 박스가 실현되고 나면 연관된 `GdkWindow`를 가질 것이다. `GdkWindow`는 화면에서 위젯이 그려지는 직사각형 영역을 나타내는 클래스다. 이것은 제목 표시줄 등이 포함된 최상위 수준의 창을 참조하는 `GtkWindow`와는 다르다. `GtkWindow`는 자식 위젯마다 하나씩, 수많은 `GdkWindow` 객체를 포함할 것이다. 이들은 화면에 위젯을 그리는 데에 사용된다.

`GtkLabel` 위젯을 클릭할 수 있도록 허용하는 김에 마우스를 라벨 위에서 왔다 갔다하면 마우스가 손모양(hand)으로 변하도록 만들어도 괜찮는데 이는 `gdk_window_set_cursor()`와 `gdk_cursor_new()`를 이용해 실행한

다. GDK에서 이용 가능한 커서 타입에는 여러 가지가 있다. 이용 가능한 커서의 전체 리스트를 확인하려면 API 문서에서 GdkCursorType 목록을 살펴본다.

```
gdk_window_set_cursor (eventbox->window, gdk_cursor_new (GDK_HAND1));
```

**Note** GtkWidget 구조에는 다수의 public member가 포함된다. 그 중 하나는 창(window)으로, 주어진 위젯과 연관된 GdkWindow에 해당한다. 앞의 코드에서 새 커서는 이벤트 박스의 GdkWindow와 연관되어 있었다.

### 자신의 이해도 시험하기

본 장에서는 GTK+에 포함된 여러 컨테이너 위젯을 소개하였다. 아래 실린 두 연습문제는 새롭게 학습한 몇 가지 위젯을 실습하도록 도와준다.

#### 연습문제 3-1. 다수의 컨테이너 사용하기

컨테이너의 중요한 특성 중 하나는 각 컨테이너가 다른 컨테이너들을 보유할 수 있다는 점이다. 이를 명확히 이해하기 위해 이번 예제에서는 수많은 컨테이너를 이용하게 될 것이다. 메인 창은 GtkNotebook과 함께 하단에 두 개의 버튼을 표시할 것이다.

노트북은 4개의 페이지를 가져야 한다. 각 노트북 페이지는 다음 페이지로 이동하는 GtkButton을 포함해야 한다(마지막 페이지의 GtkButton은 첫 페이지를 래핑해야 한다.)

창의 하단 부분을 따라 두 개의 버튼을 생성한다. 첫 번째 버튼은 GtkNotebook에서 이전 페이지로 이동하며 필요 시 마지막 페이지로 래핑한다. 두 번째 버튼을 클릭하면 창을 닫고 애플리케이션을 나가야 한다.

연습문제 3-1은 구현하기에 간단한 애플리케이션이지만 몇 가지 주요점을 설명한다. 첫째, 이것은 GtkVBox와 GtkHBox의 유용성을 보여주고, 어떻게 이들을 함께 사용하여 복잡한 사용자 인터페이스를 생성하는지 보여준다.

GtkTable을 이용해 창의 직계 자손으로 구현하여 위와 동일한 애플리케이션을 구현하는 방법도 있지만 수평 박스를 이용해 하단에 버튼을 정렬하는 편이 훨씬 수월하다. 버튼들은 박스 끝에 패킹되어 있어 박스의 우측면을 따라 정렬되고, 박스를 이용해 구현하는 편이 훨씬 수월하다는 사실을 눈치챌 것이다.

컨테이너는 다른 컨테이너를 포함할 수 있고 또 그래야 한다는 사실을 확인하였을 것이다. 가령 연습문제 3-1에서 GtkWindow는 GtkVBox를 포함하고, 이는 또 GtkHBox와 GtkNotebook을 포함한다. 이러한 구조는 애플리케이션의 크기가 커지면서 점점 더 복잡해지기도 한다.

연습문제 3-1을 완료했다면 연습문제 3-2로 넘어가라. 다음 문제에서는 수직 박스 대신 패인(paned) 컨테이너를 사용하게 될 것이다.

#### 연습문제 3-2. 좀 더 복잡한 컨테이너

이번 연습문제에서는 연습문제 3-1에서 작성한 코드를 확장할 것이다. 버튼으로 된 수평 박스와 노트북을 포함하기 위해 GtkVBox를 사용하는 대신 GtkVPaned 위젯을 생성하라.

이러한 변경 외에도 GtkNotebook 탭을 숨겨 사용자가 버튼을 누르지 않고서는 페이지 간 전환을 할 수 없도록 해야 한다. 이런 경우 페이지가 언제 변경되는지 본인은 알지 못할 것이다. 따라서 GtkNotebook 페이지에 있는 각 버튼은 그 고유의 익스펜더에 의해 포함되어야 한다. 익스펜더 라벨은 노트북 페이지 간 구별을 허용한다.

연습문제 3-2를 완료했다면 GtkBox, GtkPaned, GtkNotebook, GtkExpander를 이용해 연습할 것인데, 이들은 본 저서의 나머지 부분에서 계속 사용될 중요한 컨테이너들이다.

다음 장으로 넘어가기 전에, 이번 장에서 다루었지만 연습문제 3-1과 3-2에 필요하지 않았던 컨테이너 몇 가지를 시험해 볼 수도 있을 것이다. 추후 소개될 장들은 앞에서 소개한 정보를 검토하지 "않을 것이기" 때문에 그러한 실습을 통해 모든 컨테이너의 사용을 연습할 수 있다.

요약

이번 장에서는 두 가지 타입의 컨테이너 위젯, 즉 decorators와 레이아웃 컨테이너를 학습하였다. 본문에서 다룬 decorator 타입으로는 익스펜더, 핸들 박스, 이벤트 박스가 있다. 본문에서 다룬 레이아웃 컨테이너의 타입은 박스, 패인, 테이블, 고정 컨테이너, 노트북이 포함된다.

GtkLabel 이외에도 GDK 이벤트를 처리할 수 없는 위젯들이 있기 때문에 이벤트 박스 컨테이너는 후에 여러 장에서 살펴볼 것이다. 이는 해당 위젯들을 학습할 때 명시될 것이다. 본 장에서 다룬 컨테이너 대부분은 뒤에 여러 장에서 볼 수 있을 것이다.

해당 컨테이너들은 GTK+ 애플리케이션 개발에 필요하긴 하지만 컨테이너에 GtkLabel과 GtkButton 위젯을 표시하는 것만으로는 대부분의 애플리케이션에서 별로 유용하지 (또는 흥미롭지) 않다. 이러한 형태의 애플리케이션은 기본 사용자 상호작용을 넘어 그 이상을 제공하는 데에 거의 수고를 들이지 않는다.

따라서 다음 장에서는 사용자와 상호작용하도록 해주는 위젯을 학습할 것이다. 이러한 위젯으로는 버튼, 토글, 텍스트 엔트리, 스핀 버튼이 있다.

앞서 언급하였듯 제 4장으로 넘어가기 전에 컨테이너 위젯을 먼저 이해해야 한다. 뒤에 소개할 본문에서는 가장 중요한 컨테이너 위젯을 비롯해 이번 장에서 다룬 여러 개념들을 올바르게 이해하고 있다고 가정할 것이다.

Notes

# FoundationsofGTKDevelopment:Chapter 04

---

제 4 장 기본 위젯

기본 위젯

지금까지는 GtkButton을 제외하곤 사용자 상호작용이 가능하도록 설계된 위젯에 대해서는 학습한 내용이 없다. 하지만 이번 장부터는 사용자가 선택하고, 설정을 변경하며, 정보를 입력하도록 해주는 다양한 형태의 위젯을 다룰 것이다.

이러한 위젯으로는 스톱 버튼, 토글 버튼, 체크 버튼, 라디오 버튼, 색상 선택 버튼, 파일 선택자 버튼, 폰트 선택 버튼, 텍스트 엔트리, 숫자 선택 버튼이 있다.

본 장의 마지막에 실린 연습문제는 큰 규모의 애플리케이션에 이러한 위젯을 여러 개 결합할 수 있는 기회를 제공할 것이다.

이번 장에서 익히게 될 주제는 다음과 같다.

- 스톱 항목이 있는 클릭 가능한 버튼을 사용하는 방법
- 체크 버튼과 라디오 버튼을 포함해 토글 타입의 버튼을 사용하는 방법
- 1행의 자유 형식(free-form) 텍스트 입력에 엔트리 위젯을 사용하는 방법
- 정수나 부동 소수점 수 선택에 스핀 버튼을 사용하는 방법
- 이용 가능한 특수화 버튼의 유형

### 스톡 항목 이용하기

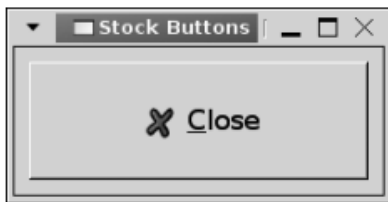
GTK+에서 애플리케이션을 생성하면서 문득 동일한 버튼 및 메뉴 항목을 여러 애플리케이션에 걸쳐 사용하고 있다는 사실을 눈치채기 시작할 것이다. 이 때문에 GTK+는 자주 사용하는 메뉴 항목과 버튼을 수용하는 문자열/이미지 쌍인 스톡(stock) 항목을 제공한다.

GTK+는 `gtk_button_new_from_stock()`을 제공하는데, 이는 미리 정의된 스톡 항목을 이용해 새 버튼을 생성한다. 각 스톡 항목은 버튼에 적용된 연상기호 라벨과 이미지를 포함한다. 스톡 항목의 전체 리스트는 부록 D에서 찾을 수 있다. 각 항목은 수많은 애플리케이션에서 사용되기 때문에 GTK+에 포함되어 있다.

부록 D는 GTK+ 2.10에서 이용 가능한 모든 스톡 아이콘을 포함하지만 애플리케이션을 실행하다보면 아이콘이 자신의 시스템에서 다르게 표시됨을 눈치챌 것이다. 이것은 이용 가능한 스톡 항목들을 항상 보유하고는 있지만 기본 이미지는 사용자의 테마 선택이나 개발자의 선택에 따라 대체될 수 있기 때문이다.

■ **Note** `gtk_button_new_from_stock()` 또는 GTK+에 포함된 다른 스톡 검색 함수에 제공된 스톡 항목을 찾을 수 없는 경우 연상기호 라벨로서 취급될 것이다. 이러한 방법을 통해 버튼이 예측할 수 없는 방식으로 렌더링되는 것을 막을 수 있다.

자신만의 스톡 아이콘을 정의하는 것도 가능하지만 이는 메뉴와 툴바를 주제로 하는 제 9장에서 다룰 것이다. `GTK_STOCK_CLOSE` 스톡 항목을 이용한 버튼의 예제를 그림 4-1에 소개하겠다.



각 스톡 항목은 그것의 문자열 값이나 매크로 정의에 의해 참조될 수 있다. 예를 들어, 리스팅 4-1에서 사용된 `close` 스톡 항목은 `gtk-close` 또는 `GTK_STOCK_CLOSE`로 참조될 수 있다. 하지만 전처리 지시어 (preprocessor directives)는 문자열 값의 편리한 alias에 불과하므로 두 가지 식별자를 모두 학습할 이유는 없다.

■ **Tip** 지원되지 않는 항목은 개발자가 코드를 컴파일할 때 플래그가 지정(flagged)되기 때문에 항상 전처리 지시어를 사용해야 한다. 스톡 항목의 문자열을 이용하는 경우 컴파일러는 오류를 표시(flag)하지 않으므로 유효하지 않은 아이콘이 표시될 것이다.

리스팅 4-1. 스톡 항목 (stockitems.c)

```
button = gtk_button_new_from_stock (GTK_STOCK_CLOSE);

g_signal_connect_swapped (G_OBJECT (button), "clicked",
    G_CALLBACK (gtk_widget_destroy),
    (gpointer) window);
```

GTK+ 2.10 배포판부터는 98개의 스톡 항목을 제공하고 있다. 이러한 항목의 리스트는 부록 D에 실려 있다. 제 9장에서 메뉴와 툴바를 다루면서 이러한 스톡 항목을 다시 사용해 보겠다.

GTK+ 2.0 배포판부터 일부 스톡 항목이 추가되었기 때문에 가장 최신 버전의 GTK+를 실행하지 않으면 몇 가지 항목을 이용할 수 없음을 기억해야 한다. 이는 새로운 애플리케이션을 생성 시 유념해야 한다. 애플리케이션의 사용자들은 최신 GTK+ 버전을 사용하지 않을지도 모르기 때문이다!



### 토글 버튼

GtkToggleButton 위젯은 클릭 시 활성화(active) 또는 비활성화(inactive) 상태를 보유하는 GtkButton 타입이다.

활성화되면 누른 상태로 표시된다. 활성화된 토글 버튼을 클릭하면 일반 상태로 돌아갈 것이다.

GtkToggleButton에서 파생된 위젯에는 두 가지, GtkCheckButton과 GtkRadioButton이 있다.

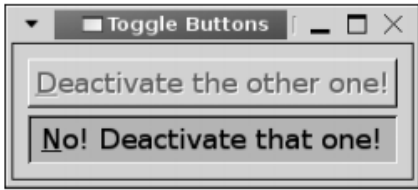
세 가지 함수 중 하나를 이용해 새 GtkToggleButton을 생성할 수 있다. 빈 토글 버튼을 생성하려면

gtk\_toggle\_button\_new()를 이용한다. 기본적으로 토글 버튼이 라벨을 포함하길 원한다면

gtk\_toggle\_button\_new\_with\_label()을 이용한다. 마지막으로 GtkToggleButton은

gtk\_toggle\_button\_new\_with\_mnemonic()을 이용해 연상기호 라벨을 지원하기도 한다.

그림 4-2는 gtk\_toggle\_button\_new\_with\_mnemonic() initializer를 호출함으로써 두 개의 연상기호 라벨과 함께 생성된 두 개의 GtkToggleButton 위젯을 보여준다. 스크린샷의 위젯은 리스팅 4-2에 실린 코드로 생성되었다.



리스팅 4-2의 예제에서 하나의 토글 버튼이 활성화되면 다른 하나는 이용할 수 없게 된다. 이를 sensitive로 만드는 유일한 방법은 원본 토글 버튼을 비활성화하는 것이다.

리스팅 4-2. 토글 버튼 사용하기 (togglebuttons.c)

```
#include <gtk/gtk.h>

static void button_toggled (GtkToggleButton*, GtkWidget*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *vbox, *toggle1, *toggle2;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Toggle Buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    vbox = gtk_vbox_new (TRUE, 5);
    toggle1 = gtk_toggle_button_new_with_mnemonic ("_Deactivate the
other one!");
    toggle2 = gtk_toggle_button_new_with_mnemonic ("_No! Deactivate
that one!");
    g_signal_connect (G_OBJECT (toggle1), "toggled",
                     G_CALLBACK (button_toggled),
                     (gpointer) toggle2);
    g_signal_connect (G_OBJECT (toggle2), "toggled",
                     G_CALLBACK (button_toggled),
                     (gpointer) toggle1);

    gtk_box_pack_start_defaults (GTK_BOX (vbox), toggle1);
```

```

gtk_box_pack_start_defaults (GTK_BOX (vbox), toggle2);

gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window),

gtk_main ();
return 0;
}

/* If the toggle button was activated, set the other as disabled.
Otherwise,
* enable the other toggle button. */
static void
button_toggled (GtkToggleButton *toggle,
                GtkWidget *other_toggle)
{
    if (gtk_toggle_button_get_active (toggle))
        gtk_widget_set_sensitive (other_toggle, FALSE);
    else
        gtk_widget_set_sensitive (other_toggle, TRUE);
}

```

`GtkToggleButton` 클래스가 추가한 유일한 시그널은 `toggled`로, 사용자가 버튼을 활성화하거나 비활성화할 때 방출된다. 해당 시그널은 리스팅 4-2에서 하나의 버튼이 나머지 버튼을 이용할 수 없도록 만들기 위해 트리거한다.

리스팅 4-2에는 또 다른 정보, 즉 다수의 위젯이 동일한 콜백 함수를 사용할 수 있다는 정보가 표시되었다. 각 토크 버튼마다 굳이 구분된 콜백 함수를 생성할 필요는 없었는데, 이는 각각이 동일한 기능을 필요로 했기 때문이다. 하나의 시그널을 다수의 콜백 함수로 연결하는 것도 가능하지만 권하진 않는다. 대신 하나의 콜백 함수에 전체 기능을 구현하면 된다.

#### 위젯 플래그 관리하기

한 가지 중요한 위젯의 특성으로, `disabled` 또는 `inactive`되는 기능을 들 수 있다. 이는 `sensitive` 프로퍼티에 의해 관리되는데, `gtk_widget_set_sensitive()`를 이용해 `FALSE`로 설정하면 위젯을 비활성화시킬 것이다.

```

gtk_widget_set_sensitive().

gtk_widget_set_sensitive (other_toggle, FALSE);

```

감도(sensitivity)는 사실 `GtkWidgetFlags` 목록에서 제공하는 많은 위젯 플래그 중 하나에 불과하다. 아래에 열거된 위젯 플래그들은 `GTK_WIDGET_SET_FLAGS()`를 이용해 설정하고, `GTK_WIDGET_USESET_FLAGS()`를 이용해 비활성화할 수 있다. `GTK_WIDGET_FLAGS()`를 이용해 위젯에 설정된 플래그 리스트를 얻을 수도 있다.

- `GTK_TOPLEVEL`: 위젯이 부모 위젯을 갖지 않는다. 이는 주로 창이나 메뉴와 같은 위젯을 대상으로 설정된다. 해당 플래그는 최상위 수준의 위젯의 수명(lifetime)에 걸쳐 항상 설정되어야 한다.
- `GTK_NO_WINDOW`: 위젯이 고유의 `GdkWindow`를 갖고 있지 않아 부모의 `GdkWindow`로 그리기(drawing)가 이루어진다. 해당 플래그를 이용하면 위젯이 GDK 이벤트를 포착하기 위해 `GtkEventBox`를 필요로 하는지 검사할 수 있다.

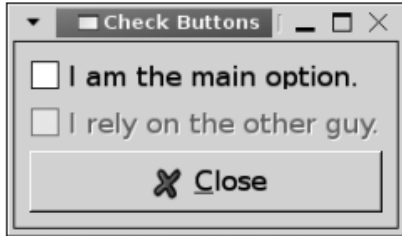
- **GTK\_REALIZED**: `gtk_widget_realize()`를 이용해 위젯이 실현되었다. 위젯을 실현취소(`unrealize`) 할 경우 해당 플래그는 자동으로 `unset`될 것이다.
- **GTK\_MAPPED**: `gtk_widget_map()`을 이용해 위젯이 매핑되었다. 기본적으로 부모 위젯이 `visible`하면 해당 위젯이 사용자에게 표시된다는 의미다.
- **GTK\_VISIBLE**: 해당 플래그는 사용자가 위젯을 볼 수 있다는 의미는 아니지만 사용자가 부모 위젯을 볼 수 있을 경우에만 자식 위젯도 볼 수 있을 것이란 의미다.
- **GTK\_SENSITIVE**: 위젯이 사용자와 상호작용이 가능하고, 버튼이나 키누름 이벤트와 같은 특정 이벤트를 수신할 수 있다.
- **GTK\_PARENT\_SENSITIVE**: 위젯이 `sensitive`로 설정되기 위해서는 위젯의 부모도 `sensitive`해야 한다. 따라서 **GTK\_SENSITIVE**는 해당 프로퍼티에 따라 좌우된다.
- **GTK\_CAN\_FOCUS**: 요청 시 위젯은 포커스를 잡을 수 있다.
- **GTK\_HAS\_FOCUS**: 위젯에 포커스가 있으며 `gtk_widget_grab_focus()`를 이용해 설정 가능하다. 해당 프로퍼티는 **GTK\_CAN\_FOCUS**에 의존한다.
- **GTK\_CAN\_DEFAULT**: 위젯이 창의 기본 위젯이 될 수 있다.
- **GTK\_HAS\_DEFAULT**: 위젯이 창의 기본 위젯이다. `gtk_widget_grab_default()`를 이용해 기본 위젯을 설정할 수 있다.
- **GTK\_HAS\_GRAB**: 위젯이 `grab` 위젯의 스택에 위치하고, 이벤트를 수신하는 데 대한 개인설정(`preference`)을 표시한다.
- **GTK\_RC\_STYLE**: GTK+가 자원(RC) 정의에서 위젯의 스타일을 검색하였다. 위젯에 어떤 스타일도 발견할 수 없는 경우 `even`으로 설정 가능하다.
- **GTK\_COMPOSITE\_CHILD**: 해당 위젯은 그 부모 위젯의 구현에 관한 세부 내용을 제공하고, 사용자에게 표시되어선 안 된다.
- **GTK\_APP\_PAINTABLE**: 설정 시 애플리케이션은 위젯에 그리기가 가능해진다. 이는 GTK+가 현재 내용을 덮어쓰는 것을 예방한다.
- **GTK\_RECEIVES\_DEFAULT**: 설정 시 위젯이 창의 기본 위젯에 해당하지 않더라도 자동으로 기본 액션을 수신할 것이다.
- **GTK\_DOUBLE\_BUFFERED**: 위젯이 사용자에게 노출되면 이중 버퍼링(`double-buffered`)되어야 한다. 이는 사용자를 위해 창의 하나의 단계에서 업데이트되도록 도와주기 때문에 겹으로 보기에 더 순조로워 보인다.
- **GTK\_NO\_SHOW\_ALL**: 이 플래그를 설정할 경우, `gtk_widget_show_all()`을 호출해도 위젯에 영향을 미치지 않을 것이다. 위젯을 수동으로 표시할 필요가 있다. 이는 위젯이 나머지 애플리케이션과 함께 표시되는 것을 막는다.

`toggled` 시그널이 방출되면 토글 버튼이 활성화되었는지 확인하길 원할 것인데, 위젯이 활성화될 때나 비활성화될 때 모두 시그널이 방출되기 때문이다. 이는 `gtk_toggle_button_get_active()`를 이용해 실행이 가능하다. 버튼이 활성화된 경우 `TRUE`가 리턴되고, 비활성화된 경우 `FALSE`를 리턴한다. 토글 버튼의 현재 상태는 `gtk_toggle_button_set_active()`를 이용해 설정하는 수도 있다.

```
void gtk_toggle_button_set_active (GtkToggleButton *toggle,
                                   gboolean active);
```

체크 버튼

대부분의 경우 GtkToggleButton 위젯의 사용을 원치 않을 것인데, 일반 GtkButton 버튼과 정확히 동일하게 생겼기 때문이다. 대신 GTK+는 디스플레이 텍스트 옆에 별개의 토글을 위치시키는 GtkCheckButton 위젯을 생성한다. GtkCheckButton은 GtkToggleButton 클래스에서 파생된다. 해당 위젯의 두 예를 그림 4-3에서 확인할 수 있다.



토글 버튼과 마찬가지로 GtkCheckButton 초기화에 3개의 함수가 제공되는데, gtk\_check\_button\_new(), gtk\_check\_button\_new\_with\_label(), gtk\_check\_button\_new\_with\_mnemonic()이 그것들이다. GtkCheckButton은 또한 중요한 toggled 시그널을 상속하는데, 리스팅 4-3에서 사용되었다.

리스팅 4-3. 체크 버튼 상호작용 (checkbuttons.c)

```
#include <gtk/gtk.h>

static void check_toggled (GtkToggleButton*, GtkWidget*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *vbox, *check1, *check2, *close;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Check Buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    check1 = gtk_check_button_new_with_label ("I am the main option.");
    check2 = gtk_check_button_new_with_label ("I rely on the other
guy.");

    /* Only enable the second check button when the first is enabled.
*/
    gtk_widget_set_sensitive (check2, FALSE);
    g_signal_connect (G_OBJECT (check1), "toggled",
                     G_CALLBACK (check_toggled),
                     (gpointer) check2);

    close = gtk_button_new_from_stock (GTK_STOCK_CLOSE);
    g_signal_connect_swapped (G_OBJECT (close), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              (gpointer) window);
}
```

```

vbox = gtk_vbox_new (FALSE, 5);
gtk_box_pack_start (GTK_BOX (vbox), check1, FALSE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (vbox), check2, FALSE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (vbox), close, FALSE, TRUE, 0);

gtk_container_add (GTK_CONTAINER (window), table);
gtk_widget_show_all (window);

gtk_main ();
return 0;
}

/* If the main check button is active, enable the other. Otherwise,
disable
the supplementary check button. */
static void
check_toggled (GtkToggleButton *check1,
               GtkWidget *check2)
{
    if (gtk_toggle_button_get_active (check1))
        gtk_widget_set_sensitive (check2, TRUE);
    else
        gtk_widget_set_sensitive (check2, FALSE);
}

```

체크 박스와 관련해 초기화 메서드를 제외한 모든 기능은 `GtkToggleButton` 클래스와 그 조상 클래스에서 구현된다. `GtkCheckButton`은 편의(convenience) 위젯에 불과하여 표준 `GtkButton` 위젯과 그래픽 차이만 제공할 뿐이다.

### 라디오 버튼

`GtkToggleButton`에서 파생된 두 번째 위젯 타입은 라디오 버튼 위젯이다. 사실 `GtkRadioButton`은 `GtkCheckButton`에서 파생된다. 라디오 버튼은 일반적으로 그룹화가 가능한 토글이다.

그룹에서 라디오 버튼 하나를 선택하면 나머지는 모두 선택해제될 것이다. 그룹은 한 번에 다수의 라디오 버튼을 선택하는 것을 금한다. 이는 하나의 버튼만 선택되어야 하는 곳에 다수의 옵션을 제공할 수 있도록 해준다.

■ **Note** 라디오 버튼을 선택해제하는 버튼은 GTK+에서 제공하지 않으므로 하나의 라디오 버튼으로 이루어진 그룹은 바람직하지 않다. 사용자가 옵션을 선택해제할 수 없기 때문이다! 하나의 버튼만 필요한 경우 `GtkCheckButton`이나 `GtkToggleButton` 위젯을 사용해야 한다.

라디오 버튼은 라벨 위젯의 면에서 구분된 원형 토글로서 그려지기 때문에 다른 타입의 토글 버튼과 구별할 수 있다. 라디오 버튼은 `GtkCheckButton`과 같은 토글로서 그리는 것도 가능하지만 사용자를 혼란스럽고 당황스럽게 만들 수 있기 때문에 실행해서는 안 된다. 수직 박스에 위치한 4개의 라디오 버튼 그룹은 그림 4-4에 보이는 바와 같다.



라디오 버튼이 올바르게 작동하기 위해서는 모두 그룹 내 다른 라디오 버튼으로 참조되어야 한다. 그렇지 않을 경우 모든 버튼은 독립적인 토글 버튼으로서 행동할 것이다. 다수의 라디오 버튼을 사용하는 방법의 예는 리스팅 4-4에서 확인할 수 있다.

리스팅 4-4. 이기적인 토글 버튼 (radiobuttons.c)

```
#include <gtk/gtk.h>

int main (int argv,
          char *argv[])
{
    GtkWidget *window, *vbox, *radio1, *radio2, *radio3;

    gtk_init (&argv, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Radio Buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Create three radio buttons where the second two join radio1's
    group. */
    radio1 = gtk_radio_button_new_with_label (NULL, "I want to be
    clicked!");
    radio2 = gtk_radio_button_new_with_label_from_widget
    (GTK_RADIO_BUTTON (radio1), "Click me instead!");
    radio3 = gtk_radio_button_new_with_label_from_widget
    (GTK_RADIO_BUTTON (radio1), "No! Click me!");

    /* Note : The radio button you create the new widget from does not
    matter as
    * long as it is already a member of the group! */
    radio4 = gtk_radio_button_new_with_label_from_widget
    (GTK_RADIO_BUTTON (radio3), "No! Clickme instead!");

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), radio1);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), radio2);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), radio3);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), radio4);

    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show_all (window);
}
```

```

gtk_main ();
return 0;
}

```

그룹에서 첫 번째 라디오 버튼은 아래 세 가지 함수 중 하나를 이용해 생성이 가능하다. 하지만 `GtkLabel` 위젯을 자식으로 사용하길 원한다면 연상기호 위젯을 사용하는 것도 가능하기 때문에 토글을 키보드에서 활성화시킬 수 있다.

```

GtkWidget* gtk_radio_button_new (GSList *group);
GtkWidget* gtk_radio_button_new_with_label (GSList *group,
const gchar *label);
GtkWidget* gtk_radio_button_new_with_mnemonic (GSList *group,
const gchar *label);

```

각 호출마다 라디오 그룹에 대해 `NULL`이 명시되어 있음을 눈치챌 것이다. 라디오 버튼의 그룹을 간단하게 생성하기 위해서는 이들을 그룹 내 다른 위젯으로 연관시켜야 하기 때문이다. 해당 방법을 이용하면 단일 연결된 리스트가 있는 `GLIB`의 사용을 피할 수 있는데, 리스트가 자동으로 생성 및 관리될 것이기 때문이다. (`GSList` 데이터 구조는 제 5장과 6장에서 후에 다룰 것이다.)

라벨 없이 라디오 버튼을 포함해 어떤 타입의 토글 버튼이든 생성할 수 있는데, 이런 경우 `gtk_container_add()`를 이용해 고유의 위젯을 추가할 수 있을 것이다. 프로그램적으로 정의된 라벨이나 연상기호 라벨이 있는 라디오 버튼도 생성 가능하다.

나머지 라디오 버튼을 생성하는 가장 쉬운 방법은 다음 3개의 `_from_widget()` 중 하나를 이용하면 된다. 첫 번째 라디오 버튼을 생성하는 방법과 비슷하게 이 버튼들 또한 라벨, 연상기호 라벨과 함께 생성되거나 초기 자식 위젯이 없이 생성될 수 있다.

```

GtkWidget* gtk_radio_button_new_from_widget (GtkRadioButton *group);
GtkWidget* gtk_radio_button_new_with_label_from_widget (GtkRadioButton
*group,
const gchar *label);
GtkWidget* gtk_radio_button_new_with_mnemonic_from_widget
(GtkRadioButton *group,
const gchar *label);

```

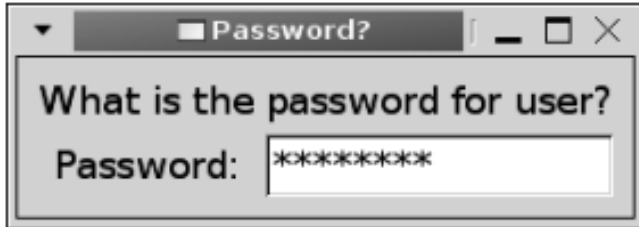
이미 존재하는 라디오 버튼에 초기화 함수를 참조하면 이들 중 하나를 생성한다. `GTK+`는 명시된 위젯으로부터 그룹으로 새 라디오 버튼을 추가할 것이다. 이 때문에 바람직한 라디오 그룹 내에 이미 존재하는 위젯으로만 참조해야 한다.

마지막으로, 그룹 내 모든 라디오 버튼은 `toggled` 시그널로 연결되어야 한다. 라디오 버튼이 선택되면 두 개의 버튼만 `toggled` 시그널을 방출할 것인데, 하나는 선택되고 하나는 선택해제될 것이기 때문이다. 라디오 버튼을 모두 `toggled`로 연결하지 않으면 모든 라디오 버튼 시그널을 포착할 수 없을 것이다.

### 텍스트 엔트리

GtkEntry 위젯은 단일 행으로 된 자유 형식의 텍스트 엔트리 위젯이다. 이는 일반적인 방식으로 구현되므로 많은 타입의 해답에 들어맞도록 만들 수 있다. 텍스트 엔트리, 비밀번호 엔트리, 심지어 숫자 선택에도 사용 가능하다.

GtkEntry는 GtkEditable 인터페이스도 구현하는데, 이것은 텍스트의 선택을 처리하도록 생성된 수많은 함수를 제공한다. 그림 4-5에 GtkEntry 위젯의 사용 예를 실었다. 해당 텍스트 엔트리는 비밀번호 엔트리에 사용된다.



**Note** GtkEditable은 인터페이스라고 불리는 특별한 타입의 객체다. 인터페이스는 다수의 위젯이 구현하는 APIs의 집합으로서, 일관성을 위해 사용된다. 인터페이스의 구현 방법과 개발자의 위젯에서 이를 활용하는 방법은 제 11장에서 다룰 것이다.

GtkEntry 위젯은 모든 텍스트를 표준 문자열로 간주한다. 일반 텍스트와 비밀번호의 유일한 차이는 비가시성 (invisibility) 문자라고 불리는 특수 문자가 비밀번호 내용 대신 표시된다는 점이다. 리스팅 4-5는 비밀번호 엔트리에 GtkEntry 위젯을 사용하는 방법을 보여준다. 일반 텍스트 엔트리에 GtkEntry 위젯을 사용하길 원할 경우 visibility만 켜면 된다.

리스팅 4-5. 사용자 정보 검색하기 (entries.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *vbox, *hbox, *question, *label, *pass;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Password?");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    str = g_strconcat ("What is the password for ", g_get_user_name(),
                      "?", NULL);
    question = gtk_label_new (str);
    label = gtk_label_new ("Password:");

    /* Create a new GtkEntry widget and hide its content from view. */
    pass = gtk_entry_new ();
    gtk_entry_set_visibility (GTK_ENTRY (pass), FALSE);
    gtk_entry_set_invisible_char (GTK_ENTRY (pass), '*');

    hbox = gtk_hbox_new (FALSE, 5);
```



```

gtk_box_pack_start_defaults (GTK_BOX (hbox), label);
gtk_box_pack_start_defaults (GTK_BOX (hbox), pass);

vbox = gtk_vbox_new (FALSE, 5);
gtk_box_pack_start_defaults (GTK_BOX (vbox), question);
gtk_box_pack_start_defaults (GTK_BOX (vbox), hbox);

gtk_container_add (GTK_CONTAINER (window), vbox);
7gtk_widget_show_all (window);

gtk_main ();
return 0;
}

```

### 엔트리 프로퍼티

GtkEntry 위젯은 매우 유연한데, 그 이유는 가장 많은 사례에서 사용되도록 설계되었기 때문이다. 클래스가 제공하는 광범위한 프로퍼티 집합에서 찾아볼 수 있다. 그 중에서 가장 중요한 사용의 예제를 이번 절에 포함시켰다. 프로퍼티 전체 리스트는 부록 A를 참고한다.

GtkEntry 위젯에서 텍스트가 항상 편집 가능해야 하는 것은 아니다. `gtk_editable_set_editable()`을 이용해 엔트리를 편집 불가능하게 설정하면 사용자는 텍스트를 편집할 수 없을 것이다. 하지만 사용자는 여전히 GtkEntry 위젯의 선택 및 복사 기능은 사용할 수 있을 것인데, 이 프로퍼티는 위젯을 *insensitive*하게 설정하는 것과 동일한 효과를 갖기 때문이다.

```

void gtk_editable_set_editable (GtkEditable *editable,
                                gboolean is_editable);

```

종종 값의 문자열 제한 때문에 엔트리 위젯에 입력한 자유 형식의 텍스트 길이를 제한하길 원하는 경우가 있을 것이다. 아래 함수 프로토타입에서 `gtk_entry_set_max_length()`는 엔트리의 텍스트를 `max_length` 문자로 제한한다. 이는 사용자명, 비밀번호, 또는 길이에 민감한 정보의 길이를 제한할 때 유용할 것이다.

```

void gtk_entry_set_max_length (GtkEntry *entry,
                               gint max_length);

```

비가시성 문자는 GTK+에서 비밀번호 엔트리 사용을 가능하게 한다. 비가시성 문자는 엔트리에 실제 비밀번호 내용을 대체하게 될 문자로, `gtk_entry_set_invisible_char()`를 이용해 설정 가능하다. 엔트리에 대한 기본값은 별표이다.

```

void
gtk_entry_set_invisible_char (GtkEntry *entry,
                              guchar inv_char);
void gtk_entry_set_visibility (GtkEntry *entry,
                              gboolean visible);

```

비가시성 문자를 명시한 후에는 `gtk_entry_set_visibility()`를 이용해 `visibility`를 `FALSE`로 설정하여 입력된 텍스트를 모두 숨길 수 있다. 숨겨져 있더라도 엔트리의 실제 내용은 프로그램적으로 검색이 가능하다.

### GtkEntry 위젯으로 텍스트 삽입하기

GtkEntry 위젯으로 텍스트를 삽입하는 방법에는 여러 가지가 있다. 그 중 가장 간단한 방법은 `gtk_entry_set_text()`를 이용하는 방법인데, 이를 사용하면 텍스트 엔트리 내용을 주어진 문자열로 오버라이드 할 것이다. 하지만 위젯이 표시하는 현재 텍스트를 더 이상 신경 쓰지 않을 때에만 유용하다.

```
void gtk_entry_set_text (GtkEntry *entry,
                        const gchar *text);
```

GtkEntry가 표시하는 현재 텍스트는 `gtk_entry_get_text()`를 이용해 검색이 가능하다. 이 문자열은 위젯에 의해 내부적으로 사용되며 어떤 방식으로든 "절대로" free되거나 수정되어선 안 된다.

`gtk_editable_insert_text()`를 이용해 GtkEntry 위젯으로 텍스트를 삽입하는 방법도 있다. 해당 함수는 삽입해야 할 텍스트의 길이를 바이트로 수락하고, 텍스트가 삽입되어야 할 위치를 수락한다.

```
void gtk_editable_insert_text (GtkEditable *editable,
                              const gchar *text,
                              gint length_of_text,
                              gint *position);
```

텍스트를 앞에 추가(`prepend`)하거나 뒤에 추가(`append`)하기 위한 함수들도 있지만 꼭 필요한 것은 아닌데, `gtk_editable_insert_text()`로 각각 0과 -1의 위치를 제공함으로써 해당 함수들을 실행할 수 있기 때문이다.

### GtkEntry Text 조작하기

`gtk_editable_delete_text()`를 이용하면 텍스트 엔트리에서 특정 내용을 쉽게 삭제할 수 있다. 이는 명시된 두 위치 사이의 모든 텍스트를 제거하지만 끝 위치에 있는 문자는 제거하지 않을 것이다.

```
void gtk_editable_delete_text (GtkEditable *editable,
                              gint start_pos,
                              gint end_pos);
```

`gtk_editable_delete_text()`를 이용 시 개발자가 위치를 명시하는 순서는 중요하지 않다. 그리고 개발자가 끝 위치를 -1으로 명시하면 텍스트의 시작부터 끝 위치까지 문자들이 삭제될 것이다.

텍스트의 특정 영역을 자동 선택되도록 만들고 싶다면 `gtk_editable_select_region()`을 이용할 수 있다. 텍스트를 삭제할 때와 마찬가지로 끝 위치에 -1을 명시하면 내용 시작부터 끝까지 모든 텍스트를 선택할 것이다. 수동 및 자동 선택은 아래와 같이 몇 가지 함수를 가능하게 한다.

```
void gtk_editable_select_region (GtkEditable *editable,
                              gint start_pos,
                              gint end_pos);
```

텍스트를 선택할 수 있으니 이제 선택내용을 삭제할 수 있게 된다면 유용하겠다. 이는

`gtk_editable_delete_selection()`을 이용하면 쉽게 가능하다. 해당 함수는 선택된 텍스트를 모두 삭제하고, 선택되지 않은 텍스트를 모두 남겨둘 것이다.

```
void gtk_editable_delete_selection (GtkEditable *editable);
```

위젯의 텍스트 내용을 모두 검색하는 기능에 더해 `gtk_editable_get_chars()`를 이용하면 텍스트의 선택내용만 검색할 수 있다. 이는 명시된 문자열의 복사본을 리턴할 것인데, 사용을 마치면 `g_free()`를 이용해 해제(`free`)시켜야만 한다.

```
gchar* gtk_editable_get_chars (GtkEditable *editable,
                              gint start_pos,
                              gint end_pos);
```

아래 3개의 함수는 다양한 클립보드 함수를 실행한다. 기본적으로는 엔트리에 자르기(Ctrl+X), 복사하기(Ctrl+C), 붙여넣기(Ctrl+V)를 내장시키는 키보드 가속기들이 있다. 따라서 보통 GtkEntry 위젯을 사용할 때는 클립보드 기능을 구현할 필요가 없다.

```
void gtk_editable_cut_clipboard (GtkEditable *editable);
void gtk_editable_copy_clipboard (GtkEditable *editable);
void gtk_editable_paste_clipboard (GtkEditable *editable);
```

### 스핀 버튼

GtkSpinButton 위젯은 숫자 선택 위젯으로 정수와 부동 소수점 수를 처리할 수 있다. GtkEntry에서 파생되므로 GtkSpinButton은 그 함수와 시그널을 모두 상속한다.

### Adjustment

GtkSpinButton 위젯을 다루기 전에 GtkAdjustment 클래스를 이해해야 한다. GtkAdjustment는 GTK+에서 위젯으로 간주되지 않는 몇 안되는 클래스들 중 하나인데, GObject에서 직접 파생되기 때문이다. 스핀 버튼, 뷰포트, GtkRange에서 파생된 다수의 위젯을 포함해 여러 위젯에 사용된다.

새로운 adjustment는 gtk\_adjustment\_new()를 이용해 생성되지만 초기화 시 GTK\_ADJUSTMENT()를 이용해 형변환(cast)되는 것이 보통인데, GObject로서 저장공간은 실용적이지 않기 때문이다. 위젯으로 추가되고 나면 adjustment의 메모리 관리는 위젯이 관리하므로 객체의 이러한 측면에 대해서는 염려하지 않아도 된다.

```
GtkObject* gtk_adjustment_new (gdouble initial_value,
                               gdouble lower_range,
                               gdouble upper_range,
                               gdouble step_increment,
                               gdouble page_increment,
                               gdouble page_size);
```

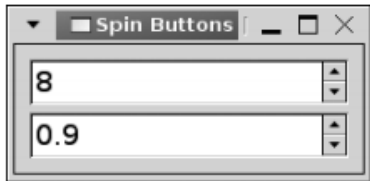
새로운 adjustment는 6개의 매개변수로 초기화되며, 그 리스트는 아래와 같다.

- initial\_value: 초기화될 때 adjustment에 의해 값이 보관된다. 이는 GtkAdjustment 클래스의 value 프로퍼티에 해당한다.
- lower\_range: adjustment가 보유하도록 허용하는 최소값이다. 이는 GtkAdjustment 클래스의 lower 프로퍼티에 해당한다.
- upper\_range: adjustment가 보유하도록 허용하는 최대값이다. 이는 GtkAdjustment 클래스의 upper 프로퍼티에 해당한다.
- step\_increment: 최소의 변경을 가능하게 만드는 increment. 1부터 10까지 모든 정수를 세고 싶다면 increment는 1로 설정될 것이다.
- page\_size: 페이지의 크기. GtkSpinButton에서 이 값은 별로 사용되지 않으므로 page\_increment와 같은 값 또는 0으로 설정되어야 한다.

GtkAdjustment 클래스가 제공하는 유용한 시그널로 2개가 있는데, changed와 value-changed이다. changed 시그널은 adjustment 프로퍼티 중 value 프로퍼티를 제외하고 하나 또는 그 이상이 수정될 때 방출된다. value-changed 시그널은 adjustment의 현재 값이 수정되면 방출된다.

스핀 버튼 예제

스핀 버튼 위젯은 사용자가 위, 아래 방향 화살표로 증가, 감소시킴으로써 정수나 부동 소수점 수를 선택하도록 해준다. 사용자는 키보드를 이용해 값을 입력할 수 있으며, 범위를 벗어날 경우 가장 가까운 허용 값으로 표시될 것이다. 그림 4-6은 정수와 부동 소수점을 표시하는 두 개의 스핀 버튼의 실행 모습을 보여준다.



앞서 언급하였듯 스핀 버튼은 정수 또는 부동 소수점 수를 표시하는 데에 사용할 수 있다. 실제로 숫자는 `gdouble` 값으로서 보관된다. 스핀 버튼은 숫자를 올바른 소수 자리수로 올·내림하는 일을 처리한다. 리스팅 4-6은 정수와 부동 소수점 수의 스핀 버튼을 모두 생성한다.

리스팅 4-6. 정수와 부동 소수점 수 선택 (spinbuttons.c)

```
#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *spin_int, *spin_float, *vbox;
    GtkAdjustment *integer, *float_pt;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Spin Buttons");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 150, 100);

    /* Create two new adjustments. The first spans between 0 and 10,
    starting at 5 and
    * moves in increments of 1. The second spans between 0 and 1,
    starting at 0.5 and
    * moves in increments of 0.1. */
    integer = GTK_ADJUSTMENT (gtk_adjustment_new (5.0, 0.0, 10.0, 1.0,
    2.0, 2.0));
    float_pt = GTK_ADJUSTMENT (gtk_adjustment_new (0.5, 0.0, 1.0, 0.1,
    0.5, 0.5));

    /* Create two new spin buttons. The first will display no decimal
    places and the
    * second will display one decimal place. */
    spin_int = gtk_spin_button_new (integer, 1.0, 0);
    spin_float = gtk_spin_button_new (float_pt, 0.1, 1);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), spin_int);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), spin_float);
}
```

```

gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window);

gtk_main ();
return 0;
}

```

스핀 버튼을 생성하기 전에는 `adjustment`를 생성해야 한다. `NULL adjustment`로 스핀 버튼을 초기화할 수도 있지만 `insensitive`로 설정될 것이다.

자신의 `adjustment`를 초기화한 후에는 `gtk_spin_button_new()`를 이용해 새 스핀 버튼을 생성할 수 있다. 초기화 함수에서 또 다른 두 개의 매개변수는 표시할 소수 자리수와 스핀 버튼의 이동 속도(`climb rate`)를 명시한다. 이동 속도는 화살표 버튼을 누르면 증가하거나 감소되는 값을 의미한다.

```

GtkWidget *gtk_spin_button_new (GtkAdjustment *adjustment,
                                gdouble climb_rate,
                                guint digits);

```

대안적인 방법으로 `gtk_spin_button_new_with_range()`를 이용해 새 스핀 버튼을 생성할 수도 있는데, 이는 개발자가 명시하는 최소, 최대, 단계(`step`) 값을 기반으로 새 `adjustment`를 자동으로 생성할 것이다. 초기 값은 기본적으로 `step_increment`에 10을 곱한 페이지 증가(`page increment`)에 최소 값을 더한 값으로 설정된다. 위젯의 정밀도는 `step_increment`의 값으로 자동 설정된다.

```

GtkWidget* gtk_spin_button_new_with_range (gdouble minimum_value,
                                           gdouble maximum_value,
                                           gdouble step_increment);

```

`gtk_spin_button_set_digits()`를 호출하면 스핀 오프의 새 정밀도를 설정하고, `gtk_spin_button_set_value()`를 이용하면 새 값을 설정할 수 있다. 값이 스핀 버튼의 범위를 벗어나면 자동으로 수정될 것이다.

```

void gtk_spin_button_set_value (GtkSpinButton *spin_button,
                                gdouble value);

```

### 수평 및 수직 스케일

스케일(`scale`)이라 불리는 또 다른 타입의 위젯은 정수나 부동 소수점 수를 선택하는 수평 또는 수직 슬라이더를 제공하도록 해준다. `GtkHScale`은 수평적 스케일 위젯이고, `GtkVScale`은 수직적 스케일 위젯이다. 두 클래스 모두 `GtkScale`에서 파생되어 프로퍼티, 시그널, 함수를 제공한다.

`GtkScale` 위젯의 기능은 `GtkSpinButton`과 크게 다르지 않다. 값은 슬라이더를 이동시켜 선택되기 때문에 사용자가 값을 입력하지 못하도록 제한할 때 종종 사용된다. 그림 4-7은 두 개의 수평적 스케일 위젯의 스크린샷이다.



스케일은 슬라이더를 이용해 숫자를 선택한다는 점을 제외하면 스핀 버튼과 기본적으로 동일한 기능을 제공한다. 위젯 간 유사점을 표시하기 위해 리스팅 4-7은 리스팅 4-6과 동일한 기능을 구현하는데, 두 개의 슬라이더는 사용자가 정수와 부동 소수점 수를 선택하도록 해준다.

리스팅 4-7. 스케일을 이용한 정수 및 부동 소수점 수 선택 (`scales.c`)

```

#include <gtk/gtk.h>

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *scale_int, *scale_float, *vbox;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Scales");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 250, -1);

    /* Create a scale that scrolls integers and one that scrolls
floating point. */
    scale_int = gtk_hscale_new_with_range (0.0, 10.0, 1.0);
    scale_float = gtk_hscale_new_with_range (0.0, 1.0, 0.1);

    /* Set the number of decimal places to display for each widget. */
    gtk_scale_set_digits (GTK_SCALE (scale_int), 0);
    gtk_scale_set_digits (GTK_SCALE (scale_float), 1);

    /* Set the position of the value with respect to the widget. */
    gtk_scale_set_value_pos (GTK_SCALE (scale_int), GTK_POS_RIGHT);
    gtk_scale_set_value_pos (GTK_SCALE (scale_float), GTK_POS_LEFT);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), scale_int);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), scale_float);

    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

```

새로운 스케일 위젯을 생성하는 방법에는 두 가지가 있다. 첫 번째는 `gtk_hscale_new()` 또는 `gtk_vscale_new()`를 이용하는 방식으로, 이는 스케일이 어떻게 작동할 것인지 정의하는 `GtkAdjustment`를 수락한다.

```
GtkWidget *gtk_hscale_new (GtkAdjustment *adjustment);
```

위의 방법이 아니라면, `gtk_hscale_new_with_range()` 또는 `gtk_vscale_new_with_range()`를 이용해 스케일을 생성하는 방법도 있다. 해당 함수는 스케일의 최소값, 최대값, 단계 증가를 수락한다.

```
GtkWidget *gtk_hscale_new_with_range (gdouble minimum,
                                       gdouble maximum,
                                       gdouble step);
```

스케일의 값은 항상 `gdouble`로서 보관되기 때문에 자신이 원하는 기본값이 아닐 경우 `gtk_scale_set_digits()`를 이용해 표시하고자 하는 소수 자리수를 정의해야 할 것이다. 기본 소수 자리수는 단계 증가에 제공되는 소수 자리수를 기반으로 계산된다. 예를 들어, 단계 증가가 0.01일 경우 기본값으로 소수 둘째자리까지 표시될 것이다.

```
void gtk_scale_set_digits (GtkScale *scale,
                          gint digits);
```

사용하는 스케일 위젯의 타입에 따라 값이 표시되는 위치를 변경하고 싶은 경우에는 `gtk_scale_set_value_pos()`를 이용한다. 위치는 `GtkPositionType` 목록에서 정의되는데, `GTK_POST_LEFT`, `GTK_POS_RIGHT`, `GTK_POS_TOP`, `GTK_POS_BOTTOM`이 있다. 사용자에게 값을 모두 숨기려면 `gtk_scale_set_draw_value()`를 이용할 수도 있다.

```
void gtk_scale_set_value_pos (GtkScale *scale,
                              GtkPositionType pos);
```

`GtkScale`은 `GtkRange`라고 불리는 위젯에서 파생된다. 해당 위젯은 추상적 타입으로, `adjustment`를 처리하는 기능을 제공한다. 이 때문에 스케일의 현재 값을 검색하려면 `gtk_range_get_value()`를 이용해야 한다. `GtkRange`는 사용자가 스케일 위치를 변경할 때 방출되는 `value-changed` 시그널과도 제공한다.

### 위젯 스타일

다음 여러 절에서는 위젯 스타일 프로퍼티를 편집할 것이므로 `GtkStyle` 구조와 자원 파일에 대해 학습하기에 알맞은 때가 되었다. 자원 파일은 향후 맞춤설정을 허용하기 위해 런타임 시 자신의 애플리케이션에 로딩 및 적용할 수 있는 스타일 설정의 외부 컬렉션이다.

### GtkStyle 구조

`GtkWidget`은 5개의 `public` 멤버를 가지는데, 이들을 아래 코드 조각에 표시하였다. 스타일 정보, 크기 요청, 크기 할당, 화면에 위젯을 그리는 데에 사용되는 `GdkWindow`, 부모 위젯에 대한 포인터가 해당한다.

```
typedef struct
{
    GtkStyle *style;
    GtkRequisition requisition;
    GtkAllocation allocation;
    GdkWindow *window;
    GtkWidget *parent;
} GtkWidget;
```

`GtkStyle` 구조는 위젯에 관한 드로잉(drawing) 정보를 저장한다. 구조의 내용은 아래와 같다.

```
typedef struct
{
    GdkColor fg[5] /* The foreground color for most widgets. */
    GdkColor bg[5] /* The background color for most widgets. */
    GdkColor light[5] /* Lighter colors used for creating widget
shadows. */
    GdkColor dark[5] /* Darker colors used for creating widget shadows.
*/
    GdkColor mid[5] /* The color midway between light and dark. */
    GdkColor text[5] /* The text color for most text widgets. */
    GdkColor base[5] /* The background color used for text-editing
```

```

widgets. */
    GdkColor text_aa[5]; /* Used for anti-aliased text colors. */
    GdkColor black, white; /* Colors that represent "Black" and
"White". */

    PangoFontDescription *font_desc; /* The default text font. */
    gint xthickness, ythickness; /* Thickness of lines. */
    GdkPixmap *bg_pixmap[5]; /* Background image to use for a widget.
*/

    /* Graphics contexts that hold drawing properties for each color
and state. */
    GdkGC *fg_gc [5], *bg_gc [5], *light_gc[5], *dark_gc[5],
*mid_gc[5], *text_gc[5],
    *base_gc[5], *text_aa_gc[5];
    GdkGC *black_gc, *white_gc;
} GtkWidget;
    
```

GtkStyle 구조에는 많은 객체들이 있다. 각 객체는 사용자의 스타일에 따라 설정되는 기본값을 갖게 될 것이므로, 이 값을 오버라이드하는 것이 항상 바람직한 것은 아니다. 하지만 필요하다면 위젯의 GtkWidget의 편집을 통해 위젯의 표시 방법을 가장 간단하게 변경할 수 있다.

스타일 프로퍼티 중 다수는 파일 요소로 구성된 배열임을 눈치챌 것이다. 이는 각 요소들이 아래에 소개된 다섯 가지의 위젯 상태 중 하나에 대해 서로 다른 값을 가질 수 있기 때문이다.

- GTK\_STATE\_NORMAL: 정상 작동 중인 위젯.
- GTK\_STATE\_ACTIVE: 토글을 누름해제할 때와 같은 활성(active) 위젯.
- GTK\_STATE\_PRELIGHT: 마우스 포인터가 위젯 위에 있을 때 위젯으로, 버튼 클릭에 반응할 것이다.
- GTK\_STATE\_SELECTED: 위젯 또는 그 텍스트를 선택할 때 위젯.
- GTK\_STATE\_INSENSITIVE: 위젯이 비활성화되고 사용자에게 반응하지 않을 것이다.

### 자원 파일

GTK+는 애플리케이션이 자원 파일(RC 파일)이라 불리는 사용자 정의 스타일을 사용할 수 있는 방도를 제공한다. RC 파일은 사용자가 위젯 타입이나 개별적 위젯에 대한 스타일을 정의하도록 해주어, 사용자의 개인 설정에 맞도록 변경할 수 있다. 이는 주로 다른 애플리케이션과 함께 사용자의 홈 디렉터리로 보관되므로 사용자는 설정을 변경할 권한을 갖는다.

자원 파일을 로딩하기 위해서는 애플리케이션을 로딩 시 gtk\_rc\_parse()를 호출해야 한다. 그러면 적절한 모든 위젯에 자동으로 스타일을 적용할 것이다.

```
void gtk_rc_parse (const gchar *filename);
```

또 RC 파일로부터 직접 위젯을 참조하고 싶다면 gtk\_widget\_set\_name()을 이용해 위젯에 대한 유일한 이름을 설정할 필요가 있다. 해당 이름은 위젯의 스타일과 그 자식들의 스타일을 설정하기 위해 RC 파일에서 사용될 것이다.

리스팅 4-8에서는 간단한 RC 파일의 예제를 제시하였다. 이 예제에서는 각각이 수많은 프로퍼티를 포함하는 위젯 스타일이 다수 생성된다.

리스팅 4-8. 위젯 스타일 정의하기 (.gtkrc)

```
style "widgets"
{
```



```

xthickness = 2
ythickness = 2

fg[ACTIVE] = "#FFFFFF"
fg[SELECTED] = "#003366"
fg[NORMAL] = "#CCCCCC"
fg[PRELIGHT] = "#FFFFFF"
fg[INSENSITIVE] = "#999999"

bg[ACTIVE] = "#003366"
bg[SELECTED] = "#FFFFFF"
bg[NORMAL] = "#666666"
bg[PRELIGHT] = "#003366"
bg[INSENSITIVE] = "#666666"
}

style "labels" = "widgets" {
    font_name = "Sans Bold 14"
}

style "buttons" = "widgets" {
    GtkButton::inner-border = { 10, 10, 10, 10 }
}

style "checks" = "buttons" {
    GtkCheckButton::indicator-size = 25
}

class "GtkWindow" style "widgets"
class "GtkLabel" style "labels"
class "GtkCheckButton" style "checks"
class "Gtk*Button" style "buttons"
    
```

그림 4-8은 리스팅 4-8에 표시된 RC 파일을 활용하는 애플리케이션을 보여준다. 색상과 글꼴은 앞 장에 실린 예제들과 다르다.



RC 파일의 모든 위젯에 이용 가능한 표준 스타일을 살펴보려면 부록 C를 참고해야 한다. 이번 절에서는 자신만의 애플리케이션에서 그러한 스타일을 적용하는 방법을 가르칠 것이다.

스타일은 앞의 예제에 표시된 class 지시어를 이용해 위젯 타입에 의해 적용될 수 있다. 이번 예제에서는 button 스타일이 모든 Gtk\*Button\* 위젯으로 적용되는데, 별표는 와일드카드(wildcard)로 사용된다. 이는 일치

하는 클래스명을 가진 애플리케이션 내의 모든 위젯에 적용된다.

```
class "Gtk*Button" style "buttons"
```

위젯 스타일을 적용하기 위한 두 번째 방법은 widget 지시어를 이용해 계층구조 패턴을 기반으로 한다. 이 예제는 GtkButton 타입으로 된 widgetname의 모든 간접 및 직접 자식들에게 stylename 스타일을 적용한다.

```
widget "widgetname.*.GtkButton" style "stylename"
```

0개 또는 그 이상의 문자에 일치하는 별표 와일드카드 외에 물음표 와일드카드를 이용하면 어떤 문자든 하나 또는 이상으로 매칭할 수 있다. 또 위젯 계층구조는 마침표(period)를 이용해 표시되는데, 마침표의 우측에 있는 위젯은 좌측에 위치한 위젯의 자식이다.

widget 지시어에서 발생하는 문제는, 위젯에 대한 이름이 명시되는 경우 클래스명 대신 그 이름이 사용되어야만 한다는 데에 있다. 위젯 클래스만 사용하길 원한다면 widget\_class 지시어를 사용할 수 있다. 이는 모든 위젯명을 무시하고, 명시된 패턴을 따르는 모든 위젯에 스타일을 적용하도록 해준다.

```
widget_class "GtkWindow.*.GtkLabel" style "stylename"
```

기본 스타일 지시어 외에 RC 파일에서 지원되는 최상위 수준 지시어를 아래 목록에서 표시한다.

- include: 다른 자원 파일을 포함한다. 절대 파일명과 상대 파일명 중 하나를 명시할 수 있다.
- module\_path: 콜론으로 구분된 경로 리스트로, RC 파일이 참조하는 테마 엔진을 검색할 것이다.
- \*pixmap\_path: 콜론으로 구분된 경로 리스트로, RC 파일이 참조하는 테마 엔진을 검색할 것이다.

애플리케이션에서 RC 파일을 이용할 계획이라면 사용자에게 예제 파일을 제공해야 함을 명심한다. pound (#) 부호를 이용해 RC 파일에 주석을 추가하여 사용자가 내용을 편집하도록 도와줄 수 있다.

이번 절은 RC 파일의 매우 기초적인 내용만 소개하였다. 더 많은 정보는 부록 C를 참조해야 한다. GTK+와 함께 테마와 RC 파일을 학습하는 데 도움이 되는 자원은 <http://art.gnome.org> 찾을 수 있다.

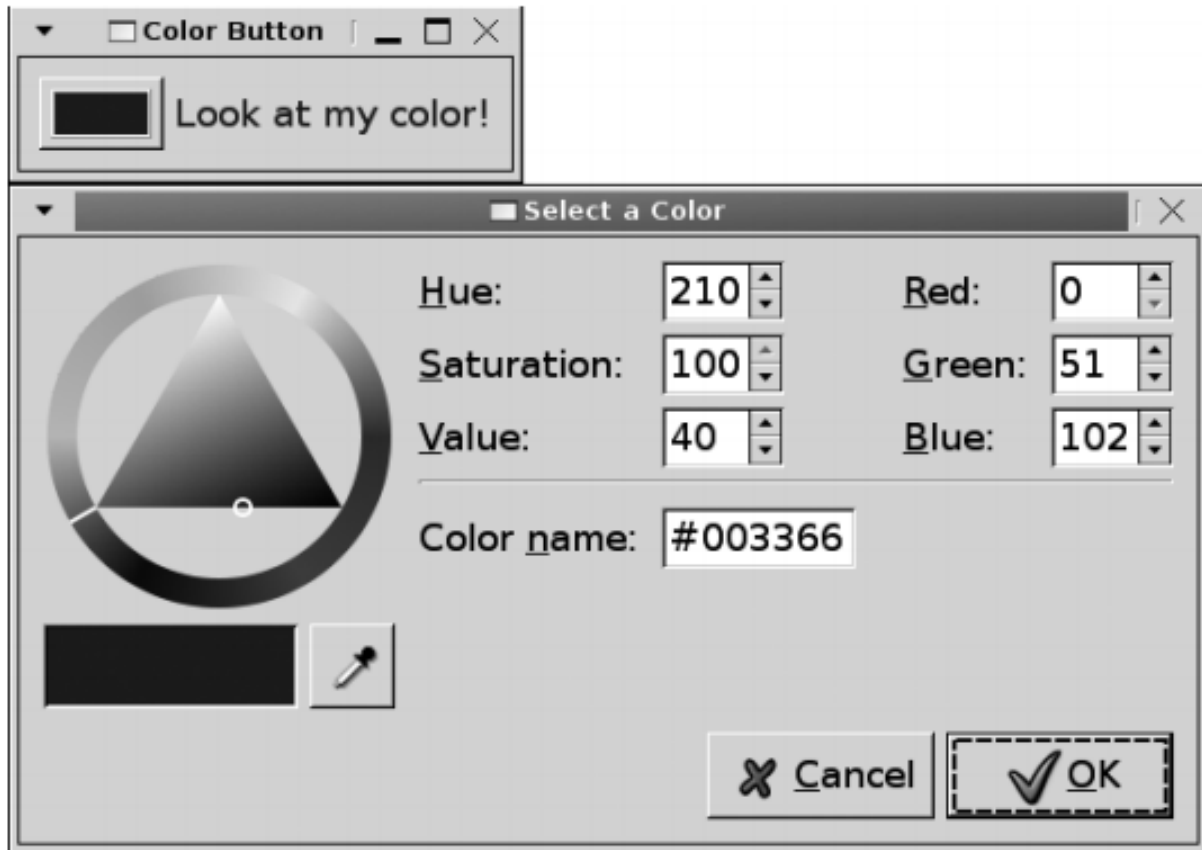
### 추가 버튼

GtkButton 위젯은 자신만의 커스텀 버튼을 생성하도록 해주지만 GTK+는 개발자가 마음대로 이용할 수 있도록 3개의 추가 버튼 위젯, 즉 색상 선택 버튼, 파일 선택자 버튼, 글꼴 선택 버튼을 제공한다.

아래의 각 절은 세 가지 위젯을 다루면서 GdkColor 구조, 파일 필터, Pango 글꼴처럼 중요한 개념들도 함께 다룰 것이다. 이러한 개념들은 이후 다른 여러 장에서 사용될 것이기 때문에 지금 알아두면 유용할 것이다.

### 색상 버튼

GtkColorButton 위젯은 사용자들이 특정 색상을 선택할 수 있도록 개발자에게 간단한 방도를 제공한다. 이러한 색상은 6자리의 16진 값이나 RGB 값으로 명시할 수 있다. 색상 자체는 버튼의 자식 위젯으로 설정된 직사각형 블록에 선택된 색상을 표시한다. 이 예는 그림 4-9에서 확인할 수 있다.



**GtkColorButton 예제**

색상 버튼을 클릭하면 사용자가 색상 휠에서 선택내용을 확인하거나 색상값을 입력하도록 해주는 대화상자가 열린다. 색상 휠이 제공되면 사용자는 색상의 수치값을 꼭 알 필요가 없다. 리스팅 4-9는 애플리케이션에서 GtkColorButton 위젯을 사용하는 방법을 보여준다.

리스팅 4-9. 색상 버튼과 GdkColors (colorbuttons.c)

```
#include <gtk/gtk.h>

static void color_changed (GtkColorButton*, GtkWidget*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *button, *label, *hbox;
    GdkColor color;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Color Button");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Set the initial color as #003366 and set the dialog title. */
    gdk_color_parse ("#003366", &color);
    button = gtk_color_button_new_with_color (&color);
```

```

    gtk_color_button_set_title (GTK_COLOR_BUTTON (button), "Select a
Color");

    label = gtk_label_new ("Look at my color!");
    gtk_widget_modify_fg (label, GTK_STATE_NORMAL, &color);

    g_signal_connect (G_OBJECT (button), "color_set",
        G_CALLBACK (color_changed),
        (gpointer) label);

    hbox = gtk_hbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (hbox), button);
    gtk_box_pack_start_defaults (GTK_BOX (hbox), label);

    gtk_container_add (GTK_CONTAINER (window), hbox);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* Retrieve the selected color and set it as the GtkLabel's foreground
color. */

static void
color_changed (GtkColorButton *button,
               GtkWidget *label)
{
    GdkColor color;
    gtk_color_button_get_color (button, &color);
    gtk_widget_modify_fg (label, GTK_STATE_NORMAL, &color);
}

```

대부분의 경우 초기 색상 값으로 된 `GtkColorButton`을 생성하길 원할 것인데, 이는 `gtk_color_button_new_with_color()`에 `GdkColor` 객체를 명시함으로써 이루어진다. 아무 것도 제공되지 않을 경우 알파 옵션이 비활성화된 불투명한 검정색이 기본색이 된다.

색상을 `GdkColor`에 보관하기

`GdkColor`는 아래 코드 조각에 표시된 바와 같이 색상에 대한 적색, 녹색, 청색을 보관하는 구조다. `pixel` 객체는 색상이 색상 맵에 할당될 때 그 색인을 자동으로 보관하므로, 개발자는 보통 이 값을 수정할 필요가 없다.

```

struct GdkColor
{
    guint32 pixel;
    guint16 red;
    guint16 green;
    guint16 blue;
};

```

새로운 GdkColor 객체를 생성한 후 색상에 대한 적색, 녹색, 청색 값을 이미 알고 있다면 다음 방법을 이용해 명시할 수 있다. 적색, 녹색, 청색은 0-65,535 범위의 부호가 없는 정수 값으로서 보관되는데, 65,535는 전체 색상 강도를 나타낸다. 가령, 아래의 색상은 흰색을 나타낸다.

```
color.red = 65535;
color.green = 65535;
color.blue = 65535;
```

대부분은 색상에 대해 6자리로 된 16진 값에 익숙할 것인데, 가령 #FFFFFF는 흰색을 나타낸다. 따라서 GDK는 16진 색상을 올바른 RGB 값으로 파싱하는 gdk\_color\_parse()를 제공한다. 해당 함수는 리스팅 4-9에서 사용되었다.

```
gboolean gdk_color_parse (const gchar *color_string,
                          GdkColor *color);
```

### 색상 버튼 사용하기

초기 값을 설정한 후에는 gtk\_color\_button\_set\_title()을 이용해 색상 선택 대화상자에 주어질 제목을 선택할 수 있다. 기본적으로 제목은 "Pick a Color"로 주어지는데, 제목이 괜찮다면 굳이 이 값을 변경할 필요는 없다.

```
void gtk_color_button_set_title (GtkColorButton *button,
                                const gchar *title);
```

다음 장에서 더 다루게 될 색상 선택 대화상자는 사용자가 버튼을 클릭하면 표시된다. 이는 선택된 색상을 사용자가 변경할 수 있도록 해준다. 색상 선택 대화상자는 그림 4-9에서 살펴볼 수 있다.

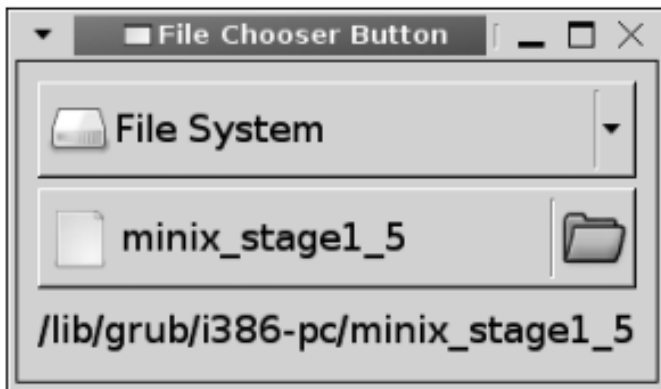
색상 값이 변경되면 위젯에 대해 color-set 시그널이 방출된다. 리스팅 4-5에서는 시그널이 잡히고, gtk\_widget\_modify\_fg()를 이용해 GtkLabel의 전경색이 다음과 같이 변경된다.

```
gtk_color_button_get_color (button, &color);
gtk_widget_modify_fg (label, GTK_STATE_NORMAL, &color);
```

리스팅 4-9에서 전경색은 일반 위젯 상태로 설정되었으며, 모든 라벨이 선택가능하지 않은 이상 대체적으로 그 상태일 것이다. gtk\_widget\_modify\_fg()에서 이용할 수 있는 GtkStateType 목록에 대한 옵션으로는 다섯 가지가 있는데, 이는 "위젯 스타일" 절에서 제시하였다. NULL 색상을 전달하면 위젯의 전경색을 기본값으로 리셋할 수 있다.

### 파일 선택자 버튼

GtkFileChooserButton 위젯은 개발자가 사용자에게 파일이나 폴더를 선택할 것을 요청하는 방법을 제공한다. 해당 위젯은 GtkFileChooser 인터페이스의 기능을 구현하는데, 바로 GTK+가 제공하는 파일 선택 프레임워크다. 그림 4-10는 폴더를 선택하도록 설정된 파일 선택자와 파일을 선택하도록 설정된 버튼을 보여준다.



사용자가 `GtkFileChooserButton`을 클릭하면 `GtkFileChooserDialog`의 인스턴스가 열리고, 개발자가 생성한 버튼의 타입에 따라 사용자가 하나의 파일이나 폴더를 브라우징 및 선택하도록 해준다.

■ **Note** 제 5장을 학습하기 전에는 `GtkFileChooserDialog` 위젯을 어떻게 사용하는지 학습하지 않을 것이지만 현재 시점에서 이를 이용해 직접 인터페이스와 접속할 필요는 없는데, `GtkFileChooserButton`이 대화상자와의 모든 상호작용을 처리할 것이기 때문이다.

### `GtkFileChooserButton` 예제

현재 선택된 파일, 현재 폴더, 파일 선택창 제목과 같은 기본 설정을 변경할 수도 있다. 리스팅 4-10은 파일 선택자 버튼의 두 가지 타입을 모두 어떻게 사용하는지를 보여준다.

리스팅 4-10. 파일 선택자 버튼 이용하기 (`filechooserbuttons.c`)

```
#include <gtk/gtk.h>

static void folder_changed (GtkFileChooser*, GtkFileChooser*);
static void file_changed (GtkFileChooser*, GtkLabel*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *chooser1, *chooser2, label, *vbox;
    GtkFileFilter *filter;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "File Chooser Button");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    label = gtk_label_new ("");

    /* Create two buttons, one to select a folder and one to select a
file. */
    chooser1 = gtk_file_chooser_button_new ("Chooser a Folder",
                                           GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER);
    chooser2 = gtk_file_chooser_button_new ("Chooser a Folder",
                                           GTK_FILE_CHOOSER_ACTION_OPEN);

    /* Monitor when the selected folder or file are changed. */
    g_signal_connect (G_OBJECT (chooser1), "selection_changed",
                     G_CALLBACK (folder_changed),
                     (gpointer) chooser2);
    g_signal_connect (G_OBJECT (chooser2), "selection_changed",
                     G_CALLBACK (file_changed),
                     (gpointer)
label);

    /* Set both file chooser buttons to the location of the user's home
```

```

directory. */
    gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER (chooser1),
        g_get_home_dir());
    gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER (chooser2),
        g_get_home_dir());

    /* Provide a filter to show all files and one to show only 3 types
of images. */
    filter1 = gtk_file_filter_new ();
    filter2 = gtk_file_filter_new ();
    gtk_file_filter_set_name (filter1, "Image Files");
    gtk_file_filter_set_name (filter2, "All Files");
    gtk_file_filter_add_pattern (filter1, "*.png");
    gtk_file_filter_add_pattern (filter1, "*.jpg");
    gtk_file_filter_add_pattern (filter1, "*.gif");
    gtk_file_filter_add_pattern (filter2, "");

    /* Add both the filters to the file chooser button that selects
files. */
    gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (chooser2), filter1);
    gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (chooser2), filter2);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), chooser1);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), chooser2);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), label);

    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* When a folder is selected, use that as the new location of the other
chooser. */
static void
folder_changed (GtkFileChooser *chooser1,
                GtkFileChooser *chooser2)
{
    gchar *folder = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER
(chooser1));
    gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER (chooser2),
folder);
}

/* When a file is selected, display the full path in the GtkLabel

```

```
widget. */
static void
file_changed (GtkFileChooser *chooser2,
             GtkWidget *label)
{
    gchar *file = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER
(chooser2));
    gtk_label_set_text (label, file);
}
```

파일 선택자 버튼 위젯은 `gtk_file_chooser_button_new()`를 이용해 생성된다. 해당 위젯은 두 가지 목적에 부응하는데, 바로 단일 파일을 선택하거나 단일 폴더를 선택하는 것이다. 생성 가능한 파일 선택자의 유형에는 네 가지가 있지만(나머지 두 가지는 제 5장에서 다루겠다) 파일 선택자 버튼은

`GTK_FILE_CHOOSER_ACTION_OPEN`과 `GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER`만 지원한다.

- `GTK_FILE_CHOOSER_ACTION_OPEN`: 사용자는 시스템에 이미 존재하는 단일 파일을 선택할 수 있을 것이다. 당신은 이러한 액션 타입으로 필터를 제공하여 특정 파일 패턴만 사용자에게 표시되도록 할 수 있다.
- `GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER`: 사용자는 시스템에 이미 존재하는 단일 폴더를 선택할 수 있을 것이다.

`gtk_file_chooser_button_new()` 내의 또 다른 매개변수는 사용자가 버튼을 클릭할 때 표시되는 파일 선택자 대화상자의 제목을 설정하도록 해준다. 기본 제목은 "Select A File"이며

`GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER`를 이용하면 제목이 리셋된다.

### GtkFileChooser

`GtkFileChooserButton` 위젯은 `GtkFileChooser` 인터페이스가 제공하는 기능의 구현이다. 즉, 버튼이 `GtkFileChooser`로부터 파생되지는 않았지만 `GTK_FILE_CHOOSER()`로 형변환할 경우 파일 선택자로서 취급할 수 있다는 의미다. 리스팅 4-10에 열거된 함수들 중 소수만이 `GtkFileChooser`가 제공한 함수들을 활용한다는 사실을 인지할 것이다.

리스팅 4-10에서 `gtk_file_chooser_set_current_folder()`는 각 파일 선택자 버튼의 현재 폴더를 사용자의 홈 디렉터리로 설정하는 데에 사용되었다. 이러한 폴더의 내용은 사용자가 처음으로 파일 선택자 버튼을 클릭할 때 표시되겠지만, 다른 방법을 통해 그 내용이 변경된 경우라면 제외된다. 폴더가 성공적으로 변경되면 해당 함수는 `TRUE`를 리턴할 것이다.

```
gboolean gtk_file_chooser_set_current_folder (GtkFileChooser *chooser,
                                             const gchar *filename);
```

`g_get_home_dir()` 함수는 `GLib`가 제공하는 유틸리티 함수로서, 현재 사용자의 홈 디렉터리를 리턴한다. `GLib` 내의 대부분 기능과 마찬가지로 해당 함수는 크로스 플랫폼적이다.

이는 파일 선택자 인터페이스의 유용한 특징을 상기시키는데, UNIX에서 실행되든 Windows 머신에서 실행되든 많은 유형의 파일 구조를 확인하는 데에 사용 가능하다는 점이다. 자신의 애플리케이션을 여러 운영체제용으로 컴파일하고자 할 경우 특히 유용하다.

파일 선택자 버튼은 한 번에 하나의 파일 선택만 허용하므로 파일 선택자 버튼의 타입에 따라 현재 선택된 파일이나 폴더를 검색하기 위해서는 `gtk_file_chooser_get_filename()`을 이용할 수 있다. 선택된 파일이 없다면 함수는 `NULL`을 리턴할 것이다. 리턴된 문자열에 대한 작업이 완료되면 `g_free()`를 이용해 해제시켜야 한다.

```
gchar* gtk_file_chooser_get_filename (GtkFileChooser *chooser);
```

`GtkFileChooser` 인터페이스에 대해 이 정도의 정보라면 지금으로선 파일 선택자 버튼을 구현하기에 충분하다. 다음 장에서 `GtkFileChooserDialog` 위젯을 학습하면서 `GtkFileChooser`를 좀 더 깊이 다룰 것이다.



파일 필터

GtkFileFilter 객체는 파일 선택자에 표시되는 파일을 제한하도록 해준다. 가령, 리스팅 4-10에서 Image Files 필터가 선택되면 사용자는 PNG, JPG, GIF 파일만 보고 선택할 수 있다.

파일 필터는 gtk\_file\_filter\_new()를 이용해 생성된다. 따라서 필터 타입에 표시되는 이름을 설정하려면 gtk\_file\_filter\_set\_name()을 이용할 필요가 있겠다. 하나 이상의 필터를 제공할 경우 이러한 이름은 사용자가 필터 간 전환하는 데에 도움이 된다.

```
GtkFileFilter* gtk_file_filter_new ();
void gtk_file_filter_set_name (GtkFileFilter *filter,
    const gchar *name);
```

마지막으로 필터를 완료하려면 표시해야 할 파일 타입을 추가해야 한다. 일반적으로는 아래의 코드 조각에 표시된 바와 같이 gtk\_file\_filter\_add\_pattern()를 이용하여 이루어진다. 해당 함수는 표시되어야 할 파일명에 대한 포맷을 명시하도록 해준다. 보통은 표시해야 할 파일 확장자를 식별하면 된다. 필터링 함수의 타입에는 와일드카드로 별표 문자를 이용할 수 있다.

```
void gtk_file_filter_add_pattern (GtkFileFilter *filter,
    const gchar *pattern);
```

**Tip** 리스팅 4-10에서와 같이 당신은 All Files 필터를 제공하여 디렉터리 내 모든 파일을 표시하길 원할지도 모른다. 이를 위해서는 하나의 패턴만 와일드카드 문자로 설정된 필터를 생성해야 한다. 해당 필터를 제공하지 않으면 사용자는 다른 필터가 제공한 패턴에 매치하지 않는 파일은 절대로 볼 수 없을 것이다.

gtk\_file\_filter\_add\_mime\_type()를 이용해 Multipurpose Internet Mail Extensions(MIME) 타입을 명시함으로써 필터 패턴을 명시하는 수도 있다. 가령, image/\* 는 이미지 MIME 타입에 해당하는 파일을 모두 표시할 것이다. 해당 함수에서 발생할 수 있는 문제는, MIME 타입에 익숙할 필요가 있다는 것이다. 하지만 MIME 타입을 이용 시 장점은 필터에 대한 모든 파일 확장자를 명시할 필요가 없다는 것이다. 이는 구체적인 MIME 범주에 모든 파일을 일반화하도록 허용한다.

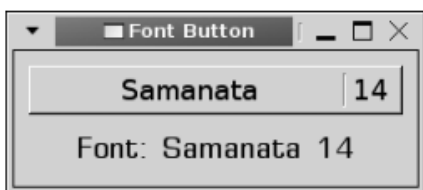
```
void gtk_file_filter_add_mime_type (GtkFileFilter *filter,
    const char *mime_type);
```

필터를 생성하고 나면 파일 선택자로 추가되어야 하는데, 이 때는 gtk\_file\_chooser\_add\_filter()를 이용하면 된다. 필터를 제공하고 나면 기본적으로 처음으로 명시된 필터들이 파일 선택자 내에서 사용될 것이다. 다수의 필터를 명시할 경우 사용자는 타입들 간 전환할 수 있을 것이다.

```
void gtk_file_chooser_add_filter (GtkFileChooser *chooser,
    GtkFileFilter *filter);
```

글꼴 버튼

GtkFontButton은 사용자가 사용자의 시스템에 현재 상주하는 글꼴에 상응하는 글꼴 매개변수를 선택하도록 해주는 또 다른 유형의 특수화 버튼이다. 글꼴 옵션은 사용자가 버튼을 클릭하면 표시되는 글꼴 선택 대화상자에서 선택된다. 이러한 옵션으로는 글꼴명, 스타일 옵션, 글꼴 크기가 포함된다. GtkFontButton 위젯 예제는 그림 4-11에 표시하였다.



글꼴 버튼 위젯은 `gtk_font_button_new_with_font()`를 이용해 초기화되는데 이는 초기 글꼴을 명시하도록 해준다. 글꼴은 Family Style Size로 된 문자열로서 제공된다. 각 매개변수는 선택적이며, `GtkFontButton`에 대한 기본 글꼴은 Sans 12로서, 스타일 매개변수를 전혀 제공하지 않는다.

"패밀리(Family)"란 "Sans", "Serif", "Arial"과 같은 공식 글꼴명을 나타낸다. 스타일 옵션은 글꼴별로 다양하지만 보통 "이탤릭체(Italic)", "볼드체(Bold)", "볼드 이탤릭체(Bold Italic)"를 포함한다. Regular로 글꼴 스타일을 정하면 어떤 글꼴도 명시되지 않을 것이다. 크기는 표시되어야 할 텍스트의 포인트 크기(point size)로, "12" 또는 "12.5"를 예로 들 수 있겠다.

### GtkFontButton 예제

리스팅 4-11은 "Sans Bold 12"의 글꼴로 초기화되는 `GtkFontButton` 위젯을 생성한다. 버튼에 선택된 글꼴이 변경되면 글꼴 버튼 아래에서 패킹된 `GtkLabel` 글꼴에 새 글꼴이 적용된다.

리스팅 4-11. 글꼴 선택 버튼 이용하기 (fontbuttons.c)

```
#include <gtk/gtk.h>

static void font_changed (GtkFontButton*, GtkWidget*);

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *vbox, *button, *label;
    PangoFontDescription *initial_font;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Font Button");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    label = gtk_label_new ("Look at the font!");
    initial_font = pango_font_description_from_string ("Sans Bold 12");
    gtk_widget_modify_font (label, initial_font);

    /* Create a new font selection button with the given default font.
    */
    button = gtk_font_button_new_with_font ("Sans Bold 12");
    gtk_font_button_set_title (GTK_FONT_BUTTON (button), "Choose a
Font");

    /* Monitor for changes to the font chosen in the font button. */
    g_signal_connect (G_OBJECT (button), "font_set",
                     G_CALLBACK (font_changed),
                     (gpointer) label);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), button);
    gtk_box_pack_start_defaults (GTK_BOX (vbox), label);
}
```



```
void gtk_font_button_set_use_size (GtkFontButton *button,
    gboolean use_size);
```

### 자신의 이해도 시험하기

이번 장을 통해 GtkEntry, GtkSpinButton, 다양한 토글 및 버튼 타입을 포함해 많은 기본 위젯을 학습하였다. 이러한 위젯들의 사용법을 연습하기 위해 아래 소개된 두 개의 연습문제를 통해 두 가지 애플리케이션을 생성할 것이다.

#### 연습문제 4-1. 파일 재명명하기

이번 연습문제에서는 GtkFileChooserButton 위젯을 이용하여 사용자가 시스템에서 파일명을 선택하도록 한다. 다음으로, 사용자가 파일에 대해 새로운 이름을 명시하도록 GtkEntry 위젯을 사용하라. (연습문제에서 필요로 하는 파일 유틸리티에 관한 함수는 GLib API 문서에서 찾을 수 있음을 명심한다.)

파일이 성공적으로 재명명되면 사용자가 새 파일을 선택할 때까지 GtkEntry 위젯과 버튼을 비활성화시켜야 한다. 사용자가 선택된 파일을 재명명할 권한을 갖지 않은 경우 GtkEntry 위젯과 버튼은 insensitive로 설정되어야 한다. 해당 연습문제를 완료한 후 부록 F에서 해답을 찾을 수 있다.

이 연습문제는 본 장에서 다룬 두 개의 위젯, GtkEntry와 GtkFileChooserButton을 이용한다. 또 파일의 재명명에 필요한 함수와 기존 파일의 권한에 관한 정보 검색에 필요한 함수를 포함해 GLib이 제공하는 많은 유틸리티 함수들을 사용하도록 요한다.

제 6장에 가셔야 GLib를 학습할 것이지만 디렉터리 생성, 파일 권한 변경, 디렉터리 구조 이동하기 등 파일에 관련된 다른 유틸리티 함수들도 경험해보고 싶을 것이다. GLib는 많은 기능을 제공하므로 시간이 충분하다면 API 문서를 살펴보는 것이 유용하겠다.

#### 연습문제 4-2. 스피너 버튼과 스케일

해당 연습문제에서는 3개의 위젯, 즉 스피너 버튼, 수평 스케일, 체크 버튼을 생성한다. 스피너 버튼과 수평 스케일은 동일한 초기값과 범위(bounds)로 설정되어야 한다. 체크 버튼을 선택하면 두 개의 adjustment 위젯이 동일한 값으로 동기화되어야 한다. 즉, 사용자가 하나의 위젯 값을 변경하면 나머지도 동일한 값으로 변경될 것이라는 의미다.

두 위젯 모두 정수와 부동 소수점 수를 지원하므로, 이 연습문제는 다양한 소수 자리수로 구현되어야 한다. adjustment와 편의 initializer를 모두 이용해 스피너 버튼과 스케일을 생성하는 연습도 해야 할 것이다.

본 장에서 소개한 위젯의 수가 많기 때문에 연습문제를 통해 모두 사용법을 익힐 수는 없다. 하지만 두 연습문제를 완료했다면 이번 장에서 다루었던 다른 위젯들도 모두 이해했는지 개별적으로 확인해야 한다.

이러한 위젯들 중 다수는 본 저서에 걸쳐, 그리고 향후 자신의 애플리케이션에서도 사용하게 될 것이므로 기본 위젯들을 이용해 이것저것 실험해 볼 것을 권한다. 위젯이 제공하는 기능들 중 본 장에서 다루지 않은 내용도 API 문서를 통해 학습해야 한다.

### 요약

본 장에서는 사용자와의 주요 상호작용 방법을 제공하는 9개의 새로운 위젯들을 학습하였다.

- GtkToggleButton: 클릭한 후 활성화 또는 비활성 상태를 보유하는 GtkButton 위젯의 타입. 활성화 시 누름 상태로 표시된다.
- GtkCheckButton: GtkToggleButton에서 파생되고, 표시된 텍스트 옆에 별개의 토글로 그려진다. GtkButton과 구별되도록 해준다.
- GtkRadioButton: 다수의 라디오 버튼 위젯을 그룹화하여 한 번에 하나의 토글만 활성화할 수 있도록 해준다.
- GtkEntry: 해당 위젯은 사용자가 하나의 행에 자유 형식의 텍스트를 입력하도록 해준다. 비밀번호 엔트리에도 사용 가능하다.



```

("_Click Me"); g_signal_connect (G_OBJECT (button), "clicked", G_CALLBACK (button_clicked), (gpointer)
window); gtk_container_add (GTK_CONTAINER (window), button); gtk_widget_show_all (window); gtk_main
(); return 0; } /* Create a new GtkDialog that will tell the user that the button was clicked. */ static void
button_clicked (GtkButton *button, GtkWidget *parent) { GtkWidget *dialog, *label, *image, *hbox; /* Create a
new dialog with one OK button. */ dialog = gtk_dialog_new_with_buttons ("Information", parent,
GTK_DIALOG_MODAL, GTK_STOCK_OK, GTK_RESPONSE_OK, NULL); gtk_dialog_set_has_separator
(GTK_DIALOG (dialog), FALSE); label = gtk_label_new ("The button was clicked!"); image =
gtk_image_new_from_stock (GTK_STOCK_DIALOG_INFO, GTK_ICON_SIZE_DIALOG); hbox =
gtk_hbox_new (FALSE, 5); gtk_container_set_border_width (GTK_CONTAINER (hbox), 10);
gtk_box_pack_start_defaults (GTK_BOX (hbox), image); gtk_box_pack_start_defaults (GTK_BOX (hbox), label);
/* Pack the dialog content into the dialog's GtkVBox. */ gtk_box_pack_start_defaults (GTK_BOX (GTK_DIALOG
(dialog)->vbox), hbox); gtk_widget_show_all (dialog); /* Create the dialog as modal and destroy it when a button is
clicked. */ gtk_dialog_run (GTK_DIALOG (dialog)); gtk_widget_destroy (dialog); }

```

**gtk\_dialog\_new\_with\_buttons()** returns **GtkDialog** object. It returns 0 if the dialog is not modal and nonmodal.
**GtkWidget\* gtk\_dialog\_new\_with\_buttons (const gchar \*title, GtkWidget \*parent, GtkDialogFlags flags, const
gchar \*first\_button\_text, ...);**

**gtk\_dialog\_new\_with\_buttons()** returns **GtkDialog** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_set\_has\_separator()** returns **void**.
**void gtk\_dialog\_set\_has\_separator (GtkDialog \*dialog, gboolean has\_separator);**

**gtk\_box\_pack\_start\_defaults()** returns **void**.
**void gtk\_box\_pack\_start\_defaults (GtkWidget \*widget, GtkWidget \*child, gboolean expand, gboolean fill,
gboolean align, gboolean baseline, gboolean spacing, gboolean spacing\_flags, gboolean spacing\_reserve,
gboolean spacing\_reserve\_flags, gboolean spacing\_reserve\_min, gboolean spacing\_reserve\_max);**

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

**gtk\_dialog\_run()** returns **GtkResponseType** object. It returns 0 if the dialog is not modal and nonmodal.
**GTK\_DIALOG\_MODAL:** The dialog is modal.
**GTK\_DIALOG\_DESTROY\_WITH\_PARENT:** The dialog is destroyed when the parent is destroyed.
**GTK\_DIALOG\_NO\_SEPARATOR:** The dialog has no separator.

`gtk_widget_destroy()` destroys the widget, and the widget is destroyed. GTK+ destroys the widget and the widget is destroyed. `gtk_widget_destroy()` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed.

`gtk_image_new_from_stock()` creates a `GtkImage` widget from a stock icon. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed.

`GTK_ICON_SIZE_MENU: 16 x 16`, `GTK_ICON_SIZE_SMALL_TOOLBAR: 18 x 18`, `GTK_ICON_SIZE_LARGE_TOOLBAR: 24 x 24`, `GTK_ICON_SIZE_BUTTON: 24 x 24`

`GTK_ICON_SIZE_DND: 32 x 32`, `GTK_ICON_SIZE_DIALOG: 48 x 48` The `GtkImage` widget is destroyed, and the widget is destroyed.

`gtk_image_new_from_file()` creates a `GtkImage` widget from a file. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed.

`gtk_image_new_from_pixbuf()` creates a `GtkImage` widget from a `GdkPixbuf`. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed.

`gtk_image_new_from_pixbuf()` creates a `GtkImage` widget from a `GdkPixbuf`. `GtkImage` destroys the widget, and the widget is destroyed. `GtkImage` destroys the widget, and the widget is destroyed.

`gtk_dialog_run()` runs the dialog. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`button_clicked()` creates a nonmodal dialog with one OK button. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`gtk_dialog_set_has_separator()` sets the separator. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`gtk_box_pack_start_defaults()` packs the widget. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`gtk_widget_show_all()` shows the widget. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`g_signal_connect()` connects the signal. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`gtk_dialog_run()` runs the dialog. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed. `GtkDialog` destroys the widget, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

`GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed. `GTK_DIALOG_MODAL` The `GtkDialog` widget is destroyed, and the widget is destroyed.

```

default values. */ gtk_entry_set_text (GTK_ENTRY (user), g_get_user_name()); gtk_entry_set_text (GTK_ENTRY
(real), g_get_real_name()); gtk_entry_set_text (GTK_ENTRY (home), g_get_home_dir()); gtk_entry_set_text
(GTK_ENTRY (host), g_get_host_name()); table = gtk_table_new (4, 2, FALSE); gtk_table_attach_defaults
(GTK_TABLE (table), lbl1, 0, 1, 0, 1); gtk_table_attach_defaults (GTK_TABLE (table), lbl2, 0, 1, 1, 2);
gtk_table_attach_defaults (GTK_TABLE (table), lbl3, 0, 1, 2, 3); gtk_table_attach_defaults (GTK_TABLE (table),
lbl4, 0, 1, 3, 4); gtk_table_attach_defaults (GTK_TABLE (table), user, 1, 2, 0, 1); gtk_table_attach_defaults
(GTK_TABLE (table), real, 1, 2, 1, 2); gtk_table_attach_defaults (GTK_TABLE (table), home, 1, 2, 2, 3);
gtk_table_attach_defaults (GTK_TABLE (table), host, 1, 2, 3, 4); gtk_table_set_row_spacings (GTK_TABLE
(table), 5); gtk_table_set_col_spacings (GTK_TABLE (table), 5); gtk_container_set_border_width
(GTK_CONTAINER (table), 5); gtk_box_pack_start_defaults (GTK_BOX (GTK_DIALOG (dialog)->vbox),
table); gtk_widget_show_all (dialog); /* Run the dialog and output the data if the user clicks the OK button. */ result
= gtk_dialog_run (GTK_DIALOG (dialog)); if (result == GTK_RESPONSE_OK) { g_print ("User Name: %s\n",
gtk_entry_get_text (GTK_ENTRY (user))); g_print ("Real Name: %s\n", gtk_entry_get_text (GTK_ENTRY
(real))); g_print ("Home Folder: %s\n", gtk_entry_get_text (GTK_ENTRY (home))); g_print ("Host Name: %s\n",
gtk_entry_get_text (GTK_ENTRY (host))); } gtk_widget_destroy (dialog); return 0; }

switch()
 GTK+
 GtkAlertDialog, GtkAboutDialog, GtkFileChooserDialog, GtkFontSelectionDialog, GtkColorSelectionDialog
 GtkAlertDialog
 GtkAlertDialog
 GTK+
 uniform look
 5-3
 5-1
 5-3. GtkAlertDialog
 5-4. GtkWidget
 (messagedialogs.c) #include <gtk/gtk.h> static void button_clicked (GtkButton*,
GtkWidget *window, *button; gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "Message
Dialogs"); gtk_container_set_border_width (GTK_CONTAINER (window), 10); button =
gtk_button_new_with_mnemonic ("_Click Me"); g_signal_connect (G_OBJECT (button), "clicked",
G_CALLBACK (button_clicked), (gpointer) window); gtk_container_add (GTK_CONTAINER (window), button);
gtk_widget_show_all (window); gtk_main (); return 0; } /* Create a new message dialog that tells the user that the
button was clicked. */ static void button_clicked (GtkButton *button, GtkWidget *parent) { GtkWidget *dialog;
dialog = gtk_message_dialog_new (parent, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO,
GTK_BUTTONS_OK, "The button was clicked!"); gtk_window_set_title (GTK_WINDOW (dialog), "
Information"); gtk_dialog_run (GTK_DIALOG (dialog)); gtk_widget_destroy (dialog); }

gtk_message_dialog_new()
 GtkWidget
 GTK+
 GtkAlertDialog
 GtkAlertDialog
 centered
 GtkWidget*
 GtkWidget
 GtkWidget
 GTK+
 GTK_MESSAGE_INFO
 lightbulb
 (GTK_STOCK_DIALOG_INFO)
 GtkWidget
 GtkWidget
 
```



GTK\_MESSAGE\_INFO: 000000 000 0000 00 000. GTK\_MESSAGE\_WARNING: 000000 00 000 000000 00.

GTK\_MESSAGE\_QUESTION: 0000 000 0000 000 00. 000 000 000000 000 000 000000 00. GTK\_MESSAGE\_ERROR: 0000 000 00 000 00. GTK\_MESSAGE\_OTHER: 0000 000 000 00 000 00 00 0000 000 0000. 0000 0000 0 000 000000 00 000 00(0) 000 0000 00. 000 000 0 000 000 000 0000 00. 00 GTK\_MESSAGE\_QUESTION 0000 000000 GTK\_BUTTONS\_YES\_NO 00

GTK\_BUTTONS\_OK\_CANCEL 0000 0000 000 00 00 000 0 000 00 00 000000. 6 00 000 GtkButtonType 00 0000 000 00.

GTK\_BUTTONS\_NONE: 00 000 0000 00 000. GTK\_BUTTONS\_OK: GTK\_STOCK\_OK 000 0000.

GTK\_BUTTONS\_CLOSE: GTK\_STOCK\_CLOSE 000 0000. GTK\_BUTTONS\_CANCEL: GTK\_STOCK\_CANCEL 0 00 0000. GTK\_BUTTONS\_YES\_NO: GTK\_STOCK\_YES 000 GTK\_STOCK\_NO 000 0000.

GTK\_BUTTONS\_OK\_CANCEL: GTK\_STOCK\_OK 000 GTK\_STOCK\_CANCEL 000 0000. 0000 0000 GTK+ 00 00 00 0000 00 bigwise 0000 000 000 GTK\_MESSAGE\_DIALOG 00 0000 0000 000 00 0000 00 000000. 00 000 00 000 0000 0000 000000

GtkHButtonBox 0000000 000 0000 GtkDialog 00 000 000 0000 000 00000. gtk\_message\_dialog\_new() 000 00000(00 0000 000 00 00) 00000 00 000 00000. 0000 printf() 0000 00 0000 000000 00. 00 000 printf() 0000 00 000 000 000 0000 C 00 000000 000 0000 00. 0000

gtk\_message\_dialog\_new() 0000 0000 000 0000 0000 000 000 00 00 00. 000 000000 0000 00000 00 Pango Text Markup Language 0 0000 0000 gtk\_message\_dialog\_new\_with\_markup() 000 000000 000 0 00. 00 gtk\_message\_dialog\_new() 000 000000 0000

gtk\_message\_dialog\_set\_markup() 000 0 0000 0000 00 00000. void gtk\_message\_dialog\_set\_format\_secondary\_text (GtkMessageDialog \*dialog, const gchar \*message\_format, ...); 000 000000 20(secondary) 0000 0000 00 00000, 00 00 0 00 00000

gtk\_message\_dialog\_set\_format\_secondary\_text() 000 0000 000 0000. 00 000 000 000 0000 printf() 0000 000 000 0000. 00 000 10 (primary) 0000 000 0000 20 0000 000 000000 000 000 00 00000. gtk\_message\_dialog\_set\_format\_secondary\_markup() 000 20 00000

markup 000 00 00. 000000 000GtkAboutDialog 000 000000 000000000 00 000 0000 000 0 00 000 0000 00 000000. 000 000000 Help 0000

GTK\_STOCK\_ABOUT 000 0000 0000 00 00000. 000 000 0 9000 000000 00 000 000 000 000 000000 000 000 0000 000 0000.

GtkAboutDialog 000 000 0 00 0000 00 000 00. 0000 000000 00, 000, 00 00, 0000 00, 000, 00 000(documenter), 0000, 0000 0000. 00 000000

00 0 00 000 00 00 0000 00 000000 000000. 00 00 0000 000 000000, 00 00 5-400 000 0000(author credits) 00 0000 00. 00 5-4. 000000 000 0000

Credits 000 0000 000000 000, 00 000, 000, 000000 000 0000. 000(contributor) 0 000 000 000 00000. License 000 000 0000 000 0000 0 000000 00

00 0000. 000 5-50 GtkAboutDialog 000 00 000 000000 000 000000 0000 000 00000. 000 5-5. GtkAboutDialog 0000 (aboutdialogs.c)

```
#include <gtk/gtk.h>
int main (int argc, char *argv[]) {
    GtkWidget *dialog;
    GdkPixbuf *logo;
    GError *error = NULL;
    gtk_init (&argc, &argv);
    const gchar *authors[] = { "Author #1", "Author #2", NULL };
    const gchar *documenters[] = { "Documenter #1", "Documenter #2", NULL };
    dialog = gtk_about_dialog_new ();
    /* You should edit '/path/to/logo.png' to point to the location of logo.png *
    * from the chapter_5 source directory on your system. */
    logo = gdk_pixbuf_new_from_file ("/path/to/logo.png", &error);
    /* Set the application logo or handle the error. */
    if (error == NULL)
        gtk_about_dialog_set_logo (GTK_ABOUT_DIALOG (dialog), logo);
    else {
        if (error->domain == GDK_PIXBUF_ERROR)
            g_print ("GdkPixbufError: %s\n", error->message);
        else if (error->domain == G_FILE_ERROR)
            g_print ("GFileError: %s\n", error->message);
        else
            g_print ("An error in the domain: %d has occurred!\n", error->domain);
        g_error_free (error);
    }
    /* Set application data that will be displayed in the main dialog. */
    gtk_about_dialog_set_name (GTK_ABOUT_DIALOG (dialog), "GtkAboutDialog");
    gtk_about_dialog_set_version (GTK_ABOUT_DIALOG (dialog), "1.0");
    gtk_about_dialog_set_copyright (GTK_ABOUT_DIALOG (dialog), "(C) 2007 Andrew Krause");
    gtk_about_dialog_set_comments (GTK_ABOUT_DIALOG (dialog), "All About GtkAboutDialog");
    /* Set the license text, which is usually loaded from a file. Also, set the *
    * web site address and label. */
    gtk_about_dialog_set_license (GTK_ABOUT_DIALOG (dialog), "Free to all!");
    gtk_about_dialog_set_website (GTK_ABOUT_DIALOG (dialog), "http://book.andrewkrause.net");
    gtk_about_dialog_set_website_label (GTK_ABOUT_DIALOG (dialog), "book.andrewkrause.net");
    /* Set the application authors, documenters and translators. */
    gtk_about_dialog_set_authors (GTK_ABOUT_DIALOG (dialog), authors);
    gtk_about_dialog_set_documenters (GTK_ABOUT_DIALOG (dialog), documenters);
    gtk_about_dialog_set_translator_credits (GTK_ABOUT_DIALOG (dialog), "Translator #1\nTranslator #2");
    gtk_dialog_run (GTK_DIALOG (dialog));
    gtk_widget_destroy (dialog);
    return 0;
}
```

00000 GtkAboutDialog 00000 000 0 000 0 00 000000 00 00. 0 5-20 000 5-50 000 000 00000 0 00 00000. 000000 00000 00000 License 000 000 00 0000. 000000 0000 Credits 000 0000 00 0000. 00000Name 0000000000 00. Version 000000 00 00 000000 0 0 00 00. Copyright 00 0 00 0 0000 000 0 00 000 000 0000. Comments 00 00 0 0 0000 000 0 00 00000000 00 000 00. License 20 000000 0

0 00 0000 00. 00 NULL 0 0 License 000 000. Web Site 00000000 0000 URL. Web Site Label URL 00 0000 00. Authors 000000 000  
 00 0 0 0 000 NULL 000 00. Artists 000000 00 0000 0000 000000 NULL 000 00. Documenters 0000 0000 00 000000 NULL 000 00.  
**Translator Credits** 000 0000 0000(00) 0000 0000. **Logo** 000000 0000 0 **GdkPixbuf** 000 00000000 0000. 0 5-2. **GtkAboutDialog** 00 00  
 0, 0000, 00 00 0 000000 00 000 000000 00 0000 00000. 0 000 000 0000 00 00 00 000 0000 00000 00 00000. 0000 **gettext** 00 0000 000 000 0000. 0  
 00 000 [www.gnu.org/software/gettext/](http://www.gnu.org/software/gettext/) 00000. **GdkPixbufGdkPixbuf** 0000 000 0000 00 000 0000 00000. 00 0000 000 00000000 0000 000  
 0 000000 00 0000 00000 0000 000 0 000 0000. 0000 00 0000 0000 000 000 000000, 0 000000 00 0000 0000. **GdkPixbuf** **GObject** 000 00000 00  
**(referencing)** 0 00000. 00 **g\_object\_ref()** 000 00 000 00000000 0000 0 00 0000 000 0000 000 0 000 0000. **GdkPixbuf** 00(**pixbufs**) 000000 0  
 000 00 0000 00000. 00000 **pixbuf** 0000 0000 0000 5-5 000 **gdk\_pixbuf\_new\_from\_file()** 000 0 00. 0 000 00 000 0000 00 000 000 000 000  
 0 0 000 0000. **GdkPixbuf\* gdk\_pixbuf\_new\_from\_file (const char \*filename, GError \*\*error);** 0000 000 000  
**gtk\_pixbuf\_scale\_simple()** 000 000 000 0 00. 00 000 **GdkPixbuf** 000 00 0000000 0000, 00000 000 00(**interpolation**) 000 00000.  
**GdkPixbuf\* gdk\_pixbuf\_scale\_simple (const GdkPixbuf \*src, int destination\_width, int destination\_height,**  
**GdkInterpType interpolation);** 0 000 **GdkInterpType** 000 000 00. **GDK\_INTERP\_NEAREST:** 0000 00 000 00 0000 00000. 00 00  
 0 00 000 00000 0000 000 00 00. 0000 00 000 000000 00 00 000000 0 00! **GDK\_INTERP\_TILES:** 00 000 00 000 000 000000 000000, 00(**edge**) 0  
 0 00 00000(**anti-aliasing**) 00000. 00 **GDK\_INTERP\_NEAREST** 000 00000 00 00000 **GDK\_INTERP\_BILINEAR** 000 000 0000  
 0 0 0 00000. **GDK\_INTERP\_BILINEAR:** 0 000 0000 000 00 000 000 00000, 0000 000 00 0 000 0000 00000.  
**GDK\_INTERP\_HYPER:** 00 0 0 0 00 00 000 00 0000. 000 000 00 00 000 000000 00. 000 0000 00 000000 000 00000 00000000 00 00000 0 00. 000  
**gtk\_pixbuf\_new\_from\_file\_at\_size()** 000 000 0000 000000 00000 000 00 000 000 00000 00 0000 000 0 00. **GdkPixbuf** 00000000 00 00 00  
 0 0 000000, 000000 00 0 0 000 00 0000. **GdkPixbuf** 00 0 00 000 **API** 000 00000 00. **GError** 000 000 00000000 000 00000 00 0000. 00000000 000 000  
**GLib** 00 000 000 **GError** 00000 0000 00 000 00000. **struct GError { GQuark domain; gchar \*message; gint code; }; GError** 000  
 3 00 00 00000. **domain** 0 000 000 0000 00000 00000. 000 5-5 000 **GDK\_PIXBUF\_ERROR** **G\_FILE\_ERROR** 000000 000 00000.  
**switch()** 000 000 00000 0000 00 0000 0000. 000 00 00 00000 00 000 00000 0000 00. 00 00000 000 0 000000 000000 00 0000, 0 0000 **case** 00 00 000000 00000.  
**message** 00 000 000000 00000 0000 00 0 00 0000000. 000 00 0000000 0000 0000 000 00 000 00 00 0000 00000. 0 0000 0000 **g\_error\_free()** 0000 00  
**(free)** 000. 000 000 **code** 000 00000 00000 00 0000. 00 00 0 5-3 **GDK\_PIXBUF\_ERROR** 000000 000 0 00 6 00 000 000 00000. 00 000 000 00  
 000 000000 **GdkPixbuf** 00000 0000 00 000 0 00 00 0000. 00 0000 **GDK\_PIXBUF\_ERROR\_CORRUPT\_IMAGE** 0000 000 0000 0000 0000.  
**GDK\_PIXBUF\_ERROR\_INSUFFICIENT\_MEMORY** 00000 00000 00 000 0 00 0000 00000.  
**GDK\_PIXBUF\_ERROR\_BAD\_OPTION** 00000 00 000 000000. 00 000 00000 00 0 000 0 00.  
**GDK\_PIXBUF\_ERROR\_UNKNOWN\_TYPE** **GdkPixbuf** 000 000 000 0 0000.  
**GDK\_PIXBUF\_ERROR\_UNSUPPORTED\_OPERATION** **GdkPixbuf** 000 000000 000 00000 000 000 000 0000.  
**GDK\_PIXBUF\_ERROR\_FAILED** 00 000 00 000 00 00 00. 0 5-3. **GdkPixbufError** 00 0 **GLib** 00 000 000 00 000 00000. 00 0000 00  
 <NAMESPACE>\_<MODULE>\_ERROR 000 0000000, 00000000 000 00000 0000000 00000, 000 00000 00 000 00000. 000000 00 00 000 00  
 0 0 0 000 00000. 00 00 00 00000 <NAMESPACE>\_<MODULE>\_ERROR\_FAILED 0000 0000, 00 0000 0000 00 0000. 00 000 000 0  
 00 00 0 00 000 0000. 00 000 00000 000 00 00000 0000 00 00 00000 0000, 00 00 000 000000 000000 00 000 0000000 00 0000. 000 000 0 00 00 0000 0000. 0 0  
 0 0 00 0 0 000 00000 00 000 00 00 0000 00000. **GError** 00000 000 0(**piling up**) 000 0 00 000 00000. 0 00 00 00000 000 **GError** 000 000 00  
 0 00 0 0 0 0 000 000 0000 0000. 000 00 000 000 00000. 000 000 00000 00000 000 00000 0000 0000 00000 0000. 00 **g\_clear\_error()** 000 **GError** 00  
 00 00 000 0 0000. 0 00000 00 000 **GError** 000 00000 0 00. **if (error && \* error) { g\_error\_free (\*error); \*error = NULL; }**  
**g\_clear\_error()** 0 000 0 0 00000 00 00 000 000 000000 00 00000. 000 000 00, **g\_error\_free()** 0000 000 00000 00 0000 0 **GError** 000 000 00  
**(slice)** 00000. 00 0 00 **NULL** 00000. **GTK+** 000 0000 000000000 00 000 00 00 000 00000 00 0000 00 000 00 **Ed** 00 0 00. 00 000 000000 00 00  
**GtkFileChooser** 000 **GtkFileChooserButton** 000 000 0 00. **GtkFileChooser** 000 000 00000000 00 00000000. 0000000 000000 0000, 0000000000 0  
 00 000000 000000 0000 00 00 000 0000 00 00000. **GTK+** **GtkFileChooser** 0000000 00000 00 0 00 000 00000. **GtkFileChooserButton:** 00 000  
 000 0 000 00000. 00 00 0 **GtkFileChooser** 000000 00000000 00000 000 00000 000 000000 0000. **GtkFileChooserDialog:** 00 000  
**GtkFileChooserWidget** 00 000000 00 0 0 000000 00000. 00 **GtkFileChooser** 0000000 000000, 0000 0 00 0000 00 0000 000 00.  
**GtkFileChooserWidget:** 00000 000 00000 000 000000 00 0 00 00000. 000 00 00 000 000 0000 0000 00. **GtkFileChooserDialog** 000 00 000  
**GtkDialog** 000 00 000 000 0000 0000. **GtkFileChooserButton** 00 00 000000 00 0000 000 000 000 000 000000 000000, 00 000 000 0000 0000 00  
 000 0 00. 00 0 00 000 00 000 000 00 000 000 00000, 00000 00000, 000 000 0000 000 00 0000. 00 0000000 5-5 000 00000 00 00000  
**GtkFileChooserDialog** 000 00000. 000 000 0 00 00 0 000 00000 000 0000, 00 000 000000 000 00 000 000 00 000 000 00000 000 00000 000 00000 000  
 000 00000. 000 000 00000 000000 00 000 000 000 0 000 000 00 00. 00 5-5. 00 000 00000 00 000 00000000 000 000000 0 00 000 00 0 00 000000 000 00000 0  
 0 00, **gtk\_dialog\_new()** 000 000 00 000 00000 000 00 0 000 00. 000 5-6 0000 00000 00000, 00 00000 00000 000 00000 0 000000 000000 0000. 000

```

5-6. GtkFileChooserDialog (savefile.c) #include <gtk/gtk.h> static void button_clicked (GtkButton*,
GtkWindow*); int main (int argc, char *argv[]) { GtkWidget *window, *button; gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "Save a
File"); gtk_container_set_border_width (GTK_CONTAINER (window), 10); gtk_widget_set_size_request
(window, 200, 100); button = gtk_button_new_with_label ("Save As ..."); g_signal_connect (G_OBJECT (button), "
clicked", G_CALLBACK (button_clicked), (gpointer) window); gtk_container_add (GTK_CONTAINER
(window), button); gtk_widget_show_all (window); gtk_main (); return 0; } /* Allow the user to enter a new file
name and location for the file and * set the button to the text of the location. */ static void button_clicked
(GtkButton *button, GtkWindow *window) { GtkWidget *dialog; gchar *filename; dialog =
gtk_file_chooser_dialog_new ("Save File As ...", window, GTK_FILE_CHOOSER_ACTION_SAVE,
GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL, GTK_STOCK_SAVE, GTK_RESPONSE_ACCEPT,
NULL); gint result = gtk_dialog_run (GTK_DIALOG (dialog)); if (result == GTK_RESPONSE_ACCEPT) {
filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog)); gtk_button_set_label (button,
filename); } gtk_widget_destroy (dialog); }
GtkWidget*
gtk_file_chooser_dialog_new (const gchar *title, GtkWindow *parent, GtkFileChooserAction action, const gchar
*first_button_text, ...);
GtkFileChooserDialog GTK_FILE_CHOOSER_ACTION_SAVE:
GTK_FILE_CHOOSER_ACTION_OPEN:
GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER:
GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER:
Create Folder
5-6 Cancel
GTK_RESPONSE_CANCEL
GTK_RESPONSE_ACCEPT
5-7
5-6
5-7
5-6
5-7
5-7. GtkFileChooserDialog (createfolder.c) #include <gtk/gtk.h> int main (int argc, char
*argv[]) { GtkWidget *dialog; gchar *filename; gint result; gtk_init (&argc, &argv); /* Create a new
GtkFileChooserDialog that will be used to create a new folder. */ dialog = gtk_file_chooser_dialog_new ("Create a
Folder ...", NULL, GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER, GTK_STOCK_CANCEL,
GTK_RESPONSE_CANCEL, GTK_STOCK_OK, GTK_RESPONSE_OK, NULL); result = gtk_dialog_run
(GTK_DIALOG (dialog)); if (result == GTK_RESPONSE_OK) { filename = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (dialog)); g_print ("Creating directory: %s\n", filename); } gtk_widget_destroy (dialog);
return 0; }
GLib g_mkdir()
5-7
GTK_FILE_CHOOSER_ACTION_OPEN
GtkFileChooserButton
GtkFileChooserDialog
5-7
5-8
5-8. GtkFileChooserDialog (multiplefiles.c) static void button_clicked
(GtkButton *button, GtkWindow *window) { GtkWidget *dialog; GSList *filenames; dialog =
gtk_file_chooser_dialog_new ("Open File(s) ...", window, GTK_FILE_CHOOSER_ACTION_OPEN,
GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL, GTK_STOCK_OPEN, GTK_RESPONSE_ACCEPT,
NULL); /* Allow the user to choose more than one file at a time. */ gtk_file_chooser_set_select_multiple
(GTK_FILE_CHOOSER (dialog), TRUE); gint result = gtk_dialog_run (GTK_DIALOG (dialog)); if (result ==
GTK_RESPONSE_ACCEPT) { filenames = gtk_file_chooser_get_filenames (GTK_FILE_CHOOSER (dialog));
while (filenames != NULL) { gchar *file = (gchar*) filenames->data; g_print ("%s was selected.\n", file);
filenames = filenames->next; } } gtk_widget_destroy (dialog); }
GLib GSList
gchar *file = (gchar*)

```

```

filenames->data; GLib 00 00 0000 0000 gpointers 000000 00 0000 0000 0 00. 0 000 g_slist_nth_data()0000 0000 0000 00 0000 000000 00
0000 00. 0000 0 0 00 00 0000 0000. GSList 0000 0000 00, 0000 00 00, 00 00, 00, 0000 0 0000 0000 0000 00. 00 0 00 00 0000 00 0 00 0000 00 0000 00
0 00. 00 00 000000 0000 0 0000 0000 0000 0000 GtkColorButton 0000 00 000000. 0000 0 0000 000000 000000. 0000 0000 0000 0 000000
GtkColorSelectionDialog 00 0 00. 00 00 000000 GtkFileChooserDialog 000000, 0000 GtkDialog 000000 GtkColorSelection 0000 00
0 000000 0000 0000. GtkColorSelection 0 00 000000 00 000000. 0000 000000 0000 000000 000000 0000 GTK+ 0000 GtkColorSelectionDialog 0
0 0 0. 00 00 000000 00 5-8 000000. 00 5-8. 0 0 00 00000000 5-9 0 00 0000 00 0000 0000 0000 0000. 0 00 0000 0000 0000
GtkColorSelectionDialog 0 000000. 0000 0000 000000 GtkColorSelectionDialog 0000 0000. 0 0000 00 00 0000 00 000000 00 000000. 00 0000 00
00 0000 000000 00 00 00 00 0 00 00 00. 0000 00000000 0000 0 000000 00 0000 0 00. 0000 5-9. GtkColorSelectionDialog 000000
(colorselection.c) #include <gtk/gtk.h> static void run_color_selection_dialog (GtkButton*, GtkWidget*,
gboolean); static void modal_clicked (GtkButton*, GtkWidget*); static void nonmodal_clicked (GtkButton*,
GtkWidget*); static void dialog_response (GtkDialog*, gint, gpointer); static GdkColor global_color; static guint
global_alpha = 65535; int main (int argc, char *argv[]) { GtkWidget *window, *hbox, *modal, *nonmodal; gint i;
gtk_init (&argc, &argv); /* Loop through the parameters. The first color name that is specified and * successfully
parsed, it will be used as the initial color of the selection. */ for (i=1; i < argc; i++) if (gdk_color_parse (argv[i],
&global_color)) break; window = gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title
(GTK_WINDOW (window), "Color Selection Dialogs"); gtk_container_set_border_width (GTK_CONTAINER
(window), 10); gtk_widget_set_size_request (window, 200, 75); modal = gtk_button_new_with_label ("Modal");
nonmodal = gtk_button_new_with_label ("Non-Modal"); g_signal_connect (G_OBJECT (modal), "clicked",
G_CALLBACK (modal_clicked), (gpointer) window); g_signal_connect (G_OBJECT (nonmodal), "clicked",
G_CALLBACK (nonmodal_clicked), (gpointer) window); hbox = gtk_hbox_new (TRUE, 10);
gtk_box_pack_start_defaults (GTK_BOX (hbox), modal); gtk_box_pack_start_defaults (GTK_BOX (hbox),
nonmodal); gtk_container_add (GvtTK_CONTAINER (window), hbox); gtk_widget_show_all (window); gtk_main
(); return 0; } /* Create a new color selection dialog that is modal. */ static void modal_clicked (GtkButton *button,
GtkWidget *window) { run_color_selection_dialog (button, window, TRUE); } /* Create a new color selection
dialog that is nonmodal. */ static void nonmodal_clicked (GtkButton *button, GtkWidget *window) {
run_color_selection_dialog (button, window, FALSE); } /* Create a new color selection dialog and allow the user to
choose a color * and an opacity value. */ static void run_color_selection_dialog (GtkButton *button, GtkWidget
*window, gboolean domodal) { GtkWidget *dialog, *colorsel; gchar *title; if (domodal) title = "Choose Color --
Modal"; else title = "Choose Color -- Non-Modal"; dialog = gtk_color_selection_dialog_new (title);
gtk_window_set_modal (GTK_WINDOW (dialog), domodal); colorsel = GTK_COLOR_SELECTION_DIALOG
(dialog)->colorsel; gtk_color_selection_set_has_opacity_control (GTK_COLOR_SELECTION (colorsel), TRUE);
gtk_color_selection_set_current_color (GTK_COLOR_SELECTION (colorsel), &global_color);
gtk_color_selection_set_current_alpha (GTK_COLOR_SELECTION (colorsel), global_alpha); g_signal_connect
(G_OBJECT (dialog), "response", G_CALLBACK (dialog_response), NULL); gtk_widget_show_all (dialog); } /*
Handle the response identifier from the assistant. Either tell the user to * read the manual, retrieve the new color
value or destroy the dialog. */ static void dialog_response (GtkDialog *dialog, gint result, gpointer data) {
GtkWidget *colorsel; GdkColor color = { 0, }; guint16 alpha = 0; switch (result) { case GTK_RESPONSE_HELP:
g_print("Read the GTK+ API documentation.\n"); break; case GTK_RESPONSE_OK: colorsel =
GTK_COLOR_SELECTION_DIALOG (dialog)->colorsel; alpha = gtk_color_selection_get_current_alpha
(GTK_COLOR_SELECTION (colorsel)); gtk_color_selection_get_current_color (GTK_COLOR_SELECTION
(colorsel), &color); g_print ("#%04X%04X%04X%04X\n", color.red, color.green, color.blue, alpha); global_color
= color; global_alpha = alpha; default: gtk_widget_destroy (GTK_WIDGET(dialog)); } } GtkWidget *colorsel;
GtkWidget *ok_button; GtkWidget *cancel_button; GtkWidget *help_button; };
GtkColorSelectionDialog 0 00 00 0000 00 0000 00 0000 0000. 00 colorsel 00 0000 0000 0000 GtkColorSelection 000000. 0000 0 00
GTK_STOCK_OK, GTK_STOCK_CANCEL, GTK_STOCK_HELP 000000. 000000 Help 0000 000000. 00 000000
gtk_widget_show() 0000 0 00. 0000 5-20 00 0 0000 response 000000 000000, 00 000000 000000 000000 0000 00 0000 00 0000 00 000000. 000000

```

`gtk_window_set_modal()`은 모달 창을 생성합니다. `0-65,535` RGB 값을 지정할 수 있으며, `0-1.0`의 불투명도를 지정할 수 있습니다. `0-65,535`의 값은 `0-1.0`의 값을 곱하여 얻습니다.

`gtk_color_selection_set_has_opacity_control()`은 `void gtk_color_selection_set_has_opacity_control(GtkColorSelection *colorsel, gboolean has_opacity);`를 호출하여 `16비트`의 불투명도를 설정할 수 있습니다. `GtkColorSelection` 객체에서 `gtk_color_selection_get_current_alpha()`를 호출하여 현재 불투명도를 가져옵니다. `g_print("#%04X%04X%04X%04X\n", color.red, color.green, color.blue, alpha);`를 사용하여 색상과 불투명도를 출력합니다. `GtkFontButton`은 `GtkColorSelectionDialog`와 `GtkFontSelectionDialog`를 포함합니다.

`5-10`의 `GtkFontSelectionDialog` 예제를 보십시오. `5-10`의 `GtkFontSelectionDialog` 예제를 보십시오. `5-10`의 `GtkFontSelectionDialog` 예제를 보십시오.

```
(GTK_FONT_SELECTION_DIALOG (dialog), "Sans Bold Italic 12");
gtk_font_selection_dialog_set_preview_text (GTK_FONT_SELECTION_DIALOG (dialog), "Foundations of
GTK+ Development"); g_signal_connect (G_OBJECT (dialog), "response", G_CALLBACK
(font_dialog_response), NULL); gtk_widget_show_all (dialog); gtk_main (); return 0; } /* If the user clicks "
Apply", display the font, but do not destroy the dialog. If "OK" is pressed, display the font and destroy the dialog.
Otherwise, just destroy the dialog. */ static void font_dialog_response (GtkFontSelectionDialog *dialog, gint
response, gpointer data) { gchar *font; GtkWidget *message; switch (response) { case
(GTK_RESPONSE_APPLY): case (GTK_RESPONSE_OK): font = gtk_font_selection_dialog_get_font_name
(dialog); message = gtk_message_dialog_new (NULL, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO,
GTK_BUTTONS_OK, font); gtk_window_set_title (GTK_WINDOW (message), "Selected Font"); gtk_dialog_run
(GTK_DIALOG (message)); gtk_widget_destroy (message); g_free (font); break; default: gtk_widget_destroy
(GTK_WIDGET (dialog)); } if (response == GTK_RESPONSE_OK) gtk_widget_destroy (GTK_WIDGET
(dialog)); }
struct
GtkFontSelectionDialog { GtkWidget *ok_button; GtkWidget *apply_button; GtkWidget *cancel_button; };
GTK_STOCK_OK, GTK_STOCK_APPLY, GTK_STOCK_CANCEL를 사용하여 버튼을 설정합니다. response는 OK, Apply, Cancel를 나타냅니다. GtkEntry는 텍스트 입력을 위한 widget입니다. "abcdefghijk ABCDEFGHIJK"을 설정합니다. "Foundations of GTK+
Development"를 설정합니다. GtkFontSelectionDialog는 gtk_font_selection_dialog_get_font_name()를 사용하여 현재 선택된 폰트를 가져옵니다.
```

`gtk_font_selection_dialog_set_font_name()`과 `gtk_font_selection_dialog_get_font_name()`은 `PangoFontDescription` 객체를 사용하여 폰트를 설정하고 가져옵니다. `GTK+ 2.10`에서는 `GtkAssistant`을 사용하여 복잡한 작업을 안내할 수 있습니다. `5-10`의 `GtkAssistant` 예제를 보십시오.

`5-10`의 `GtkAssistant` 예제를 보십시오. `5-11`의 `GtkAssistant` 예제를 보십시오. `5-11`의 `GtkAssistant` 예제를 보십시오. `5-11`의 `GtkAssistant` 예제를 보십시오.

```
(assistant.c) #include <gtk/gtk.h> #include <string.h> static void entry_changed (GtkEditable*, GtkAssistant*);
static void button_toggled (GtkCheckButton*, GtkAssistant*); static void button_clicked (GtkButton*,
GtkAssistant*); static void assistant_cancel (GtkAssistant*, gpointer); static void assistant_close (GtkAssistant*,
gpointer); typedef struct { GtkWidget *widget; gint index; const gchar *title; GtkAssistantPageType type; gboolean
complete; } PageInfo; int main (int argc, char *argv[]) { GtkWidget *assistant, *entry, *label, *button, *progress,
*hbox; guint i; PageInfo page[5] = { { NULL, -1, "Introduction", GTK_ASSISTANT_PAGE_INTRO, TRUE }, {
NULL, -1, NULL, GTK_ASSISTANT_PAGE_CONTENT, FALSE }, { NULL, -1, "Click the Check Button",
GTK_ASSISTANT_PAGE_CONTENT, FALSE }, { NULL, -1, "Click the Button",
GTK_ASSISTANT_PAGE_PROGRESS, FALSE }, { NULL, -1, "Confirmation",
```

```

GTK_ASSISTANT_PAGE_CONFIRM, TRUE}, }; gtk_init (&argc, &argv); /* Create a new assistant widget with
no pages. */ assistant = gtk_assistant_new (); gtk_widget_set_size_request (assistant, 450, 300);
gtk_window_set_title (GTK_WINDOW (assistant), "GtkAssistant Example"); g_signal_connect (G_OBJECT
(assistant), "destroy", G_CALLBACK (gtk_main_quit), NULL); page[0].widget = gtk_label_new ("This is an
example of a GtkAssistant. By\n\n" "clicking the forward button, you can continue\n\n" "to the next section!");
page[1].widget = gtk_hbox_new (FALSE, 5); page[2].widget = gtk_check_button_new_with_label ("Click Me To
Continue!"); page[3].widget = gtk_alignment_new (0.5, 0.5, 0.0, 0.0); page[4].widget = gtk_label_new ("Text has
been entered in the label and the\n\n" "combo box is clicked. If you are done, then\n\n" "it is time to leave!"); /* Create
the necessary widgets for the second page. */ label = gtk_label_new ("Your Name: "); entry = gtk_entry_new ();
gtk_box_pack_start (GTK_BOX (page[1].widget), label, FALSE, FALSE, 5); gtk_box_pack_start (GTK_BOX
(page[1].widget), entry, FALSE, FALSE, 5); /* Create the necessary widgets for the fourth page. The, Attach the
progress bar * to the GtkAlignment widget for later access. */ button = gtk_button_new_with_label ("Click me!");
progress = gtk_progress_bar_new (); hbox = gtk_hbox_new (FALSE, 5); gtk_box_pack_start (GTK_BOX (hbox),
progress, TRUE, FALSE, 5); gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 5);
gtk_container_add (GTK_CONTAINER (page[3].widget), hbox); g_object_set_data (G_OBJECT (page[3].widget), "
pbar", (gpointer) progress); /* Add five pages to the GtkAssistant dialog. */ for (i = 0; i < 5; i++) {
page[i].index = gtk_assistant_append_page (GTK_ASSISTANT (assistant), page[i].widget);
gtk_assistant_set_page_title (GTK_ASSISTANT (assistant), page[i].widget, page[i].title);
gtk_assistant_set_page_type (GTK_ASSISTANT (assistant), page[i].widget, page[i].type); /* Set the introduction
and conclusion pages as complete so they can be * incremented or closed. */ gtk_assistant_set_page_complete
(GTK_ASSISTANT (assistant), page[i].widget, page[i].complete); } /* Update whether pages 2 through 4 are
complete based upon whether there is * text in the GtkEntry, the check button is active, or the progress bar * is
completely filled. */ g_signal_connect (G_OBJECT (entry), "changed", G_CALLBACK (entry_changed),
(gpointer) assistant); g_signal_connect (G_OBJECT (page[2].widget), "toggled", G_CALLBACK (button_toggled),
(gpointer) assistant); g_signal_connect (G_OBJECT (button), "clicked", G_CALLBACK (button_clicked),
(gpointer) assistant); g_signal_connect (G_OBJECT (assistant), "cancel", G_CALLBACK (assistant_cancel),
NULL); g_signal_connect (G_OBJECT (assistant), "close", G_CALLBACK (assistant_close), NULL);
gtk_widget_show_all (assistant); gtk_main (); return 0; } /* If there is text in the GtkEntry, set the page as complete.
Otherwise, * stop the user from progressing the next page. */ static void entry_changed (GtkEditable *entry,
GtkAssistant *assistant) { const gchar *text = gtk_entry_get_text (GTK_ENTRY (entry)); gint num =
gtk_assistant_get_current_page (assistant); GtkWidget *page = gtk_assistant_get_nth_page (assistant, num);
gtk_assistant_set_page_complete (assistant, page, (strlen (text) > 0)); } /* If the check button is toggled, set the page
as complete. Otherwise, * stop the user from progressing the next page. */ static void button_toggled
(GtkCheckButton *toggle, GtkAssistant *assistant) { gboolean active = gtk_toggle_button_get_active
(GTK_TOGGLE_BUTTON (toggle)); gtk_assistant_set_page_complete (assistant, GTK_WIDGET (toggle),
active); } /* Fill up the progress bar, 10% every second when the button is clicked. Then, * set the page as complete
when the progress bar is filled. */ static void button_clicked (GtkButton *button, GtkAssistant *assistant) {
GtkProgressBar *progress; GtkWidget *page; gdouble percent = 0.0; gtk_widget_set_sensitive (GTK_WIDGET
(button), FALSE); page = gtk_assistant_get_nth_page (assistant, 3); progress = GTK_PROGRESS_BAR
(g_object_get_data (G_OBJECT (page), "pbar")); while (percent <= 100.0) { gchar *message = g_strdup_printf
("%0.f%% Complete", percent); gtk_progress_bar_set_fraction (progress, percent / 100.0);
gtk_progress_bar_set_text (progress, message); while (gtk_events_pending ()) gtk_main_iteration (); g_usleep
(500000); percent += 5.0; } gtk_assistant_set_page_complete (assistant, page, TRUE); } /* If the dialog is
cancelled, delete it from memory and then clean up after * the Assistant structure. */ static void assistant_cancel
(GtkAssistant *assistant, gpointer data) { gtk_widget_destroy (GTK_WIDGET (assistant)); } /* This function is
where you would apply the changes and destroy the assistant. */ static void assistant_close (GtkAssistant *assistant,
gpointer data) { g_print ("You would apply your changes now!\n"); gtk_widget_destroy (GTK_WIDGET

```



pulsing)을 호출하여 깜빡이게 합니다. assistant 객체를 생성하고 assistant 객체를 forward로 호출하여 GtkAssistantPageFunc 객체를 호출합니다.

```

gtk_assistant_set_forward_page_func()은 assistant 객체를 호출하여 forward로 호출하여 GtkAssistantPageFunc 객체를 호출합니다.
void gtk_assistant_set_forward_page_func (GtkAssistant *assistant,
GtkAssistantPageFunc page_func, gpointer data, GDestroyNotify destroy_func);
assistant_forward()은
decide_next_page()은 2, 3, 4로 호출하여 GtkAssistantPageFunc 객체를 호출합니다.
static gint assistant_forward
(gint current_page, gpointer data) { gint next_page = 0; switch (current_page) { case 0: next_page = 1; break; case
1: next_page = (decide_next_page() ? 2 : 3); break; case 2: case 3: next_page = 4; break; default: next_page = -1; }
return next_page; }
assistant_forward()은 -1로 호출하여 assistant 객체를 호출하여 assistant_forward()은 4로 호출하여 assistant_forward()은
GTK+의 GtkAssistant 객체를 호출하여 GtkAssistant 객체를 호출하여 GtkAssistant 객체를 호출하여 GtkAssistant 객체를 호출하여
GtkNotebook, GtkFileChooserWidget, GtkDialog, GtkFileChooserDialog, GtkDialog, GtkFileChooserWidget, GTK_FILE_CHOOSER_ACTION_SAVE,
GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER, g_mkdir(), GTK_FILE_CHOOSER_ACTION_OPEN, GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
GtkVBox, gtk_dialog_run(), GtkMessageDialog, GtkAboutDialog, GtkFileChooserDialog, GtkColorSelectionDialog, GtkFontSelectionDialog,
GTK+ 2.10의 GtkAssistant, assistant, GtkDialog, GtkWidget, gtk_dialog_run(), GLib, idle, timeouts, (process spawning),
(thread), Notes
  
```







GLib 2.12.1 0000 "2"0 major 000 0000. GLIB\_MINOR\_VERSION000 GLib 000 minor 00. 0000 000 000000 minor 000 0000 glib\_minor\_version0 0000 00. GLib 2.12.1 0000 "12"0 minor 000 0000. GLIB\_MINOR\_VERSION000 GLib 000 micro 00. 0 000 000 000000 micro 000 0000 glib\_micro\_version0 0000 00. GLib 2.12.1 0000 "1"0 micro 000 0000.

GLIB\_CHECK\_VERSION(major, minor, micro)0000 00 00 GLib 00 000 000 000 00 000000 000 000 00 000 00 TRUE0 0000. 00 00 0 0000 0 0000 GLib0 00 000 00 000 0000 00 000 0 00. 0 6-2. GLib 00 00 0 6-20 000 00 00 00 glib\_check\_version()0 0000 000 0 00 00 00 GLib0 000 000 0 00. 000000 0000 00 000 NULL0 0000, 000 000 000000 00 000 0 0000 0000 0000. 00 000 0000 000 000 GLib0 000 000000 0 00 000 000 0000. const gchar\* glib\_check\_version (guint major, guint minor, guint micro); GLib0 00 00, 00 00, 000 0000 000 000 Boolean 00 TRUE0 FALSE0 0000000 00 00 0000 0000 0000 0000 00. 0 0 000 0000 0 6-300 00 0 00. 000000ABS (a)00 a0 0000 0000. 000 0 0 00 00 000 000 00 00 0000 000. CLAMP (a, low, high)a0 low0 high 000 000 0000. a0 low0 high 000 00 00 00 000 00 20 00 000 00 0. 0 00 00 000 00 0000 00 0 000 0000. G\_DIR\_SEPARATORG\_DIR\_SEPARATOR\_SUNIX 0000 000000 000(/)0 0000 000 000000 00 0 0(\)0 0000. G\_DIR\_SEPARATOR0 000 000000 000 0000 G\_DIR\_SEPARATOR\_S0 0000 0000 000 0000. GINT\_TO\_POINTER (i)GPOINTER\_TO\_INT (p)000 gpointer0, 00 gpointer0 000 0000. 32 00 000 000 000 000 00 0000 000 00 32 000 00 000 0000 000 000 000 000 000 0000. 00 000 00 0 000000 000 0 00 0000. GSIZE\_TO\_POINTER (s)GPOINTER\_TO\_SIZE (p)gsizе 00 gpointer0, 00 gpointer0 gsize 000 0000. gsize 000 000 00 0000 000000 GSIZE\_TO\_POINTER()0 000 0000 000000 0000.

GINT\_TO\_POINTER()00 0 00 000 0000. GUINT\_TO\_POINTER (u)GPOINTER\_TO\_UINT (p)000 00 000 gpointer0, 00 gpointer0 000 00 000 0000. 00 0000 000000 GUINT\_TO\_POINTER()0 000 0000 000 0000. GINT\_TO\_POINTER()00 0 00 000 0000.

G\_OS\_WIN32G\_OS\_BEOSG\_OS\_UNIX0 0 00 0000 00 00000000 000 000 000000 0000. 0000 0000 0000 0000 000 000 000 #ifdef G\_OS\_\*0 000 0000 000000 000000 000 000 00 0 00. G\_STRUCT\_MEMBER(type, struct\_p, offset)000 offset0 000 0000 member0 0 0000. 00 offset0 struct\_p 00 000 00. type0 0000 00 00 000 000 0000 0000. G\_STRUCT\_MEMBER\_P(struct\_p, offset)000 offset0 0 00 0000 member0 000 000 0000 00 0000 00000. offset0 struct\_p 00 000 00. G\_STRUCT\_OFFSET(type, member)000 00 member 0 000 0000 0000. 000 000 type0 00 00000. MIN (a, b)MAX (a, b)00 00 00 a0 b0 00 0 0000 00 0000. TRUE 0 FALSEFALSE0 000 00000, FALSE0 000 0000 TRUE0 0000. 000 00 gboolean 000 0000. 0 6-3. 00 GLib 000 GLib0 00 000 000 0000 000000 0000, 00 0000 0 0000 0 00 500 0000 000000 00. GLib 2.120 000 0000 000 00. G\_E: 0000 000 490 000 00 000 0 G\_LN2: 0000 000 500 000 20 00 00 G\_LN10: 00 00 000 490 000 100 00 00 G\_PI: 0000 000 490 000 00(pi) 0 G\_PI\_2: 0000 000 490 000, 000 20 00 0 G\_PI\_4: 0000 000 500 000, 000 40 00 0 G\_SORT2: 0000 000 490 000 20 000 G\_LOG\_2\_BASE\_10: 0000 00 200 000, 00 100 20 00 000 0000 00 000 000 000 0000 000 00. 0 00 0000 g\_point()0 0000 000 GLib0 0 00 000 000 00 0000 0000. g\_log()0 0000 00 000 000 0000 000 0 00. 00 000 0 00 000000 000 0000 000000 0000. 00 0000 0000 00 00000000 000 00000000 0000 000000 0000 0000, GLogFunc0 00 00. void g\_log (const gchar \*log\_domain, GLogLevelFlags log\_level, const gchar \*message, ...); 000000 0000 00 0000 G\_LOG\_DOMAIN0 000000 0000 00. 00 000 000000 0000 000000 00 0000 00 000 00 000 000 0000 0000. 00 0000 0000 000 G\_LOG\_DOMAIN0 00 0 000. 00 GTK+ 000000 "Gtk"0 000000 0000 0000 0000 000000 0000 0 0 000 00. g\_log()0 0 00 000000 00 000 0000 000000 0000. 00 00, 000 0 000 0000 00 00 0000 0000 000 G\_LOG\_LEVEL\_ERROR0 0000 00. GLogLevelFlags0 0000 000 00.

G\_LOG\_FLAG\_RECURSION: 000 0000 0000 0000. G\_LOG\_FLAG\_FATAL: 0 0000 000 00 000 00 0 00000000 000 0000 00(core)0 000 0000. G\_LOG\_LEVEL\_ERROR: 000 0000 00 00. G\_LOG\_LEVEL\_CRITICAL: 0000 000 0000 000000 00000000 000 000 00. G\_LOG\_LEVEL\_WARNING: 0000000 000 0000 00 00 00 00. G\_LOG\_LEVEL\_MESSAGE: 0000 00 00 0000 000 00. G\_LOG\_LEVEL\_INFO: 00 0000 000 00 000 0000, 00 000 00 0 0 00. G\_LOG\_LEVEL\_DEBUG: 000 000 0000 00 000. G\_LOG\_LEVEL\_MASK: (G\_LOG\_FLAG\_RECURSION0G\_LOG\_FLAG\_FATAL)0 00. 00 00 g\_malloc()0 000 000 000 00 0000000 000000, G\_LOG\_LEVEL\_ERROR0 0000 0000. 00 g\_try\_malloc()0 00 0 000 0000 00 0000 0000 00 0000. 00 NULL 0000 00 00. g\_log()0 0000 00 00 0000 g\_print()0 000 000 000 000000 00. 0000 GLib0 g\_log()0 000 0 000 000000 000000 500 000 000000 00. 000 0000 00 0 0 000 00 g\_print()0 000 0000 0000000 00. 0000 0000 000 00 0000 000000 0000, G\_LOG\_DOMAIN 000000 000 0000. 0000 00 000 00 0000 000 0 0. void g\_message (...); /\* G\_LOG\_LEVEL\_MESSAGE \*/ void g\_warning (...); /\* G\_LOG\_LEVEL\_WARNING \*/ void g\_critical (...); /\* G\_LOG\_LEVEL\_CRITICAL \*/ void g\_error (...); /\* G\_LOG\_LEVEL\_ERROR \*/ void g\_debug (...); /\* G\_LOG\_LEVEL\_DEBUG \*/ 000000, 0000 00000000 0000 0000 000 00 00 000 000 (fatal)0 000 000 000 000 00. 000 000 G\_LOG\_LEVEL\_ERROR 0000 00000000 0000 0000. 00 00 0000 0 000 00 000000. 00 000 000 000000 000 0000 g\_log\_set\_always\_fatal()0 0000 00. 00 G\_LOG\_FLAG\_FATAL 0000 000 000 0000 0000. g\_log\_set\_always\_fatal (G\_LOG\_LEVEL\_DEBUG | G\_LOG\_LEVEL\_WARNING); 00 00, 00 00 000 0000 000000 000 0 00 0000 0000 00000000 000 00 00 000 0000 0000 0000 0000, 00 00 00 000 00000000 00 0000! 000 000000 000 00 0000000000 00 000 00000, 00000000 0000 000000 000 0 0 0 000 0000. GLib0 000 000 00 000 000 000000 00 0000 00 00 0000 000 0 00 0000 0000. 0000 0000GLib 2.10 00 0000 000 000 00 0 000 0000

(memory allocator) and how it works. The memory allocator is a library that provides a set of functions for allocating and freeing memory. The most commonly used memory allocator is **malloc()**, which is part of the **libc** library. Other memory allocators include **calloc()**, **realloc()**, and **free()**.

**GLib** provides a set of memory allocation functions that are designed to be more portable and easier to use than the standard C library functions. The **GLib** memory allocation functions are:

- g\_malloc()**: Allocates a block of memory of the specified size.
- g\_malloc0()**: Allocates a block of memory of the specified size, initialized to zero.
- g\_malloc0\_n()**: Allocates a block of memory of the specified size, initialized to zero, and returns a pointer to the first element of the array.
- g\_try\_malloc()**: Allocates a block of memory of the specified size, returning **NULL** if the allocation fails.
- g\_try\_malloc0()**: Allocates a block of memory of the specified size, initialized to zero, returning **NULL** if the allocation fails.
- g\_try\_malloc0\_n()**: Allocates a block of memory of the specified size, initialized to zero, returning **NULL** if the allocation fails, and returns a pointer to the first element of the array.
- g\_free()**: Frees a block of memory that was allocated by **g\_malloc()**, **g\_malloc0()**, or **g\_malloc0\_n()**.
- g\_memmove()**: Moves a block of memory from one location to another. It is similar to **memmove()** in the C library.
- g\_memset()**: Sets a block of memory to a specified value. It is similar to **memset()** in the C library.
- g\_strndup()**: Allocates a block of memory and copies a string of length **len** from **str** into it. It is similar to **strndup()** in the C library.
- g\_strdup()**: Allocates a block of memory and copies a string from **str** into it. It is similar to **strdup()** in the C library.
- g\_strdup\_printf()**: Allocates a block of memory and copies a string into it, formatted according to the format string **fmt** and the arguments **args**. It is similar to **strdup\_printf()** in the C library.

The **GLib** memory allocation functions are designed to be more portable and easier to use than the standard C library functions. They are also designed to be more efficient and to provide better error handling. For example, **g\_malloc0()** is more efficient than **calloc()** because it does not need to initialize the memory to zero. Similarly, **g\_free()** is more efficient than **free()** because it does not need to check if the pointer is **NULL** before freeing the memory.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_try\_malloc()** and **g\_try\_malloc0()** return **NULL** if the allocation fails, which allows you to handle the error more gracefully. Similarly, **g\_strndup()** and **g\_strdup()** return **NULL** if the allocation fails, which allows you to handle the error more gracefully.

The **GLib** memory allocation functions are also designed to be more efficient and to provide better error handling. For example, **g\_memmove()** is more efficient than **memmove()** because it does not need to check if the source and destination pointers are valid. Similarly, **g\_memset()** is more efficient than **memset()** because it does not need to check if the pointer is valid.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_strndup()** and **g\_strdup()** return **NULL** if the allocation fails, which allows you to handle the error more gracefully. Similarly, **g\_strdup\_printf()** returns **NULL** if the allocation fails, which allows you to handle the error more gracefully.

The **GLib** memory allocation functions are also designed to be more efficient and to provide better error handling. For example, **g\_memmove()** is more efficient than **memmove()** because it does not need to check if the source and destination pointers are valid. Similarly, **g\_memset()** is more efficient than **memset()** because it does not need to check if the pointer is valid.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_strndup()** and **g\_strdup()** return **NULL** if the allocation fails, which allows you to handle the error more gracefully. Similarly, **g\_strdup\_printf()** returns **NULL** if the allocation fails, which allows you to handle the error more gracefully.

```

#define SLICE_SIZE 10
gchar *strings[100];
gint i;
for (i = 0; i < 100; i++)
    strings[i] = g_slice_alloc(SLICE_SIZE);
/* ... Use the strings in some way ... */
/* Free all of the memory after you are done using it. */
for (i = 0; i < 100; i++)
    g_slice_free1(SLICE_SIZE, strings[i]);

```

The **GLib** memory allocation functions are also designed to be more efficient and to provide better error handling. For example, **g\_slice\_alloc()** and **g\_slice\_free1()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0\_n()** and **g\_slice\_free0\_n()** are more efficient than **g\_malloc0\_n()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

```

typedef struct {
    GtkWidget *window;
    GtkWidget *label;
} Widgets;
Widgets *w = g_slice_new(Widgets);
/* Use the structure just as you would any other structure. */
w->window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
w->label = gtk_label_new("I belong to widgets!");
/* Free the block of memory of size "Widgets" so it can be reused. */
g_slice_free(Widgets, w);

```

The **GLib** memory allocation functions are also designed to be more efficient and to provide better error handling. For example, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new0\_n()** and **g\_slice\_free0\_n()** are more efficient than **g\_malloc0\_n()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

```

#define g_slice_new(type) ((type*) g_slice_alloc(sizeof(type))
g_slice_new() g_slice_alloc() g_slice_new0() g_slice_new0_n()
g_slice_free() w Widgets g_malloc() g_malloc0() g_malloc0_n()
g_malloc() g_malloc0() g_malloc0_n() g_try_malloc() g_try_malloc0() g_try_malloc0_n()
g_free() GTK+ API void g_free(gpointer memory);
g_memmove() g_memset() g_strndup() g_strdup() g_strdup_printf()
g_slice_alloc() g_slice_free1() g_slice_new() g_slice_free()
g_slice_new0() g_slice_free0() g_slice_new0_n() g_slice_free0_n()
g_malloc() g_malloc0() g_malloc0_n() g_try_malloc() g_try_malloc0() g_try_malloc0_n()
g_free() GTK+ API void g_free(gpointer memory);
g_memmove() g_memset() g_strndup() g_strdup() g_strdup_printf()

```

The **GLib** memory allocation functions are also designed to be more efficient and to provide better error handling. For example, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new0\_n()** and **g\_slice\_free0\_n()** are more efficient than **g\_malloc0\_n()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

The **GLib** memory allocation functions are also designed to be more flexible and to provide better error handling. For example, **g\_slice\_new()** and **g\_slice\_free()** are more efficient than **g\_malloc()** and **g\_free()** because they are designed to be used in a loop. Similarly, **g\_slice\_new0()** and **g\_slice\_free0()** are more efficient than **g\_malloc0()** and **g\_free()** because they are designed to be used in a loop.

```

#include <glib.h>
int main (int argc, char *argv[]) {

```

```

GSLList *list = NULL; /* Set the GMemVTable to the default table. This needs to be called before * any other call to
a GLib function. */ g_mem_set_vtable (glib_mem_profiler_table); /* Call g_mem_profile() when the application
exits. */ g_atexit (g_mem_profile); list = (GSLList*) g_malloc (sizeof (GSLList)); list->next = (GSLList*) g_malloc
(sizeof (GSLList)); /* Only free one of the GSLList objects to see the memory profiler output. */ g_free (list->next);
return 0; }
=====
8|2|1|0|0|+8 GLib Memory statistics (falling operations): --- none --- Total bytes: allocated=16,
zero-initialized=0 (0.00%), freed=8 (50.00%), remaining=8
=====

```

The text continues with a detailed explanation of the memory profiler output and the `glib_mem_profiler_table` structure. It describes how the profiler tracks memory operations like `malloc()`, `realloc()`, and `free()` for various GLib objects. The output shows that 16 bytes were allocated, 8 were freed, and 8 remain. The remaining bytes are categorized by operation type: 8 bytes from successful operations and 0 bytes from falling operations.

```

g_get_current_dir() returns the current directory. g_get_home_dir() returns the user's home directory. Windows returns
HOME or USERPROFILE or %HOMEDRIVE%\%HOMEPATH% depending on the platform. UNIX returns passwd or $HOME. g_get_host_name()
returns the host name. g_get_real_name() returns the real name. UNIX returns passwd or $USER. g_get_tmp_dir() returns the
/tmp directory. g_get_user_name() returns the user name. Windows returns %USERNAME%. UTF-8 encoding is used for
UNIX. g_getenv() returns the value of an environment variable. g_setenv() sets the value of an environment variable. g_unsetenv()
removes an environment variable. g_timer_new() creates a new GTimer. g_timer_start() starts a timer. g_timer_stop() stops a
timer. g_timer_continue() continues a timer. g_timer_elapsed() returns the elapsed time. g_timer_elapsed() returns the elapsed
time in microseconds. g_timer_elapsed() returns the elapsed time in fractional seconds.

```

microseconds. `g_timer_reset()` resets the timer to 0. `g_timer_start()` starts the timer. `g_timer_destroy()` destroys the timer.

**6-5** `g_file_set_contents()` sets the contents of a file. `g_file_get_contents()` gets the contents of a file. `g_file_test()` tests if a file exists and what type it is.

```

#include <glib.h>
static void handle_error (GError*);
int main (int argc, char *argv[]) {
  gchar *filename, *content;
  gsize bytes;
  GError *error = NULL;
  /* Build a filename in the user's home directory. */
  filename = g_build_filename (g_get_home_dir (), "temp", NULL);
  /* Set the contents of the given file and report any errors. */
  g_file_set_contents (filename, "Hello World!", -1, &error);
  handle_error (error);
  /* Get the contents of the given file and report any errors. */
  g_file_get_contents (filename, &content, &bytes, &error);
  handle_error (error);
  g_print ("%s\n", content);
  g_free (content);
  g_free (filename);
  return 0;
}
static void handle_error (GError *error) {
  if (error != NULL) {
    g_printf (error->message);
    g_clear_error (&error);
  }
}

```

`g_file_set_contents()` returns `TRUE` if successful, `FALSE` if not. `G_FILE_ERROR` is a macro for `GError` codes. `g_file_get_contents()` returns `TRUE` if successful, `FALSE` if not. `G_FILE_ERROR` is a macro for `GError` codes. `g_file_test()` returns `TRUE` if the file exists and is a regular file, `FALSE` if not. `G_FILE_TEST_IS_REGULAR`, `G_FILE_TEST_IS_SYMLINK`, `G_FILE_TEST_IS_DIR`, `G_FILE_TEST_IS_EXECUTABLE`, and `G_FILE_TEST_EXISTS` are bitwise ORed together. `G_FILE_TEST_IS_DIR | G_FILE_TEST_IS_REGULAR` returns `TRUE` if the file is a regular file or a directory. `G_FILE_TEST_IS_REGULAR` returns `TRUE` if the file is a regular file. `g_dir_read_name()` returns the next directory entry. `g_dir_rewind()` rewinds the directory iterator. `g_dir_close()` closes the directory iterator. `g_rename()` renames a file. `g_rmdir()` removes a directory. `g_mkdir()` creates a directory.

```

#include <glib.h>
int main (int argc, char *argv[]) {
  /* Open the user's home directory for reading. */
  GDir *dir = g_dir_open (g_get_home_dir (), 0, NULL);
  const gchar *file;
  if (!g_file_test (g_get_home_dir (), G_FILE_TEST_IS_DIR))
    g_error ("Error: You do not have a home directory!");
  while ((file = g_dir_read_name (dir)))
    g_print ("%s\n", file);
  g_dir_close (dir);
  return 0;
}

```

`g_dir_open()` returns a `GDir` object. `g_dir_rewind()` rewinds the directory iterator. `g_dir_read_name()` returns the next directory entry. `g_dir_close()` closes the directory iterator. `g_rename()` renames a file. `g_rmdir()` removes a directory. `g_mkdir()` creates a directory.

```

int g_rename (const gchar *old_filename, const gchar *new_filename);
g_rename() returns 0 if successful, non-zero if not.
g_rmdir() returns 0 if successful, non-zero if not.
g_mkdir() returns 0 if successful, non-zero if not.

```

```

*filename, int permissions);
int g_chdir(const gchar *path);
int g_chmod(const gchar *filename, int permissions);
n int GLib GTK+
gtk_init() GLib GTK+
GTK X GLib sleep
poll() Linux poll() GLib
sleep GTK+
gtk_main() gtk_main_quit() gtk_main_level()
GMainContext g_main_context_get() g_main_context_get_default()
G_PRIORITY_DEFAULT timeout idle GLib
GMainLoop g_main_loop_new() NULL
(default) is_running TRUE g_main_loop_run()
g_main_loop_new(GMainContext *context, gboolean is_running);
gtk_dialog_run() g_main_loop_new()
GLib g_main_loop_quit() GTK+ gtk_main()
GMainLoop GLib GTK+ GLib
GSource GLib g_timeout_source_new() g_idle_source_new()
timeout idle
g_source_new() GSourceFuncs *source_funcs, guint struct_size);
g_source_attach()
GMainContext
GLib API
Timeouts Timeout FALSE g_timeout_add_full() g_timeout_add()
6-7 1/10 0.1 (pulse step) 100
timeout 25 6-7. Timeout (timeouts.c) #include <gtk/gtk.h> static gboolean pulse_progress
(GtkProgressBar*); int main(int argc, char *argv[]) { GtkWidget *window, *progress;
gtk_init(&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(window), "Timeouts");
gtk_container_set_border_width(GTK_CONTAINER(window), 10);
gtk_widget_set_size_request(window, 200, -1);
progress = gtk_progress_bar_new();
gtk_progress_bar_set_pulse_step(GTK_PROGRESS_BAR(progress), 0.1);
g_timeout_add(100, (GSourceFunc) pulse_progress, (gpointer) progress);
gtk_container_add(GTK_CONTAINER(window), progress);
gtk_widget_show_all(window);
gtk_main();
return 0; }
/* Pulse the progress bar and return TRUE so the timeout is called again. */
static gboolean pulse_progress(GtkProgressBar *progress) {
static gint count = 0;
gtk_progress_bar_pulse(progress);
i++;
return (i < 25); }
Timeout g_timeout_add() g_timeout_add_full()
GDestroyNotify
timeout FALSE
guint g_timeout_add_full(gint priority,
guint interval_in_milliseconds,
GSourceFunc timeout_function,
gpointer data,
GDestroyNotify destroy_function);
g_timeout_add_full() timeout G_PRIORITY_DEFAULT
G_PRIORITY_HIGH: GLib GTK+
CPU (CPU-intensive) G_PRIORITY_DEFAULT:
GDKit timeout X idle
G_PRIORITY_HIGH_IDLE: idle (redrawing) GTK+
G_PRIORITY_DEFAULT_IDLE: idle
G_PRIORITY_LOW: GLib GTK+
g_timeout_add_full() 6-7 1/10 100 timeout
Timeout 300 (interval)
Timeout 300
Timeout g_timeout_add_full()
gboolean FALSE gpointer g_timeout_add_full()

```

(destroy) 000 0000 idle 000 000 0, 0 idle 00000 FALSE 000 0 00000 00. 00 00000 NULL 0000 00 0000. g\_timeout\_add\_full() 000 000 00 00 0000 000 gpointer 000 000000 000 0. 000 gpointer 000 timeout 000 00 000 00 000 0000, 00 0 00000 000 000 00000 0000.

Idle 0 0 0 100 000 00 00 GLib 00 000000 00 00 0000 00 0 0 000 00 000 000 idle 000 00000. 00 00 0000 0 00 0 00 00 000 0000 0000. Idle 000 g\_idle\_add() 00 g\_idle\_add\_full() 00 00000. 00 000 000 0 00 00 G\_PRIORITY\_DEFAULT\_IDLE 0000 0000 00 00000 00 0000 00 000 0000 00 00. 00 000 0 00 00000 idle 000 000000. Idle 000 00 000000 00 00 0000 00 000 0000. 000 000000 0000 000 00 000 0000. 00 00 00 idle 0 0 0 G\_PRIORITY\_HIGH\_IDLE 00 G\_PRIORITY\_DEFAULT\_IDLE 00000 00 000. guint g\_idle\_add\_full (gint priority, GSourceFunc idle\_function, gpointer data, GDestroyNotify destroy\_function); g\_idle\_add\_full() 00 0 00 000000 00 idle 0 00. Timeout 000 0000 idle 000 gpointer 0000 gboolean 00 0000. Idle 00000 FALSE 0000 000 000 000. gpointer 000000 g\_idle\_add\_full() 0 00 00000 00 00000. 000 000000 idle 0000 FALSE 0000 000 000 0 0000 00 000 0000. 00 000000 NULL 0000 00 00000. g\_idle\_add\_full() 000 00 000 00 00 0000 0 000 gpointer 0000000 00000. 000 gpointer 000 idle 000 00 000 00 000 0000, 00 0 000000 000 000 00 000 0000. 0000000 FALSE 0000 idle 000 0000 00 000 g\_idle\_remove\_data() 0000 0000000 0000 000 000 0 00. 0 000 idle 000 000 000 0 0 00 0000, 00000 0000 TRUE 000 0000. gboolean g\_idle\_remove\_by\_data (gpointer data); 00 00 0000 idle 000 g\_idle\_remove\_by\_data() 00000 00 0 00. 00 idle 00 0 000 000 000 0 00. 00 0000 idle 000 00000 00 FALSE 0000. 000 00 GLib 0000 00 000 000 0 000 000 000 000 000 0000, 00 0000 00 000 000 000 0 00 0000, 0 0 000 GTK+ 000000 00000. 0 0000 00 0 00 00 0000 000 0000 0000, GTK+ 000 00 0000 0000. 00 00 000 0000 000 000 000 000 000 0000 000 0000 000 0000 0000 000 0000. 00 00 000 000 0 000 0000. 000 000 0000 000 0-0000 000 00 000 0000 000 00 000 0 000 00. 0000000 00000000 00 0000 000 000 0000 GString 0000 C 0000000 0 00 0000 0 00 0 00. 00 0000 0000 0000 000 0000 0000 0000 0000 00 0000 0 00 000 0000. 0 000 C 0000 00 0 00 00 0000000 00 000 00 00 000 0000 0000. Glib 0000 00 000 00, C 0000 00 000 00 0000, 000 000 000 000 00000 00. 000 0 00000 GString 0000 00 000 00 0 0000 C 0000 000 0 00. GString 000 0 300 member 000000, 00 0000 00 000 0000 C 000, 000 0000 000 str 0 00, 00 0000 000 00000 00000. 0000 000 00 0000 000 00 00 GString 0 0 0 0 0000 0 000 000. typedef struct { gchar \*str; gsize len; gsize allocated\_len; } GString; GString 000 str member 00 00 (permanent reference) 0000 0 00. 0000 000 000 0000 000 0000 0000 00, 00 00 00000 00 000 000 00 00! 000 GString 000 0000 0000 0 000 00. g\_string\_new() 0000 00 00000 0 GString 000 0 00. GString 000 initial\_str 000 00000 0000 0000 0 00 00 000 000 0 00. 00 0000 NULL 0000 g\_string\_new() 0000 0 GString 000 0000. GString\* g\_string\_new (const gchar \*initial\_str); GString\* g\_string\_new\_len (const gchar \*initial\_str, gssize length); GString\* g\_string\_sized\_new (gsize default\_size); 0 GString 0000 0 00 000 g\_string\_new\_len() 0000 0000, 00 initial\_str 0 length 000 GString 00000 00 length 0 -1 0000 00 0000 000 00. GString 00 000 000(embedded) 0 0000 000 0 000 0 00 000 00. 00000 0 00 GString 000 000 g\_string\_sized\_new(), default\_size 000 0 0 0000 000 0000. 0 000 0000 0 0000 00000 00 00 00000 000 0 0 00. 0 00 0 0 000 000 g\_string\_printf() 0 0 000, 00 GString 000 0000 00 printf() 0000 000 000 0 000 0000. 000 00000, GString 00 00 0 0000 000 0 00 0000. GString 000 000 0000 000 00 00000. void g\_string\_printf (GString \*string, const gchar \*format, ...); 00 000 0000 00 0 GString 00 0000 0000 0000 g\_string\_append\_printf() 0000 00 00. 00 00000 00 GString 00 0000 00 0 000 0 00 000 00 00. 000 0000 00 val 00 00, val 0 len 00, 0 00, 00 UCS-4 000 00 00000 0000. GString\* g\_string\_append (GString \*string, const gchar \*val); GString\* g\_string\_append\_len (GString \*string, const gchar \*str, gssize len); GString\* g\_string\_append\_c (GString \*string, gchar c); GString\* g\_string\_append\_unichar (GString \*string, gunichar wc); 00 0 00 00 000 GString 0 0 00000 0000 00 000 00 0000 00000. 00 g\_string\_prepend\_c() GString 00 000 000 00000, g\_string\_insert() GString 000 000 00000 000 0000. 000 0000 0 0 000 000 GLib API 0000 "String(000)" 0 000 00 00000 00. 0000 GString 000 0000 000 0000000 0000 0000 000 00 00 00000. g\_string\_erase() 0000 000 0000 000 GString 00000 000 000 000 0 00. 00 000 000 0000 0000 00 00000, 000 000 000 0 00. GString\* g\_string\_erase (GString \*string, gssize pos, gssize len); GString 000 000 g\_string\_free() 000 00000 00000 00. free\_segment TRUE 0000 C 0000 00000 NULL 0000. 00 000 000 C 0000 000000 0000 00 000 00000 00. gchar\* g\_string\_free (GString \*string, gboolean free\_segment); GString 000 000 00 00000 0000 00 0000 GLib 0000 00 000 000 000 00 00 000 0000 00 00 000 0000 00. GString 0000 00 000 00 00 000 000 00 000 0000. 000 C 0000 000000 000000 00 0000 000 0000. 00 00000 00 00 000 0000 GLib 00 0000 00 0 00 0000 0000. GLib 00 00 0000 00 00 0000, 0 00 000 00000. GLib 00 g\_slist\_foo() g\_list\_foo() 0000 000 0 00 000 000 0000. 00 00 000(GSList) 00 000 000 00 0000 0 000 000 000 00 000 00 0000 0000. NULL 0000 0000 000 000 00000. GSList 0000 00 0 0 000 000 0000. typedef struct { gpointer data; GSList \*next; } GSList; 00 00 000(GList) 0000 00 000 0000 00 00 00000 00 0 000 00 00 0000 000 0000. 00 0 0 00 00000 00(traverse) 0 000 0000. NULL 000 000(previous) 0000 000000 0 000 00000. typedef struct { gpointer data; GList \*next; GList \*prev; } GList; 00 00 0000 00000 000 0 00 000 0000 0 00 000 000 00 000 000 00 00. 000 00 00 0000 00 00 000 000 000 0000(prefix) 0000 00 00000 00 00000. 00 000 00 00 000 000 000 GLib 0000 00000. 0 00 0 0000 0000, 00 0 0 000 00 00 000 000 00 000 00000 000 00000. 000 00 000 0 000 00000 g\_list\_prepend() 0000. g\_list\_append() 000 000 00 0000 00 0000

**g\_list\_reverse()** returns the reversed list. **GList\* g\_list\_prepend (GList \*list, gpointer data);** prepends data to the list. **g\_list\_insert()** inserts data at the given position. **g\_list\_append()** appends data to the list. **g\_list\_insert\_before()** inserts data before the given element. **g\_list\_length()** returns the number of elements in the list. **GList\* g\_list\_insert (GList \*list, gpointer data, gint position);** inserts data at the given position. **g\_list\_remove()** removes the first element matching the given data. **g\_list\_remove\_link()** removes the first element matching the given data. **GList\* g\_list\_remove (GList \*list, gconstpointer data);** removes the first element matching the given data. **g\_list\_remove\_all()** removes all elements matching the given data. **g\_list\_remove\_link\_all()** removes all elements matching the given data. **g\_list\_remove\_all()** removes all elements matching the given data. **void g\_list\_free (GList \*list);** frees the list. **g\_list\_sort()** sorts the list using the given compare function. **GCompareFunc** is a function that takes two pointers and returns an integer. **GList\* g\_list\_sort (GList \*list, GCompareFunc compare\_func);** sorts the list using the given compare function. **g\_list\_find()** returns the first element matching the given data. **GList\* g\_list\_find (GList \*list, gconstpointer data);** returns the first element matching the given data. **g\_list\_find\_custom()** returns the first element matching the given data using the given find function. **g\_list\_sort()** sorts the list using the given compare function. **GCompareFunc** is a function that takes two pointers and returns an integer. **g\_tree\_new()** creates a new tree. **GTree\* g\_tree\_new (GCompareFunc key\_compare\_func);** creates a new tree. **g\_tree\_new\_with\_data()** creates a new tree with the given data. **GTree\* g\_tree\_new\_with\_data (GCompareDataFunc key\_compare\_func, gpointer key\_compare\_data);** creates a new tree with the given data. **g\_tree\_new\_full()** creates a new tree with the given data and destroy functions. **GTree\* g\_tree\_new\_full (GCompareDataFunc key\_compare\_func, gpointer key\_compare\_data, GDestroyNotify key\_destroy\_func, GDestroyNotify value\_destroy\_func);** creates a new tree with the given data and destroy functions. **g\_tree\_insert()** inserts a new node into the tree. **void g\_tree\_insert (GTree \*tree, gpointer key, gpointer value);** inserts a new node into the tree. **g\_tree\_replace()** replaces a node in the tree. **g\_tree\_lookup()** returns the value of a node in the tree. **gpointer g\_tree\_lookup (GTree \*tree, gconstpointer key);** returns the value of a node in the tree. **g\_tree\_lookup\_extended()** returns the value of a node in the tree and the next node. **g\_tree\_foreach()** iterates over all nodes in the tree. **GTraverseFunc** is a function that takes a tree, a pointer, and a gpointer. **void g\_tree\_foreach (GTree \*tree, GTraverseFunc func, gpointer data);** iterates over all nodes in the tree. **g\_tree\_search()** searches for a node in the tree. **gpointer g\_tree\_search (GTree \*tree, GCompareFunc search\_func, gconstpointer value);** searches for a node in the tree. **g\_tree\_remove()** removes a node from the tree. **gboolean g\_tree\_remove (GTree \*tree, gconstpointer key);** removes a node from the tree. **g\_tree\_destroy()** destroys the tree.



N-ary GLib n-ary GNode GNode 5 GNode data. GLib typedef struct { gpointer data; GNode \*next; GNode \*prev; GNode \*parent; GNode \*children; } GNode; member (next) (previous) 6-1 GNode 3 6-1 GNode 3 g\_node\_new() GNode NULL g\_node\_append() NULL sibling g\_node\_insert\_before() g\_node\_append\_data() g\_node\_insert() g\_node\_insert\_after() NULL g\_node\_insert\_before() g\_node\_insert\_data() g\_node\_insert\_data\_before() g\_node\_prepend() g\_node\_prepend\_data() 6-5 N-ary n-ary g\_node\_traverse() void g\_node\_traverse (GNode \*root, GTraverseType order, GTraverseFlags flags, gint max\_depth, GNodeTraverseFunc func, gpointer data); G\_IN\_ORDER: G\_PRE\_ORDER: G\_POST\_ORDER: G\_LEVEL\_ORDER: G\_TRAVERSE\_LEAVES: G\_TRAVERSE\_LEAFS G\_TRAVERSE\_NON\_LEAVES: G\_TRAVERSE\_NON\_FLAGS G\_TRAVERSE\_ALL: (G\_TRAVERSE\_LEAVES G\_TRAVERSE\_NON\_LEAVES bitwise mask) G\_TRAVERSE\_MASK: g\_node\_traverse() 3 -1 GNodeTraverseFunc TRUE FALSE n-ary g\_node\_destroy() GArrayGLib GArray GPtrArray GByteArray GArrayGLib API GArray public member, data g\_array\_sized\_new() (reserved\_size) GArray\* g\_array\_sized\_new (gboolean zero\_terminated, gboolean set\_to\_zero, guint element\_size guint reserved\_size); zero\_terminated TRUE 0 element\_size g\_array\_sized\_new() g\_array\_new() GArray set\_to\_zero, element\_size) GArray\* g\_array\_new (gboolean zero\_terminated, gboolean set\_to\_zero, guint element\_size); GArray g\_array\_append\_vals() data len g\_array\_append\_val() g\_array\_append\_vals() GArray\* g\_array\_append\_vals (GArray \*array, gconstpointer data, guint len); g\_array\_append\_val() GArray g\_array\_append\_val()

```

g_array_append_vals() 00 0000 00 0 00 00 00 00000 0000 00 000 0000. #define g_array_append_val(a,v)
g_array_append_vals (a, &(v), 1) g_array_remove_index() 0000 000 000 000 000 0 00. 0000 00 00 000 000 00 000 0 00
000(forward) 0000 000. 000 0000 GArray 000 000 000 0000 00. GArray* g_array_remove_index (GArray *array, guint
index); g_array_remove_index_fast() 0000 000 000 000 000 000 0000 000. 00 000 g_array_remove_index() 00 0000 0000 000 0
0000 00 000. 000 0 000 00 000 0 00 000 000. 000 000 0 00 00000 0000 000 g_array_remove_range() 0000 00. 00 000 index 00 000
length 0000 0000, 0 00 00 0000 0 0000 000000. 00 000 g_array_remove_index() 00 00 000 000 0000 000 000 0 00000 000 00000 00.
GArray* g_array_remove_range (GArray *array, guint index); guint length); GArray 000 0 0000 0000 000 000 0000 00 00.
000 000 0000 0 00 000 00(indexing) 00 000 000000 000, 000 000 0000 0000. g_array_index() 000 000 000 0 000, 00 GArray 00, 0000 000
0 000 000 00, 00 000 00000. 000 00 0000 0 00 00000 00 0 000 0000 00 0000 000. GLib 00 000 000 000000 g_array_sort() 000 GArray 0
000 0 00. 00 000 0 00 000 00000 0 0000 00 GCompareFunc 000 0000. 000000 000 000 00000 00 000 0000 000 g_array_sort_data() 000 0 0
0. void g_array_sort (GArray *array, GCompareFunc compare_func); GArray 000 000 000 g_array_free() 000 000000 00.
00 000 0000 00000 000 00 0000 000 00 000 00 00 000 0000 00 000 0000 00. gchar* g_array_free (GArray *array, gboolean
free_segment); free_segment 0 true 0 0 0 00 00 0000 0000 000 NULL 000 000. 0 00 000 0000 000 00 00 000 000 000. 00 00 GArray
000 000 000 000 0000 00 000 0 00. 000 00 GPtrArray API 0 GArray 000000 0000 0000 000 00000 000 000. 0, 00 000 0 000 00 000 000 00000
0 0000 00 000 000 0000 000 000 0000. GPtrArray 0000 0000 00 000 00 000 000 0000. typedef struct { gpointer *pdata; guint len; }
GPtrArray; 000 GPtrArray 0000 0 00 000 000 g_ptr_array_add() 0000. 0 000 00 000 0000 00 0000. 000 0000 000 00 000000 0000 0
00 00, g_ptr_array_remove() 0 g_ptr_array_remove_fast() 0000. 000 0000 0000 00 0 0000 000 0000 0000 000 0000. 000 00000 0000
TRUE 0000 000 0000. gboolean g_ptr_array_remove (GPtrArray *array, gpointer data); GPtrArray 000
g_ptr_array_foreach() 000 000000, 00 00 0 00 000 0 foreach_fun() 0 000 000. 00 000 00 000 000 0000 g_ptr_array_foreach() 000 0
00 00000 0000. void g_ptr_array_foreach (GPtrArray *array, GFunc foreach_func, gpointer data); 000 000 000 000
g_ptr_array_free() 000 0000. 0 000 00 00 000 000 00 0 000 0000 00 0000 00000 000. 000 00000 000 guint8 0000 GArray 00000. 000 000 0
0 00000 00 00000 000 0000. GArray 000 00 GByteArray 0 000 000 00 000 0000 000 000 000 00 00000 00 000 0 00 000 0000. typedef struct
{ guint8 *data; guint len; } GByteArray; 00 00, 00 0 000 000 000 00 00000 000 000 000 00 0000 GByteArray 0 GArray 000 0000. 0
0 GByteArray 000 00000 GArray 000 0000. 000 000 00 0 000 g_byte_array_free() 000 0000 00. 00 000 000 00000 000 0000 00000 000
0 0 00 00 000000 00000 00 00. 00000 000 00 g_byte_array_free() 000 NULL 000 000. 00 00000 0000 000 00 000 000 0 00 0000 000 0000.
0000 000 0-0 000 0000. 000 GLib 00 00 00 00 0 GHashTable 00 0000 00 000 000 00 00 0000 00 00 0000 00. 0, GTK+ 000000 000 0
0 00 000 0000 0000 0 000 000. 00 000 0 000 00 0000 000 00(copy) 000 00 g_strdup() 0000 00. 00 0000 000000 000 00(lookup) 000 000
0 000 000 000 00 0 0000. 000 0 0 0 0 0 000 00 0 0000. 00 000 00 00 000 00 000 0 000 00 000. GLib 000 000 0000 g_hash_table_new() 0
00 000000, 00 0 00 000 0000. 00 00 0 0 0 0 000 00 0 0000 0 000000, NULL 00 0000. 0 00 000 0 00 00 0000 0000 0 0000. GHashTable*
g_hash_table_new (GHashFunc hash_func, GEqualFunc key_equal_func); 00 000 00 000 GHashFunc 00 00 000000 00 000
0. 00 0 00 0000 00 0000 00 00 0000. 0000 0000 00 000 0 000 000 GLib 0000 00 00 0000 00 0 00 0000 00. 000 g_direct_hash(),
g_int_hash(), g_str_hash(), 00 000 gpointer, gint, 0000 0 000 0 000. guint (*GHashFunc) (gconstpointer key); 0 00 000 00
GEqualFunc 000000 00 0000. ad b 0 000 00 000 TRUE 0, 0000 000 FALSE 0000. GLib 000 gpointer, gint, 000 000 00 00 00 000
000 g_direct_equal(), g_int_equal(), g_str_equal() 000 0000. gboolean (*GEqualFunc) (gconstpointer a, gconstpointer
b); GLib 0 g_hash_table_new() 000 00 00000 00 00 000 0 00 00 00 00(destroy callback) 000 00000 000 g_hash_table_new_full()
00 0000. 000 0-0 00 00 0000 0000 0000 0 000 00. 0 000 g_hash_table_insert() 0000 0000. 000 00 00 00 00000 value 00 00 000 000. 0 0
0 000 g_hash_table_replace() 0000 0000, 00 00 000 000 0000. 000 000 00 00 000 00 0 000 0 000 0 000 000 000. void
g_hash_table_insert (GHashTable *hash_table, gpointer key, gpointer value); 00 00000 0-0 00 000 00
g_hash_table_remove() 0000. 00 00 00 00 000 00000 0 0 000 000. 0000 000 0 00 0000 0000 000 0 000 0000 0000 00 00000 0000 00. 00 0000
0 0000 00 000 TRUE 0 000 000. g_hash_table_remove_all() 000 00 00000 0 0 0-0 00 000 00 00. gboolean
g_hash_table_remove (GHashTable *hash_table, gconstpointer key); 00 00000 00 0000 00 0 0 0 0 000 000 0000 00 00 000
00000 00 00. g_hash_table_lookup() 00 0 00 00 0000 00 000 0 00. 000 00 000000, 00 00 0 000 NULL 0 000 0 0 0. gpointer
g_hash_table_lookup (GHashTable *hash_table, gconstpointer key); 0 000 g_hash_table_lookup_extended() 00 000 000
00 00. GHashTable 000 0000 TRUE 0 000. 00 00 00 00 000 000, 000 000 0000 0 00 00 00000 0000. 00 0000 000 000
g_hash_table_destroy() 000 000000 00. 00 00 00 00 000 00 0 0 0 0 0 0000 000 000. void g_hash_table_destroy (GHashTable
*hash_table); 0 000 000 000 00 00 0000 00 000 00 000 000 00 00 0 0000. 00 0000 00 000 g_hash_table_ref()
g_hash_table_unref() 0 000 00 0 0000 0 00. 000 00 000 0 0000 00 00 0000 000 0000 000 0 00. 0000(Quark) 0 32 00 000 000 00 000 0

```

g\_quark\_from\_string() returns a GQuark for a given string. GQuark is a 32-bit integer, and g\_quark\_from\_string() returns a unique GQuark for each string.

```

GQuark g_quark_from_string(const gchar *string);
GQuark g_quark_from_string_static(const gchar *string);
GQuark g_quark_try_string(const gchar *string);
const gchar* g_quark_to_string(GQuark quark);
void g_datalist_init(GData **datalist);
void g_datalist_id_set_data_full(GData **datalist, GQuark key_id, gpointer data, GDestroyNotify destroy_func);
void g_datalist_id_remove_no_notify(GData **datalist, GQuark key_id);
void g_datalist_foreach(GData **datalist, GDataForeachFunc func, gpointer user_data);
void g_datalist_clear(GData **datalist);
GIOChannel* g_io_channel_new_file(const gchar *filename, gchar mode, GError **error);
void g_io_channel_write_chars(GIOChannel *channel, const gchar *text, gssize size_of_buffer, gsize *bytes_written, GError **error);
void g_io_channel_shutdown(GIOChannel *channel, gboolean flushed);

```

The g\_io\_channel\_new\_file() function creates a new GIOChannel for a file. The mode parameter is a character string that specifies the file access mode. The modes are:

- w: write only
- a: append only
- r: read only
- w+: write and append
- a+: append and write
- r+: read and write

The g\_io\_channel\_write\_chars() function writes a string of text to the channel. The size\_of\_buffer parameter is the number of bytes to write, and bytes\_written is a pointer to a gsize variable that will contain the number of bytes actually written.

The g\_io\_channel\_shutdown() function shuts down the channel. The flushed parameter is a gboolean that indicates whether the channel should be flushed before shutting down.

```

g_io_channel_shutdown (GIOChannel *channel, gboolean flush, GError **error); GIOChannel
    g_io_channel_read_to_end() GIOChannel
GIOChannelError GConvertError GIOStatus g_io_channel_read_to_end (GIOChannel
    *channel, gchar **text, gsize *length, GError **error); GIOChannels
    UNIX watch
    UNIX ID(pid)
    fork()
    forking
    UNIX forked
    switch (fork()) { case -1: g_error ("Error: The fork() failed!"); exit (1); case 0:
    g_message (We are currently running the child process!"); exit (0); default: gint status_of_child; wait
    (&status_of_child); }
    pipe()
    forking
    UNIX GLib
    6-9 GIOChannel
    UNIX watch
    GLib
    Watch
    6-9 GtkEntry
    #include <gtk/gtk.h> #include <stdlib.h> #include <stdio.h> #include <errno.h> #include <unistd.h> #include <string.h>
    static void entry_changed (GtkEditable*, GIOChannel*); static void setup_app (gint input[], gint output[], gint pid);
    static gboolean iochannel_read (GIOChannel*, GIOCondition, GtkEntry*); gulong signal_id = 0; int main (int argc,
    char* argv[]) { gint child_to_parent[2], parent_to_child[2], pid, ret_value; /* Set up read and write pipes for the
    child and parent processes. */ ret_value = pipe (parent_to_child); if (ret_value == -1) { g_error ("Error: %s\n",
    g_strerror (errno)); exit (1); } ret_value = pipe (child_to_parent); if (ret_value == -1) { g_error ("Error: %s\n",
    g_strerror (errno)); exit (1); } /* Fork the application, setting up both instances accordingly. */ pid = fork (); switch
    (pid) { case -1: g_error ("Error: %s\n", g_strerror (errno)); exit (1); case 0: gtk_init (&argc, &argv); setup_app
    (parent_to_child, child_to_parent, pid); break; default: gtk_init (&argc, &argv); setup_app (child_to_parent,
    parent_to_child, pid); } gtk_main (); return 0; } /* Set up the GUI aspects of each window and setup IO channel
    watches. */ static void setup_app (gint input[], gint output[], gint pid) { GtkWidget *window, *entry; GIOChannel
    *channel_read, *channel_write; window = gtk_window_new (GTK_WINDOW_TOPLEVEL); entry =
    gtk_entry_new (); gtk_container_add (GTK_CONTAINER (window), entry); gtk_container_set_border_width
    (GTK_CONTAINER (window), 10); gtk_widget_set_size_request (window, 200, -1); gtk_widget_show_all
    (window); /* Close the unnecessary pipes for the given process. */ close (input[1]); close (output[0]); /* Create read
    and write channels out of the remaining pipes. */ channel_read = g_io_channel_unix_new (input[0]); channel_write
    = g_io_channel_unix_new (output[1]); if (channel_read == NULL || channel_write == NULL) g_error ("Error: The
    GIOChannels could not be created!\n"); /* Watch the read channel for changes. This will send the appropriate data.
    */ if (!g_io_add_watch (channel_read, G_IO_IN | G_IO_HUP, iochannel_read, (gpointer) entry)) g_error ("Error:
    Read watch could not be added to the GIOChannel!\n"); signal_id = g_signal_connect (G_OBJECT (entry), "
    changed", G_CALLBACK (entry_changed), (gpointer) channel_write); /* Set the window title depending on the
    process identifier. */ if (pid == 0) gtk_window_set_title (GTK_WINDOW (window), "Child Process"); else
    gtk_window_set_title (GTK_WINDOW (window), "Parent Process"); } /* Read the message from the pipe and set
    the text to the GtkEntry. */ static gboolean iochannel_read (GIOChannel *channel, GIOCondition condition,
    GtkEntry *entry) { GIOStatus ret_value; gchar *message; gsize length; /* The pipe has died unexpectedly, so exit
    the application. */ if (condition & G_IO_HUP) g_error ("Error: The pipe has died!\n"); /* Read the data that has
    been sent through the pipe. */ ret_value = g_io_channel_read_line (channel, &message, &length, NULL, NULL); if

```

```

(ret_value == G_IO_STATUS_ERROR) g_error ("Error: The line could not be read!\n"); /* Synchronize the
GtkEntry text, blocking the changed signal. Otherwise, an * infinite loop of communication would ensue. */
g_signal_handler_block ((gpointer) entry, signal_id); message[length-1] = 0; gtk_entry_set_text (entry, message);
g_signal_handler_unblock ((gpointer) entry, signal_id); return TRUE; } /* Write the new contents of the GtkEntry
to the write IO channel. */ static void entry_changed (GtkEditable *entry, GIOChannel *channel) { gchar *text;
gsize length; GIOStatus ret_value; text = g_strconcat (gtk_entry_get_text (GTK_ENTRY (entry)), "\n", NULL); /*
Write the text to the channel so that the other process will get it. */ ret_value = g_io_channel_write_chars (channel,
text, -1, &length, NULL); if (ret_value = G_IO_STATUS_ERROR) g_error ("Error: The changes could not be
written to the pipe!\n"); else g_io_channel_flush (channel, NULL); } IO
pipe()
6-9
GIOChannels
fork()
fork()
setup_app()
close()
GIOChannel
channel_read,
GtkEntry
channel_write
channel_read = g_io_channel_unix_new (input[0]);
channel_write = g_io_channel_unix_new (output[1]);
IO
channel_read
watch
g_io_add_watch()
guint g_io_add_watch (GIOChannel *channel, GIOCondition condition,
GIOFunc func, gpointer data);
g_io_add_watch()
GIOFunc
G_IO_IN
GIOCondition
g_io_add_watch()
pipe
G_IO_IN:
G_IO_OUT:
G_IO_PRI:
GIOFunc
G_IO_ERR:
G_IO_HUP:
G_IO_NVAL:
GIOFunc
TRUE
gboolean (*GIOFunc) (GIOChannel
*source, GIOCondition condition, gpointer data);
GIOChannel
g_io_channel_read_*()
g_io_channel_write_*()
GIOChannel
GError
GIOStatus
G_IO_STATUS_ERROR:
G_IO_STATUS_NORMAL:
G_IO_STATUS_EOF:
G_IO_STATUS_AGAIN:
GIOStatus
G_IO_STATUS_AGAIN
poll()
g_io_channel_flush()
GIOChannel
flush
GIOChannelError
pipe()
fork()
Microsoft Windows
GLib
g_spawn_async_with_pipes()
gboolean
g_spawn_async_with_pipes (const gchar *working_directory, gchar **argv, gchar **envp, GSpawnFlags flags,
GSpawnChildSetupFunc child_setup, gpointer data, GPid *child_pid, gint *standard_input, gint *standard_output,
gint *standard_error, GError **error);
G_SPAWN_SEARCH_PATH
envp
KEY=VALUE
G_SPAWN_LEAVE_DESCRIPTOR_OPEN:
open file
G_SPAWN_DO_NOT_REAP_CHILD:
reap
waitpid()
SIGCHLD
zombie
G_SPAWN_SEARCH_PATH:
argv[0]
G_SPAWN_STDOUT_TO_DEV_NULL:
G_SPAWN_STDERR_TO_DEV_NULL:
G_SPAWN_CHILD_INHERITS_STDIN:
/dev/null
G_SPAWN_FILE_AND_ARGV_ZERO:
argv[0]
g_spawn_async_with_pipes()
GSpawnChildSetupFunc
GLib
exec()
g_spawn_async_with_pipes()
data

```

```

NULL return TRUE. Return FALSE if g_spawn_async_with_pipes() returned TRUE. If the GSpawnError is NULL, return
FALSE. GPid returns the pid of the child process. On Microsoft Windows, the pid is the process ID. void
g_spawn_close_pid (GPid pid); This function closes the pipe to the child process. If the child process is still
running, this function returns FALSE. If the child process is no longer running, this function returns TRUE.
gcc -shared modules-plugin.c -o plugin.so `pkg-config --libs glib-2.0` \ `pkg-config --cflags glib-2.0` sudo mv plugin.so /usr/lib
GCC build system options for modules-plugin.c: gcc -shared modules-plugin.c -o plugin.so `pkg-config --libs
glib-2.0` \ `pkg-config --cflags glib-2.0` sudo mv plugin.so /usr/lib GNU ld options: -shared modules-plugin.c
GModule options: #include <glib.h> #include <gmodule.h> G_MODULE_EXPORT
gboolean print_the_message (gpointer data) { g_printf ("%s\n", (gchar*) data); return TRUE; }
G_MODULE_EXPORT gboolean print_another_one (gpointer data) { g_printf ("%s\n", (gchar*) data); return
TRUE; }
main() { GModule *module; PrintMessageFunc print_the_message; PrintAnotherFunc print_another_one; gchar
*symbol; module = g_module_open ("/usr/lib/plugin.so", G_MODULE_BIND_LAZY); if (!module) { g_error ("Error: %s\n",
(gchar*) g_module_error ()); return -1; } Load the print_the_message() function. */ if (!g_module_symbol (module, "
print_the_message", (gpointer*) &print_the_message)) { g_error ("Error: %s\n", (gchar*) g_module_error ()); return -1; }
Load the destroy_the_evidence() function. */ if (!g_module_symbol (module, "print_another_one", (gpointer*) &print_another_one)) { g_error ("Error: %s\n",
(gchar*) g_module_error ()); return -1; } Run both loaded functions since there were no errors reported loading *
neither the module nor the symbols. */ print_the_message ((gpointer) text); print_another_one ("Another
Message!"); /* Close the module and free allocated resources. */ if (!g_module_close (module)) g_error ("Error:
%s\n", (gchar*) g_module_error ()); return 0; }
GModule* g_module_supported() returns TRUE if the library is supported. Returns NULL otherwise.
G_MODULE_SUFFIX: The suffix of the module filename.
GModule* g_module_open (const gchar *library, GModuleFlags flags); g_module_open() opens the library and
resolves the symbols. GModuleFlags flags: G_MODULE_BIND_LAZY: The library is not resolved until the
g_module_symbol() function is called. G_MODULE_BIND_LOCAL: The library is resolved only when the
g_module_symbol() function is called. G_MODULE_BIND_MASK: A mask of the G_MODULE_BIND_LAZY and G_MODULE_BIND_LOCAL
options. g_module_error() returns the error message.
gboolean g_module_symbol (GModule *module, const gchar *symbol_name, gpointer *symbol);
g_module_symbol() returns TRUE if the symbol is found in the module. Returns FALSE otherwise.
GModule* g_module_close() closes the module. Returns TRUE if the module was closed successfully. Returns FALSE
otherwise. g_module_make_resident() makes the module resident in memory. Returns TRUE if the module was
made resident successfully. Returns FALSE otherwise.
gchar * g_module_text (GModule *module); Returns the text of the module. Returns NULL if the module is not
supported. timeout: The timeout in milliseconds for the module to be loaded. Returns 0 if the module is not supported.
timeout: 6-1. The timeout in milliseconds for the module to be loaded. Returns 0 if the module is not supported.

```

0 0. 00 **GtkFileChooserButton** 000 0 00 0 0 00000 0 00. 0 00 000 0 000 0000 00. 000 0000 00 00 0 **GtkFileChooserButton** 000 000 000 **GtkEntry**000 00 00000 00000. 0 0000 0 0 00 0 00 000 0 0 00000 00. 0000 6-10 00 0000. 00 0000 **GTK+** 0 0000000 0000 00. 00 0 00 0 00, 00 000 00, **Save** 000 **GtkVBox** 00 00 0 000 0. 000000 000 00 **F** 00 00. 000 0000 0000 0000 0000 0000 0000 00. 0 0 0 00 000 000 00 0 0 000 00000 00. 000 000 000000 0000000 0000 00 **GError** 0000 000 0 00. 0000 6-2. **Timeout** 00 00 000000 **GtkLabel** 00 0 0 000 00 00. 00 0 000 "0"00 000 00000. **timeout** 000 00000 0 00 0 000 0 00 000 0000 0. 000 000 0000 0000 00 000 00000. 00 000 00 00 0000 00 00 00 0 0 0 00 00 **timeout** 000000 0 00. 000 0000 000 00 000 00 0 00 00. 0 00 000 00 000 00 000 000 **timeout** 000 0000 000 000 000 0 00000 000000. 00 0 0 00 0 00 0 0000? 0000 6-20 6-1000 00 0 0000, **GtkLabel** 00 000 **timeout** 000 00 000 000 000 00. 00 00 000 000 00 000 0000 00 0000 00 0000 0 0 0 00. 00 **F** 00 000 00000, 0 00 000 **timeout** 000 00 000 0 00 0000 000000. 00 000 00000 0 0 000 0 000 000 000 0000 0000000000 0 000 0000 00. 00 0 000 0 0 0 **timeout** 000 0000000 000 000 00 000 0 000. **Clear** 000 000 000 0 000 000 **0**00 00000 00. 0 00 0000 00 000 0000 0000 0 000000. 00 00 00 000 00 00 0 000 000 0000. 00 000 0000 **GTK+** 00 000 00 000 00 000 0000 00 0 0 000. 0000 000 00 0 00 000 00 00 0000 000! 00 0000 **GLib** 00 00 00 000 0 00 0 000 000 000000. 00 0 000 000 00 0000 000, 00 00 000 0000 00 000 000 0 0000 00. 000 000 0000 0000000000 0000 0 **API** 0000 00 0 00 00000 0 0. 00 0 0 0 0 00000 000 00, 0 00, 000 00, 00 00, 000 00, 00 00, 0000 00, 0 0 0000 0 00 **GLib** 000 000 0000000 000 000000. **GLib**000 0000 0000 0 00 0000 0. **malloc()** **friends** 0 000 00 **GSlice** 0000 00 0000 0000 000 0 0000. 0 **GLib** 0000000 000 000 000 0000000000 00 0 00000. 00 0 00 00 0 00 000 000 0 0 000 0 0 000. 00 0 00 000 **GLib**00 **GMainLoop**, **GMainContext**, **GSource** 00 00000 000000. 00 000 0 00 00 000 0 **timeout** 0 00 **idle** 0 00 00. **Timeout** 0 00 000 000 00 0000 00000, **idle** 00 0 00000 0 00 00 00000 0 000 00 00 0000. **GLib** 00000 0 00 000 000 00000. 000000 000 00 0 0 0 0 00 000 0000 0. 0000 0000 0000 0000 000 00 000 0000. 00 **C++** 00 000 0000000 0000 000 000 0000 00000. 00 00 0 0 0 00 0000 0 0000 000 0 000 00, 00, 000 0 0 000. **GLib**000 00 00 0000 00 00 0000 0 000. 00 00 000 000 000 0000 00 0000. **N-ary** 000 0 000 000 0 000 000 **branch** 000 0 00. 00 0 0 000 0 0 00. 000 000 0000 00 00 0000 0000. 00 00 000 0000 000 0000 0 0000 0000 0 0 000 000 0000. 00 0000 00 0 000 000000, 000 0000 0 00 00 0 0 0000 00 000 00 000 00. 0000000 0000 00 000 00 0 0 0. **GLib** 00 0 0000 0000 00 00 0000. 000 000 00, 0 0000 00 0 0, **UNIX** 00 000 00 000 0 0 0000. **GIOChannel** 000 00000 00000 00000 00 0000 0 000 0 000 0000. 00000 00 00 0000 000 **GLib** **GModule** 00 0 0 0000 0000. 00 0000 0 0 00 0 0 00 00 0 000 000000 000 000 0 000 000. 00000000 0 0 00 0(modular)00 00 0 00000 00. 000000 00 000 00 **GTK+** 00 0 **GLib** 000 00 00 0 0 0 00. 0 0 0 000 0 0 00 000 0 0 000 0000 000 000. 0 700 **GtkTextView** 00 000 000 000 000 000 00 000 000. 0 000 000000 **GtkSourceView** 00 0000 00 0 00 000. **Notes**







```

gtk_vbox_new (TRUE, 5); gtk_box_pack_start_defaults (GTK_BOX (vbox), viewport);
gtk_box_pack_start_defaults (GTK_BOX (vbox), swin); gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window); gtk_main(); return 0; }
gtk_scrolled_window_new()

```

GTK\_POLICY\_ALWAYS: disabled()

GTK\_POLICY\_AUTOMATIC: GTK\_POLICY\_NEVER:

```

gtk_scrolled_window_set_placement()
void gtk_scrolled_window_set_placement (GtkScrolledWindow *swin, GtkCornerType window_placement);

```

GTK\_CORNER\_TOP\_LEFT, GTK\_CORNER\_BOTTOM\_LEFT, GRK\_CORNER\_TOP\_RIGHT, GTK\_CORNER\_BOTTOM\_RIGHT

```

gtk_scrolled_window_set_shadow_type()
void

```

```

gtk_scrolled_window_set_placement (GtkScrolledWindow *swin, GtkCornerType window_placement);
GtkShadowType

```

```

gtk_scrolled_window_add_with_viewport()
GtkTextView, GtkTreeView, GtkIconView, GtkViewport,
GtkLayout

```

```

gtk_scrolled_window_add_with_viewport()

```

```

GtkScrolledWindow
value-changed

```

```

GtkViewport
GtkTextView

```

```

GtkTextView
Gaim

```

```

GtkTextBuffer
UTF-8

```

```

UTF-8

```

```

GtkTextView
textview.c

```

```

#include <gtk/gtk.h> int main (int argc, char *argv[]) { GtkWidget *window, *scrolled_win, *textview;
GtkTextBuffer *buffer; gtk_init (&argc, &argv); window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "Text Views"); gtk_container_set_border_width

```

```
(GTK_CONTAINER (window), 10); gtk_widget_set_size_request (window, 250, 150); textview =
gtk_text_view_new (); buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview));
gtk_text_buffer_set_text (buffer, "Your 1st GtkTextView widget!", -1); scrolled_win = gtk_scrolled_window_new
(NULL, NULL); gtk_container_add (GTK_CONTAINER (scrolled_win), textview); gtk_container_add
(GTK_CONTAINER (window), scrolled_win); gtk_widget_show_all (window); gtk_main(); return 0; }
```

GtkTextView 的 gtk\_text\_view\_new() 函数返回一个 GtkWidget 对象。该对象包含一个 GtkTextBuffer 对象。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取该对象。您可以通过 gtk\_text\_view\_set\_buffer() 函数设置该对象。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_text\_view\_set\_wrap\_mode() 函数设置换行模式。您可以通过 gtk\_text\_view\_set\_justification() 函数设置文本对齐方式。您可以通过 gtk\_text\_view\_set\_editable() 函数设置是否可编辑。您可以通过 gtk\_text\_view\_set\_cursor\_visible() 函数设置是否显示光标。您可以通过 gtk\_text\_view\_set\_pixels\_above\_lines() 函数设置行上方像素。您可以通过 gtk\_text\_view\_set\_pixels\_below\_lines() 函数设置行下方像素。您可以通过 gtk\_text\_view\_set\_pixels\_inside\_wrap() 函数设置换行内部像素。您可以通过 gtk\_text\_view\_set\_left\_margin() 函数设置左边距。您可以通过 gtk\_text\_view\_set\_right\_margin() 函数设置右边距。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取文本缓冲区。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_scrolled\_window\_set\_policy() 函数设置滚动策略。您可以通过 gtk\_container\_add() 函数将 widget 添加到容器。您可以通过 gtk\_widget\_show\_all() 函数显示所有 widget。您可以通过 gtk\_main() 函数运行 GTK+ 主循环。您可以通过 return 0; 退出程序。

```
(textview2.c) #include <gtk/gtk.h> int main (int argc, char *argv[]) { GtkWidget *window, *scrolled_win,
*textview; GtkTextBuffer *buffer; PangoFontDescription *font; gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "Text
Views Properties"); gtk_container_set_border_width (GTK_CONTAINER (window), 10);
gtk_widget_set_size_request (window, 250, 150); font = pango_font_description_from_string ("Monospace Bold
10"); textview = gtk_text_view_new (); gtk_widget_modify_font (textview, font); gtk_text_view_set_wrap_mode
(GTK_TEXT_VIEW (textview), GTK_WRAP_WORD); gtk_text_view_set_justification (GTK_TEXT_VIEW
(textview), GTK_JUSTIFY_RIGHT); gtk_text_view_set_editable (GTK_TEXT_VIEW (textview), TRUE);
gtk_text_view_set_cursor_visible (GTK_TEXT_VIEW (textview), TRUE); gtk_text_view_set_pixels_above_lines
(GTK_TEXT_VIEW (textview), 5); gtk_text_view_set_pixels_below_lines (GTK_TEXT_VIEW (textview), 5);
gtk_text_view_set_pixels_inside_wrap (GTK_TEXT_VIEW (textview), 5); gtk_text_view_set_left_margin
(GTK_TEXT_VIEW (textview), 10); gtk_text_view_set_right_margin (GTK_TEXT_VIEW (textview), 10); buffer
= gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)); gtk_text_buffer_set_text (buffer, "This is some
text!\nChange me!\nPlease!", -1); scrolled_win = gtk_scrolled_window_new (NULL, NULL);
gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_win), GTK_POLICY_AUTOMATIC,
GTK_POLICY_ALWAYS); gtk_container_add (GTK_CONTAINER (scrolled_win), textview); gtk_container_add
(GTK_CONTAINER (window), scrolled_win); gtk_widget_show_all (window); gtk_main(); return 0; }
```

GtkTextView 的 gtk\_text\_view\_new() 函数返回一个 GtkWidget 对象。该对象包含一个 GtkTextBuffer 对象。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取该对象。您可以通过 gtk\_text\_view\_set\_buffer() 函数设置该对象。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_text\_view\_set\_wrap\_mode() 函数设置换行模式。您可以通过 gtk\_text\_view\_set\_justification() 函数设置文本对齐方式。您可以通过 gtk\_text\_view\_set\_editable() 函数设置是否可编辑。您可以通过 gtk\_text\_view\_set\_cursor\_visible() 函数设置是否显示光标。您可以通过 gtk\_text\_view\_set\_pixels\_above\_lines() 函数设置行上方像素。您可以通过 gtk\_text\_view\_set\_pixels\_below\_lines() 函数设置行下方像素。您可以通过 gtk\_text\_view\_set\_pixels\_inside\_wrap() 函数设置换行内部像素。您可以通过 gtk\_text\_view\_set\_left\_margin() 函数设置左边距。您可以通过 gtk\_text\_view\_set\_right\_margin() 函数设置右边距。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取文本缓冲区。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_scrolled\_window\_set\_policy() 函数设置滚动策略。您可以通过 gtk\_container\_add() 函数将 widget 添加到容器。您可以通过 gtk\_widget\_show\_all() 函数显示所有 widget。您可以通过 gtk\_main() 函数运行 GTK+ 主循环。您可以通过 return 0; 退出程序。

7-3 图显示了运行后的结果。您可以通过 gtk\_text\_view\_set\_wrap\_mode() 函数设置换行模式。您可以通过 gtk\_text\_view\_set\_justification() 函数设置文本对齐方式。您可以通过 gtk\_text\_view\_set\_editable() 函数设置是否可编辑。您可以通过 gtk\_text\_view\_set\_cursor\_visible() 函数设置是否显示光标。您可以通过 gtk\_text\_view\_set\_pixels\_above\_lines() 函数设置行上方像素。您可以通过 gtk\_text\_view\_set\_pixels\_below\_lines() 函数设置行下方像素。您可以通过 gtk\_text\_view\_set\_pixels\_inside\_wrap() 函数设置换行内部像素。您可以通过 gtk\_text\_view\_set\_left\_margin() 函数设置左边距。您可以通过 gtk\_text\_view\_set\_right\_margin() 函数设置右边距。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取文本缓冲区。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_scrolled\_window\_set\_policy() 函数设置滚动策略。您可以通过 gtk\_container\_add() 函数将 widget 添加到容器。您可以通过 gtk\_widget\_show\_all() 函数显示所有 widget。您可以通过 gtk\_main() 函数运行 GTK+ 主循环。您可以通过 return 0; 退出程序。

7-3 图显示了运行后的结果。您可以通过 gtk\_text\_view\_set\_wrap\_mode() 函数设置换行模式。您可以通过 gtk\_text\_view\_set\_justification() 函数设置文本对齐方式。您可以通过 gtk\_text\_view\_set\_editable() 函数设置是否可编辑。您可以通过 gtk\_text\_view\_set\_cursor\_visible() 函数设置是否显示光标。您可以通过 gtk\_text\_view\_set\_pixels\_above\_lines() 函数设置行上方像素。您可以通过 gtk\_text\_view\_set\_pixels\_below\_lines() 函数设置行下方像素。您可以通过 gtk\_text\_view\_set\_pixels\_inside\_wrap() 函数设置换行内部像素。您可以通过 gtk\_text\_view\_set\_left\_margin() 函数设置左边距。您可以通过 gtk\_text\_view\_set\_right\_margin() 函数设置右边距。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取文本缓冲区。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_scrolled\_window\_set\_policy() 函数设置滚动策略。您可以通过 gtk\_container\_add() 函数将 widget 添加到容器。您可以通过 gtk\_widget\_show\_all() 函数显示所有 widget。您可以通过 gtk\_main() 函数运行 GTK+ 主循环。您可以通过 return 0; 退出程序。

7-3 图显示了运行后的结果。您可以通过 gtk\_text\_view\_set\_wrap\_mode() 函数设置换行模式。您可以通过 gtk\_text\_view\_set\_justification() 函数设置文本对齐方式。您可以通过 gtk\_text\_view\_set\_editable() 函数设置是否可编辑。您可以通过 gtk\_text\_view\_set\_cursor\_visible() 函数设置是否显示光标。您可以通过 gtk\_text\_view\_set\_pixels\_above\_lines() 函数设置行上方像素。您可以通过 gtk\_text\_view\_set\_pixels\_below\_lines() 函数设置行下方像素。您可以通过 gtk\_text\_view\_set\_pixels\_inside\_wrap() 函数设置换行内部像素。您可以通过 gtk\_text\_view\_set\_left\_margin() 函数设置左边距。您可以通过 gtk\_text\_view\_set\_right\_margin() 函数设置右边距。您可以通过 gtk\_text\_view\_get\_buffer() 函数获取文本缓冲区。您可以通过 gtk\_text\_buffer\_set\_text() 函数设置文本。您可以通过 gtk\_scrolled\_window\_set\_policy() 函数设置滚动策略。您可以通过 gtk\_container\_add() 函数将 widget 添加到容器。您可以通过 gtk\_widget\_show\_all() 函数显示所有 widget。您可以通过 gtk\_main() 函数运行 GTK+ 主循环。您可以通过 return 0; 退出程序。

```

gtk_text_view_set_justification()
GTK_JUSTIFY_LEFT
void gtk_text_view_set_justification (GtkTextView
*textview, GtkJustification justification);
gtk_text_view_set_left_margin()
gtk_text_view_set_right_margin()
Pango
PangoTabArray
make_tab_array()
static void make_tab_array
(PangoFontDescription *fd, gsize tab_size, GtkWidget *textview) {
PangoTabArray *tab_array; PangoLayout
*layout; gchar *tab_string; gint width, height; g_return_if_fail (tab_size < 100);
tab_string = g_strnfill (tab_size, ' ');
layout = gtk_widget_create_pango_layout (textview, tab_string);
pango_layout_set_font_description (layout, fd);
pango_layout_get_pixel_size (layout, &width, &height);
tab_array = pango_tab_array_new (1, TRUE);
pango_tab_array_set_tab (tab_array, 0, PANGO_TAB_LEFT, width);
gtk_text_view_set_tabs (GTK_TEXT_VIEW
(textview), tab_array);
g_free (tab_string);
}
PangoLayout
Pango
GtkTextVew
PangoLayout
gtk_widget_create_pango_layout()
PangoLayout*
gtk_widget_create_pango_layout (GtkWidget *textview, const gchar *text);
pango_layout_set_font_description()
Pango
PangoFontDescription
void pango_layout_set_font_description (PangoLayout *layout,
const PangoFontDescription *fd);
pango_layout_get_pixel_size()
PangoLayout
pango_layout_get_pixel_size (PangoLayout *layout, int *width,
int *height);
pango_tab_array_new()
PangoTabArray
pango_tab_array_new (gint initial_size, gboolean positions_in_pixels);
PangoTabArray
pango_tab_array_set_tab()
PangoTabArray
PANGO_TAB_LEFT
pango_tab_array_set_tab (PangoTabArray *tabarray, gint tab_index, PangoTabAlign alignment, gint location);
gtk_text_view_set_tabs()
GtkTextView
PangoTabArray
void gtk_text_view_set_tabs (GtkTextView *textview, PangoTabArray *tabs);
pango_tab_array_free()
GtkTextBuffer
GtkTextIter
GtkTextMark
GTK+
GtkTextBuffer: insert
selection_bound
insert
selection_bound
GTK+
GtkTextBuffer: insert
selection_bound
7-4
GtkTextBuffer
7-4
GtkTextBuffer
7-4
GtkTextBuffer
7-4
GtkWidget *entry, *textview; } Widgets;
static void insert_text (GtkButton*, Widgets*);
static void
retrieve_text (GtkButton*, Widgets*);
int main (int argc, char *argv[]) {
GtkWidget *window, *scrolled_win,

```

```

* hbox, * vbox, * insert, * retrieve; Widgets *w = g_slice_new (Widgets); gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "Text
Iterators"); gtk_container_set_border_width (GTK_CONTAINER (window), 10); gtk_widget_set_size_request
(window, -1, 200); w->textview = gtk_text_view_new (); w->entry = gtk_entry_new (); insert =
gtk_button_new_with_label ("Insert Text"); retrieve = gtk_button_new_with_label ("Get Text"); g_signal_connect
(G_OBJECT (insert), "clicked", G_CALLBACK (insert_text), (gpointer) w); g_signal_connect (G_OBJECT
(retrieve), "clicked", G_CALLBACK (retrieve_text), (gpointer) w); scrolled_win = gtk_scrolled_window_new
(NULL, NULL); gtk_container_add (GTK_CONTAINER (scrolled_win), w->textview); hbox = gtk_hbox_new
(FALSE, 5); gtk_box_pack_start_defaults (GTK_BOX (hbox), w->entry); gtk_box_pack_start_defaults
(GTK_BOX (hbox), insert); gtk_box_pack_start_defaults (GTK_BOX (hbox), retrieve); vbox = gtk_vbox_new
(FALSE, 5); gtk_box_pack_start (GTK_BOX (vbox), scrolled_win, TRUE, TRUE, 0); gtk_box_pack_start
(GTK_BOX (vbox), hbox, FALSE, TRUE, 0); gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window); gtk_main(); return 0; } /* Insert the text from the GtkEntry into the GtkTextView.
*/ static void insert_text (GtkButton *button, Widgets *w) { GtkTextBuffer *buffer; GtkTextMark *mark;
GtkTextIter iter; const gchar *text; buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (w->textview)); text =
gtk_entry_get_text (GTK_ENTRY (w->entry)); mark = gtk_text_buffer_get_insert (buffer);
gtk_text_buffer_get_iter_at_mark (buffer, &iter, mark); gtk_text_buffer_insert (buffer, &iter, text, -1); } /* Retrieve
the selected text from the GtkTextView and display it * to the user. */ static void retrieve_text (GtkButton *button,
Widgets *w) { GtkTextBuffer *buffer; GtkTextIter start, end; gchar *text; buffer = gtk_text_view_get_buffer
(GTK_TEXT_VIEW (w->textview)); gtk_text_buffer_get_selection_bounds (buffer, &start, &end); text =
gtk_text_buffer_get_text (buffer, &start, &end, FALSE); g_print ("%s\n", text); }

7-40 GTK+ 的 GtkTextBuffer 接口。
GtkTextBuffer 是 GTK+ 提供的一个用于管理文本的类。它提供了一个用于插入和检索文本的接口。
GtkTextBuffer 的接口如下：

void gtk_text_buffer_get_selection_bounds (GtkTextBuffer *buffer, GtkTextIter *start,
GtkTextIter *end);
void gtk_text_buffer_get_text (GtkTextBuffer *buffer, const GtkTextIter *start,
const GtkTextIter *end, gboolean include_hidden_chars);
gchar* gtk_text_buffer_get_text (GtkTextBuffer *buffer, const
GtkTextIter *start, const GtkTextIter *end, gboolean include_hidden_chars);
void gtk_text_buffer_get_slice (GtkTextBuffer *buffer, const GtkTextIter *start,
const GtkTextIter *end, gint character_offset);
void gtk_text_buffer_get_iter_at_offset (GtkTextBuffer *buffer, GtkTextIter *iter,
gint character_offset);
void gtk_text_buffer_get_iter_at_line_index (GtkTextBuffer *buffer, GtkTextIter *iter,
gint character_offset, gboolean utf8);
void gtk_text_buffer_get_iter_at_line (GtkTextBuffer *buffer, GtkTextIter *iter,
gint character_offset);
void gtk_text_buffer_get_iter_at_line_offset (GtkTextBuffer *buffer, GtkTextIter *iter,
gint character_offset);
void gtk_text_buffer_insert (GtkTextBuffer *buffer, GtkTextIter *iter,
const gchar *text, gint length);
void gtk_text_buffer_insert_at_cursor (GtkTextBuffer *buffer, const gchar *text,
gint length);

```

```

void gtk_text_buffer_insert_at_cursor (GtkTextBuffer *buffer, const gchar *text, gint length);
gtk_text_buffer_delete()
void
gtk_text_buffer_delete (GtkTextBuffer *buffer, GtkTextIter *start, GtkTextIter *end);
delete-range
GtkTextView
7-5
GtkTextView
(GtkTextView*)
(GtkButton*, GtkTextView*)
(GtkButton*, GtkTextView*)
(GtkButton*, GtkTextView*)
(Gtk_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "
Cut, Copy & Paste");
gtk_container_set_border_width (GTK_CONTAINER (window), 10);
textview =
gtk_text_view_new ();
cut = gtk_button_new_from_stock (GTK_STOCK_CUT);
copy =
gtk_button_new_from_stock (GTK_STOCK_COPY);
paste = gtk_button_new_from_stock
(GTK_STOCK_PASTE);
g_signal_connect (G_OBJECT (cut), "clicked", G_CALLBACK (cut_clicked), (gpointer)
textview);
g_signal_connect (G_OBJECT (copy), "clicked", G_CALLBACK (copy_clicked), (gpointer) textview);
g_signal_connect (G_OBJECT (paste), "clicked", G_CALLBACK (paste_clicked), (gpointer) textview);
scrolled_win = gtk_scrolled_window_new (NULL, NULL);
gtk_widget_set_size_request (scrolled_win, 300, 200);
gtk_container_add (GTK_CONTAINER (scrolled_win), textview);
hbox = gtk_hbox_new (TRUE, 5);
gtk_box_pack_start (GTK_BOX (hbox), cut, TRUE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (hbox), copy,
TRUE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (hbox), paste, TRUE, TRUE, 0);
vbox = gtk_vbox_new
(FALSE, 5);
gtk_box_pack_start (GTK_BOX (vbox), scrolled_win, TRUE, TRUE, 0);
gtk_box_pack_start
(GTK_BOX (vbox), hbox, FALSE, TRUE, 0);
gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show_all (window);
gtk_main();
return 0;
}
/* Copy the selected text to the clipboard and remove it
from the buffer. */
static void cut_clicked (GtkButton *cut, GtkTextView *textview) {
GtkClipboard *clipboard =
gtk_clipboard_get (GDK_SELECTION_CLIPBOARD);
GtkTextBuffer *buffer = gtk_text_view_get_buffer
(textview);
gtk_text_buffer_cut_clipboard (buffer, clipboard, TRUE);
}
/* Copy the selected text to the clipboard. */
static void copy_clicked (GtkButton *copy, GtkTextView *textview) {
GtkClipboard *clipboard =
gtk_clipboard_get (GDK_SELECTION_CLIPBOARD);
GtkTextBuffer *buffer = gtk_text_view_get_buffer
(textview);
gtk_text_buffer_copy_clipboard (buffer, clipboard);
}
/* Insert the text from the clipboard into the text
buffer. */
static void paste_clicked (GtkButton *paste, GtkTextView *textview) {
GtkClipboard *clipboard =
gtk_clipboard_get (GDK_SELECTION_CLIPBOARD);
GtkTextBuffer *buffer = gtk_text_view_get_buffer
(textview);
gtk_text_buffer_paste_clipboard (buffer, clipboard, NULL, TRUE);
}
GtkClipboard
gtk_clipboard_get()
GDK_SELECTION_CLIPBOARD
gtk_clipboard_get()
GtkClipboard
GtkTextBuffer
GtkTextBuffer
void gtk_text_buffer_copy_clipboard (GtkTextBuffer *buffer, GtkClipboard *clipboard);
gtk_text_buffer_cut_clipboard()
gtk_text_buffer_cut_clipboard (GtkTextBuffer
*buffer, GtkClipboard *clipboard, gboolean default_editable);
gtk_text_buffer_paste_clipboard()
GtkTextIter
NULL
void gtk_text_buffer_paste_clipboard (GtkTextBuffer *buffer, GtkClipboard *clipboard, GtkTextIter
*override_location, gboolean default_editable);
TRUE
GTK+
gtk_text_iter_forward_search()

```

```

gtk_text_iter_backward_search(), or you can use gtk_text_iter_forward_search() to search forward from the current position.
GTK_STOCK_FIND is a stock ID for a button with the label "Find...".
7-6. A simple GTK application that searches for a string in a text buffer.
7-6. #include <gtk/gtk.h>
typedef struct { GtkWidget *entry, *textview; } Widgets;
static void search(GtkButton*, Widgets*);
int main(int argc, char *argv[]) { GtkWidget *window, *scrolled_win, *vbox, *hbox, *find;
Widgets *w = g_slice_new(Widgets);
gtk_init(&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(window), "Searching Buffers");
gtk_container_set_border_width(GTK_CONTAINER(window), 10);
w->textview = gtk_text_view_new();
w->entry = gtk_entry_new();
gtk_entry_set_text(GTK_ENTRY(w->entry), "Search for ...");
find = gtk_button_new_from_stock(GTK_STOCK_FIND);
g_signal_connect(G_OBJECT(find), "clicked", G_CALLBACK(search), (gpointer)w);
scrolled_win = gtk_scrolled_window_new(NULL, NULL);
gtk_widget_set_size_request(scrolled_win, 250, 200);
gtk_container_add(GTK_CONTAINER(scrolled_win), w->textview);
hbox = gtk_hbox_new(FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), w->entry, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(hbox), find, FALSE, TRUE, 0);
vbox = gtk_vbox_new(FALSE, 5);
gtk_box_pack_start(GTK_BOX(vbox), scrolled_win, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(vbox), hbox, FALSE, TRUE, 0);
gtk_container_add(GTK_CONTAINER(window), vbox);
gtk_widget_show_all(window);
gtk_main();
return 0; }
/* Search for the entered string within the GtkTextView. Then tell the user how many times it was found. */
static void search(GtkButton *button, Widgets *w) { const gchar *find;
gchar *output;
GtkWidget *dialog;
GtkTextBuffer *buffer;
GtkTextIter start, begin, end;
gboolean success;
gint i = 0;
find = gtk_entry_get_text(GTK_ENTRY(w->entry));
buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(w->textview));
gtk_text_buffer_get_start_iter(buffer, &start);
success = gtk_text_iter_forward_search(&start, (gchar*)find, 0, &begin, &end, NULL);
while(success) { gtk_text_iter_forward_char(&start);
success = gtk_text_iter_forward_search(&start, (gchar*)find, 0, &begin, &end, NULL);
start = begin;
i++; }
output = g_strdup_printf("The string '%s' was found %i times!", find, i);
dialog = gtk_message_dialog_new(NULL, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK, output, NULL);
gtk_dialog_run(GTK_DIALOG(dialog));
gtk_widget_destroy(dialog);
g_free(output); }
void gtk_text_buffer_get_start_iter(GtkTextBuffer *buffer, GtkTextIter *iter,
gint lower_search_bound) { gtk_text_iter_set_offset(iter, 0); }
void gtk_text_iter_forward_search(GtkTextIter *iter, gchar *text, gboolean TRUE, gboolean FALSE, gboolean success,
GtkTextIter *start, find, 0, &begin, &end, NULL);
void gtk_text_iter_backward_search(GtkTextIter *iter, gchar *text, gboolean TRUE, gboolean FALSE, gboolean success,
GtkTextIter *start, find, 0, &begin, &end, NULL);
GTK_TEXT_SEARCH_VISIBLE_ONLY: Only search visible text.
GTK_TEXT_SEARCH_TEXT_ONLY: Only search text, not pixbufs.
oxFFFC: Only search non-textual text.
NULL: Search all text.
limit: Start position of the search range.
gboolean gtk_text_iter_backward_search(const GtkTextIter *start_pos, const gchar *text_string,
GtkTextSearchFlags flags, GtkTextIter *match_start, GtkTextIter *match_end, const GtkTextIter *limit);
void gtk_text_buffer_select_range(GtkTextBuffer *buffer, const GtkTextIter *ins, const GtkTextIter *sel_bound);
GtkTextMark* gtk_text_buffer_create_mark(GtkTextBuffer *buffer, const gchar *name, const GtkTextIter *location,
gboolean left_gravity);
void gtk_text_view_scroll_mark_onscreen(GtkTextView *textview, GtkTextMark *mark);

```

```

gtk_text_view_scroll_mark_onscreen()
    gtk_text_view_scroll_to_mark()
void gtk_text_view_scroll_to_mark (GtkTextView
*textview, GtkTextMark *mark, gdouble margin, gboolean use_align, gdouble xalign, gdouble yalign);
(margin)
    FALSE
    0.0
    1.0
    0.5
    (character tall)
gtk_text_view_scroll_to_iter()
gtk_text_view_scroll_to_mark()
    GtkTextMark
    GtkTextIter
    GtkTextBuffer
    GtkTextTag
    rich
    GtkWidget
    7-7
    7-7
    7-7
    (texttags.c)
#include <gtk/gtk.h>
typedef struct { gchar *str; double scale; } text_to_double;
const text_to_double text_scales[] = { { "Quarter Sized", (double) 0.25 }, { "Double Extra Small", PANGO_SCALE_XX_SMALL }, { "Extra Small", PANGO_SCALE_X_SMALL }, { "Small", PANGO_SCALE_SMALL }, { "Medium", PANGO_SCALE_MEDIUM }, { "Large", PANGO_SCALE_LARGE }, { "Extra Large", PANGO_SCALE_X_LARGE }, { "Double Extra Large", PANGO_SCALE_XX_LARGE }, { "Double Sized", (double) 2.0 }, { NULL, 0 } };
static void format (GtkWidget*, GtkTextView*);
static void scale_changed (GtkComboBox*, GtkTextView*);
static void clear_clicked (GtkButton*, GtkTextView*);
int main (int argc, char *argv[]) {
    GtkWidget *window, *scrolled_win, *textview, *hbox, *vbox;
    GtkWidget *bold, *italic, *underline, *strike, *scale, *clear;
    GtkTextBuffer *buffer;
    gint i = 0;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Text Tags");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 500, -1);
    textview = gtk_text_view_new ();
    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview));
    gtk_text_buffer_create_tag (buffer, "bold", "weight", PANGO_WEIGHT_BOLD, NULL);
    gtk_text_buffer_create_tag (buffer, "italic", "style", PANGO_STYLE_ITALIC, NULL);
    gtk_text_buffer_create_tag (buffer, "strike", "strikethrough", TRUE, NULL);
    gtk_text_buffer_create_tag (buffer, "underline", "underline", PANGO_UNDERLINE_SINGLE, NULL);
    bold = gtk_button_new_from_stock (GTK_STOCK_BOLD);
    italic = gtk_button_new_from_stock (GTK_STOCK_ITALIC);
    underline = gtk_button_new_from_stock (GTK_STOCK_UNDERLINE);
    strike = gtk_button_new_from_stock (GTK_STOCK_STRIKETHROUGH);
    clear = gtk_button_new_from_stock (GTK_STOCK_CLEAR);
    scale = gtk_combo_box_new_text();
    /* Add choices to the GtkWidget widget. */
    for (i = 0; text_scales[i].str != NULL; i++) {
        gtk_combo_box_append_text (GTK_COMBO_BOX (scale), text_scales[i].str);
        gtk_text_buffer_create_tag (buffer, text_scales[i].str, "scale", text_scales[i].scale, NULL );
    }
    /* Add the name of the text tag as a data parameter of the object. */
    g_object_set_data (G_OBJECT (bold), "tag", "bold");
    g_object_set_data (G_OBJECT (italic), "tag", "italic");
    g_object_set_data (G_OBJECT (underline), "tag", "underline");
    g_object_set_data (G_OBJECT (strike), "tag", "strike");
    /* Connect each of the buttons and the combo box to the necessary signals. */
    g_signal_connect (G_OBJECT (bold), "clicked", G_CALLBACK (format), (gpointer) textview);
    g_signal_connect (G_OBJECT (italic), "clicked", G_CALLBACK (format), (gpointer) textview);
    g_signal_connect (G_OBJECT (underline), "clicked", G_CALLBACK (format), (gpointer) textview);
    g_signal_connect (G_OBJECT (strike), "clicked", G_CALLBACK (format), (gpointer) textview);
    g_signal_connect (G_OBJECT (scale), "changed", G_CALLBACK (scale_changed), (gpointer) textview);
    g_signal_connect (G_OBJECT (clear), "clicked", G_CALLBACK (clear_clicked), (gpointer) textview);
    /* Pack the widgets into a GtkWidget, GtkWidget, and then into the window. */
    vbox = gtk_vbox_new (TRUE, 5);
    gtk_box_pack_start (GTK_BOX (vbox), bold, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), italic, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), underline, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), strike, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), scale, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), clear,

```

```

FALSE, FALSE, 0); scrolled_win = gtk_scrolled_window_new (NULL, NULL); gtk_container_add
(GTK_CONTAINER (scrolled_win), textview); gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW
(scrolled_win), GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS); hbox = gtk_hbox_new (FALSE, 5);
gtk_box_pack_start (GTK_BOX (hbox), scrolled_win, TRUE, TRUE, 0); gtk_box_pack_start (GTK_BOX (hbox),
vbox, FALSE, TRUE, 0); gtk_container_add (GTK_CONTAINER (window), hbox); gtk_widget_show_all
(window); gtk_main(); return 0; } /* Retrieve the tag from the "tag" object data and apply it to the selection. */ static
void format (GtkWidget *widget, GtkTextView *textview) { GtkTextIter start, end; GtkTextBuffer *buffer; gchar
*tagname; tagname = (gchar*) g_object_get_data (G_OBJECT (widget), "tag"); buffer = gtk_text_view_get_buffer
(textview); gtk_text_buffer_get_selection_bounds (buffer, &start, &end); gtk_text_buffer_apply_tag_by_name
(buffer, tagname, &start, &end); } /* Apply the selected text size property as the tag. */ static void scale_changed
(GtkComboBox *combo, GtkTextView *textview) { const gchar *text; if (gtk_combo_box_get_active (combo) ==
-1) return; text = gtk_combo_box_get_active_text (combo); g_object_set_data (G_OBJECT (combo), "tag",
(gpointer) text); format (GTK_WIDGET (combo), textview); gtk_combo_box_set_active (combo, -1); } /* Remove
all of the tags from the selected text. */ static void clear_clicked (GtkButton *button, GtkTextView *textview) {
GtkTextIter start, end; GtkTextBuffer *buffer; buffer = gtk_text_view_get_buffer (textview);
gtk_text_buffer_get_selection_bounds (buffer, &start, &end); gtk_text_buffer_remove_all_tags (buffer, &start,
&end); }
GtkTextBuffer
GtkTextTag
gtk_text_buffer_create_tag()
GtkTextTag*
gtk_text_buffer_create_tag (GtkTextBuffer *buffer, const gchar *tag_name, const gchar *first_property_name, ...);
GtkTextTag
GtkTextTag
NULL
GtkTextTag
tag = gtk_text_buffer_create_tag (buffer, "
colors", " background", "#000000", "foreground", "#FFFFFF", NULL);
GtkTextTag
C
GtkTextBuffer
gtk_text_buffer_apply_tag_by_name()
start
end
GtkTextTag
gtk_text_buffer_apply_tag()
void gtk_text_buffer_apply_tag_by_name (GtkTextBuffer *buffer,
const gchar *tag_name, const GtkTextIter *start, const GtkTextIter *end);
gtk_text_buffer_remove_tag_by_name()
void
gtk_text_buffer_remove_tag_by_name (GtkTextBuffer *buffer, const gchar *name, const GtkTextIter *start, const
GtkTextIter *end);
bounds
GtkTextTag
gtk_text_buffer_remove_tag()
gtk_text_buffer_remove_all_tags()
GdkPixbuf
GdkPixbuf
pixbuf
GtkTextIter
gtk_text_buffer_insert_pixbuf()
GdkPixbuf
images.c
#include <gtk/gtk.h> #define
IMAGE_UNDO "/path/to/undo.png" #define IMAGE_REDO "/path/to/redo.png" int main (int argc, char *argv[]) {
GtkWidget *window, *scrolled_win, *textview; GdkPixbuf *undo, *redo; GtkTextIter line; GtkTextBuffer *buffer;
gtk_init (&argc, &argv); window = gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title
(GTK_WINDOW (window), "Pixbufs"); gtk_container_set_border_width (GTK_CONTAINER (window), 10);
gtk_widget_set_size_request (window, 200, 150); textview = gtk_text_view_new (); buffer =
gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)); gtk_text_buffer_set_text (buffer, " Undo\n Redo", -1);
/* Create two images and insert them into the text buffer. */ undo = gdk_pixbuf_new_from_file (IMAGE_UNDO,
NULL); gtk_text_buffer_get_iter_at_line (buffer, &line, 0); gtk_text_buffer_insert_pixbuf (buffer, &line, undo);
redo = gdk_pixbuf_new_from_file (IMAGE_REDO, NULL); gtk_text_buffer_get_iter_at_line (buffer, &line, 1);
gtk_text_buffer_insert_pixbuf (buffer, &line, redo); scrolled_win = gtk_scrolled_window_new (NULL, NULL);
gtk_container_add (GTK_CONTAINER (scrolled_win), textview); gtk_container_add (GTK_CONTAINER

```



```
(window), scrolled_win); gtk_widget_show_all (window); gtk_main(); return 0; }
gtk_text_buffer_insert_pixbuf()
    GdkPixbuf
    GdkPixbuf
    void
gtk_text_buffer_insert_pixbuf (GtkTextBuffer *buffer, GtkTextIter *iter, GdkPixbuf *pixbuf);
    pixbuf
    0xFFFC
    pixbuf
    GtkTextBuffer
    pixbuf
    gtk_text_buffer_get_slice()
    GdkPixbuf
    pixbuf
    NULL
    GdkPixbuf*
    GtkTextIter
    GtkTextChildAnchor
    GtkTextView
    7-9
    GtkWidget
    GtkTextView
    7-9
    gtk_main_quit()
    7-9
    (childwidgets.c)
#include <gtk/gtk.h>
int main (int argc, char *argv[]) {
    GtkWidget *window, *scrolled_win, *textview, *button;
    GtkTextChildAnchor *anchor;
    GtkTextIter iter;
    GtkTextBuffer *buffer;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Child Widgets");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    gtk_widget_set_size_request (window, 250, 100);
    textview = gtk_text_view_new ();
    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview));
    gtk_text_buffer_set_text (buffer, "\n Click to exit!", -1);
    /* Create a new child widget anchor at the specified iterator. */
    gtk_text_buffer_get_iter_at_offset (buffer, &iter, 8);
    anchor = gtk_text_buffer_create_child_anchor (buffer, &iter);
    /* Insert a GtkWidget widget at the child anchor. */
    button = gtk_button_new_with_label ("the button");
    gtk_text_view_add_child_at_anchor (GTK_TEXT_VIEW (textview), button, anchor);
    g_signal_connect_swapped (G_OBJECT (button), "clicked", G_CALLBACK (gtk_widget_destroy), (gpointer) window);
    scrolled_win = gtk_scrolled_window_new (NULL, NULL);
    gtk_container_add (GTK_CONTAINER (scrolled_win), textview);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_win), GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS);
    gtk_container_add (GTK_CONTAINER (window), scrolled_win);
    gtk_widget_show_all (window);
    gtk_main();
    return 0;
}
GtkTextChildAnchor
GtkTextBuffer
gtk_text_buffer_create_child_anchor()
    GtkTextChildAnchor*
    GtkTextBuffer *buffer,
    GtkTextIter *iter;
    (anchor)
    GTK+
    (mark)
    gtk_text_view_add_child_at_anchor()
    GdkPixbuf
    0xFFFC
    pixbuf
    void
gtk_text_view_add_child_at_anchor (GtkTextView *textview, GtkWidget *child, GtkTextChildAnchor *anchor);
    0xFFFC
    gtk_text_iter_get_child_anchor()
    NULL
    GtkTextChildAnchor*
    gtk_text_iter_get_child_anchor (const GtkTextIter *iter);
    gtk_text_child_anchor_get_widgets()
    GList*
    gtk_text_child_anchor_get_widgets (GtkTextChildAnchor *anchor);
    GList
    g_list_free()
    GtkSourceView
    GtkWidget
    GtkSourceView
    GtkSourceView
    (line numbering)
    (bracket matching)
    (Undo)/
    (Redo)
    (source marker)
    7-10
    GtkSourceView
    GEdit
    GtkSourceView
    GtkSourceView
    API
    http://gtksourceview.sourceforge.net
    'pkg-config --cflags --libs gtksourceview-1.0'
    GtkSourceView
    GtkTextView
    7-1
    GtkTextView
    Save
    F9
    GTK+
```





```
(GtkWidget *treeview) { GtkCellRenderer *renderer; GtkTreeViewColumn *column; /* Create a new
GtkCellRendererText, add it to the tree view column and * append the column to the tree view. */ renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Buy", renderer, "text",
BUY_IT, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Count", renderer, "text",
QUANTITY, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Product", renderer, "text",
PRODUCT, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); }
GtkTreeView
gtk_tree_view_new()
gtk_tree_view_new_with_model()
gtk_tree_view_set_model()
gtk_tree_view_new_with_model()
GtkTreeView
void gtk_tree_view_set_headers_visible (GtkTreeView *treeview, gboolean visible);
FALSE
void gtk_tree_view_set_rules_hint (GtkTreeView *treeview, gboolean alternate_colors);
GTK+
GtkTreeViewColumn
GtkCellRenderer
GtkCellRendererText
gtk_cell_renderer_text_new()
GtkCellRendererText
g_object_set()
TRUE
gtk_tree_view_column_new_with_attributes()
gtk_tree_view_column_new_with_attributes()
gtk_tree_view_colun_new()
Buy"
column = gtk_tree_view_column_new ();
gtk_tree_view_column_set_title (column, "Buy");
gtk_tree_view_column_pack_start (column, renderer, FALSE);
gtk_tree_view_column_set_attributes (column, renderer, "text", BUY_IT, NULL);
TRUE
gtk_tree_view_column_set_attributes()
NULL
gtk_tree_view_column_pack_start()
gtk_tree_view_column_add_attribute()
void gtk_tree_view_column_add_attribute (GtkTreeViewColumn *column, GtkCellRenderer *renderer,
const gchar *attribute, gint column);
gtk_tree_view_column_new_with_attributes()
g_object_set()
gtk_tree_view_append_column()
gtk_tree_view_insert_column()
```

```

gtk_tree_view_remove_column()()
GtkListStore
gtk_list_store_new()
GtkListStore* gtk_list_store_new (gint
n_columns, /* List of column types */);
gtk_list_store_append()
void gtk_list_store_append
(GtkListStore *store, GtkTreeIter *iter);
gtk_list_store_prepend()
gtk_list_store_insert()
GtkListStore API
gtk_list_store_remove()
GtkTreeIter
gboolean gtk_list_store_remove (GtkListStore *store, GtkTreeIter *iter);
gtk_list_store_clear()
gtk_list_store_set()
gboolean
gtk_list_store_new()
gtk_list_store_set (store, &iter,
BUY_IT, list[i].buy, QUANTITY, list[i].quantity, PRODUCT, list[i].product, -1);
GTK+
terminal
GtkCellRendererText
Boolean
text
GtkListStore
gtk_tree_view_set_model()
g_object_unref()
GtkTreeStore
GtkTreeStore
GtkTreeStore
8-5
GtkTreeStore
GtkListStore
GtkTreeStore
GtkTreeView
8-2
setup_tree_view()
8-2
Grocery List
Cleaning Supplies()
Food()
0
8-2.
GtkTreeStore (treestore.c) #include <gtk/gtk.h> enum { BUY_IT = 0, QUANTITY, PRODUCT, COLUMNS };
enum { PRODUCT_CATEGORY, PRODUCT_CHILD }; typedef struct { gint product_type; gboolean buy; gint
quantity; gchar *product; } GroceryItem; GroceryItem list[] = { { PRODUCT_CATEGORY, TRUE, 0, "Cleaning
Supplies" }, { PRODUCT_CHILD, TRUE, 1, "Paper Towels" }, { PRODUCT_CHILD, TRUE, 3, "Toilet Paper" },
{ PRODUCT_CATEGORY, TRUE, 0, "Food" }, { PRODUCT_CHILD, TRUE, 2, "Bread" }, {
PRODUCT_CHILD, FALSE, 1, "Butter" }, { PRODUCT_CHILD, TRUE, 1, "Milk" }, { PRODUCT_CHILD,
FALSE, 3, "Chips" }, { PRODUCT_CHILD, TRUE, 4, "Soda" }, { PRODUCT_CATEGORY, FALSE, 0, NULL }
}; /* The implementation of this function is the same as in Listing 8-1. */ static void setup_tree_view (GtkWidget*);
int main (int argc, char *argv[]) { GtkWidget *window, *treeview, *scrolled_win; GtkTreeStore *store; GtkTreeIter
iter, child; guint i = 0, j; gtk_init (&argc, &argv); window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "Grocery List"); gtk_container_set_border_width
(GTK_CONTAINER (window), 10); gtk_widget_set_size_request (window, 275, 300); treeview =
gtk_tree_view_new (); setup_tree_view (treeview); store = gtk_tree_store_new (COLUMNS,
G_TYPE_BOOLEAN, G_TYPE_INT, G_TYPE_STRING); while (list[i].product != NULL) { /* If the product type
is a category, count the quantity of all of the products * in the category that are going to be bought. */ if
(list[i].product_type == PRODUCT_CATEGORY) { j = i + 1; /* Calculate how many products will be bought in
the category. */ while (list[j].product != NULL && list[j].product_type != PRODUCT_CATEGORY) { if
(list[j].buy) list[i].quantity += list[j].quantity; j++; } /* Add the category as a new root element. */
gtk_tree_store_append (store, &iter, NULL); gtk_tree_store_set (store, &iter, BUY_IT, list[i].buy, QUANTITY,
list[i].quantity, PRODUCT, list[i].product, -1); } /* Otherwise, add the product as a child of the category. */ else {
gtk_tree_store_append (store, &child, &iter); gtk_tree_store_set (store, &child, BUY_IT, list[i].buy, QUANTITY,
list[i].quantity, PRODUCT, list[i].product, -1); } i++; } gtk_tree_view_set_model (GTK_TREE_VIEW (treeview),
GTK_TREE_MODEL (store)); gtk_tree_view_expand_all (GTK_TREE_VIEW (treeview)); g_object_unref (store);
scrolled_win = gtk_scrolled_window_new (NULL, NULL); gtk_scrolled_window_set_policy
(GTK_SCROLLED_WINDOW (scrolled_win), GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

```

```

gtk_container_add (GTK_CONTAINER (scrolled_win), treeview); gtk_container_add (GTK_CONTAINER
(window), scrolled_win); gtk_widget_show_all (window); gtk_main (); return 0; }
gtk_list_store_new()
gtk_tree_store_new()
gtk_tree_store_append()
gtk_tree_store_append (store, &iter, NULL);
gtk_tree_store_append (store, &child, &iter);
gtk_tree_store_prepend(), gtk_tree_store_insert_before()
gtk_tree_store_set()
gtk_list_store_set()
gtk_tree_store_set (store, &child, BUY_IT, list[i].buy, QUANTITY, list[i].quantity, PRODUCT, list[i].product, -1);
gtk_tree_store_remove()
gboolean gtk_tree_store_remove (GtkTreeStore *store, GtkTreeIter *iter);
gtk_tree_store_clear()
gtk_main()
gtk_tree_view_expand_all()
gtk_tree_view_collapse_all()
GtkTreePath, GtkTreeIter, GtkTreeRowReference
gtk_tree_path_to_string()
gtk_tree_path_new_from_string()
gtk_tree_path_up()
gtk_tree_path_down()
gtk_tree_path_next()
gtk_tree_path_prev()
gtk_tree_path_get_indices()
gint* gtk_tree_path_get_indices (GtkTreePath *path);
GtkTreeModel
GtkTreePath
gtk_tree_row_reference_new()
GtkTreeRowReference*
gtk_tree_row_reference_get_path()
gtk_tree_path_free()
gtk_tree_row_reference_free()
GTK+
GtkTreeModel
GtkTreeIter
GtkTextIter
gtk_tree_model_iter_*(*)
gtk_tree_model_iter_next()
GtkTreeModel API
gtk_tree_model_get_path()
gtk_tree_model_get_iter()

```

path = gtk\_tree\_model\_get\_path (model, &iter); gtk\_tree\_model\_get\_iter (model, &iter, path);  
 gtk\_tree\_path\_free (path);

gtk\_tree\_model\_get\_iter()

**GTK\_TREE\_MODEL\_ITERS\_PERSIST**

GtkTreeModelFlags gtk\_tree\_model\_get\_flags (GtkTreeModel \*model);

**GTK\_SELECTION\_NONE**; **GTK\_SELECTION\_SINGLE**; **GTK\_SELECTION\_BROWSE**; **GTK\_SELECTION\_MULTIPLE**;

**GTK\_SELECTION\_BROWSE**

gtk\_tree\_selection\_get\_selected()

gboolean gtk\_tree\_selection\_get\_selected (GtkTreeSelection \*selection, GtkTreeModel \*\*model, GtkTreeIter \*iter);

**GTK+**

gtk\_tree\_selection\_get\_selected\_rows()

GList\* gtk\_tree\_selection\_get\_selected\_rows (GtkTreeSelection \*selection, GtkTreeModel \*\*model);

static gboolean foreach\_func (GtkTreeModel \*model, GtkTreePath \*path, GtkTreeIter \*iter, gpointer data) {
 }

**GTK\_STOCK\_ADD** **GTK\_STOCK\_REMOVE**

**GtkDialog**

**New Product**

```

static void add_product (GtkButton *add, GtkTreeView *treeview) {
  GtkWidget *dialog, *table, *combobox, *entry, *spin, *check;
  GtkTreeIter iter, child;
  GtkTreePath *path;
  GtkTreeModel *model;
  const gchar *product;
  gchar *category, *name;
  gint quantity, i = 0;
  gboolean buy;
  /* Create a dialog that will be used to create a new product. */
  dialog = gtk_dialog_new_with_buttons ("Add a Product", NULL, GTK_DIALOG_MODAL,
  GTK_STOCK_ADD, GTK_RESPONSE_OK, GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL, NULL);
  /* Create widgets that will be packed into the dialog. */
  combobox = gtk_combo_box_new_text ();
  entry = gtk_entry_new ();
  spin = gtk_spin_button_new_with_range (0, 100, 1);
  check =
  
```

```

gtk_check_button_new_with_mnemonic ("_Buy the Product"); gtk_spin_button_set_digits (GTK_SPIN_BUTTON
(spin), 0); /* Add all of the categories to the combo box. */ while (list[i].product != NULL) { if (list[i].product_type
== PRODUCT_CATEGORY) gtk_combo_box_append_text (GTK_COMBO_BOX (combobox), list[i].product);
i++; } table = gtk_table_new (4, 2, FALSE); gtk_table_set_row_spacings (GTK_TABLE (table), 5);
gtk_table_set_col_spacings (GTK_TABLE (table), 5); gtk_container_set_border_width (GTK_CONTAINER
(table), 5); /* Pack the table that will hold the dialog widgets. */ gtk_table_attach (GTK_TABLE (table),
gtk_label_new ("Category:"), 0, 1, 0, 1, GTK_SHRINK | GTK_FILL, GTK_SHRINK | GTK_FILL, 0, 0);
gtk_table_attach (GTK_TABLE (table), combobox, 1, 2, 0, 1, GTK_EXPAND | GTK_FILL, GTK_SHRINK |
GTK_FILL, 0, 0); gtk_table_attach (GTK_TABLE (table), gtk_label_new ("Product:"), 0, 1, 1, 2, GTK_SHRINK |
GTK_FILL, GTK_SHRINK | GTK_FILL, 0, 0); gtk_table_attach (GTK_TABLE (table), entry, 1, 2, 1, 2,
GTK_EXPAND | GTK_FILL, GTK_SHRINK | GTK_FILL, 0, 0); gtk_table_attach (GTK_TABLE (table),
gtk_label_new ("Quantity:"), 0, 1, 2, 3, GTK_SHRINK | GTK_FILL, GTK_SHRINK | GTK_FILL, 0, 0);
gtk_table_attach (GTK_TABLE (table), spin, 1, 2, 2, 3, GTK_EXPAND | GTK_FILL, GTK_SHRINK |
GTK_FILL, 0, 0); gtk_table_attach (GTK_TABLE (table), check, 1, 2, 3, 4, GTK_EXPAND | GTK_FILL,
GTK_SHRINK | GTK_FILL, 0, 0); gtk_box_pack_start_defaults (GTK_BOX (GTK_DIALOG (dialog)->vbox),
table); gtk_widget_show_all (dialog); /* If the user presses OK, verify the entries and add the product. */ if
(gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_OK) { quantity = (gint)
gtk_spin_button_get_value (GTK_SPIN_BUTTON (spin)); product = gtk_entry_get_text (GTK_ENTRY (entry));
category = gtk_combo_box_get_active_text (GTK_COMBO_BOX (combobox)); buy =
gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (check)); if (g_ascii_strcasecmp (product, "") || category
== NULL) { g_warning ("All of the fields were not correctly filled out!"); gtk_widget_destroy (dialog); if (category !
= NULL) g_free (category) return; } model = gtk_tree_view_get_model (treeview);
gtk_tree_model_get_iter_from_string (model, &iter, "0"); /* Retrieve an iterator pointing to the selected category. */
do { gtk_tree_model_get (model, &iter, PRODUCT, &name, -1); if (g_ascii_strcasecmp (name, category) == 0) {
g_free (name); break; } g_free (name); } while (gtk_tree_model_iter_next (model, &iter)); /* Convert the category
iterator to a path so that it will not become invalid * and add the new product as a child of the category. */ path =
gtk_tree_model_get_path (model, &iter); gtk_tree_store_append (GTK_TREE_STORE (model), &child, &iter);
gtk_tree_store_set (GTK_TREE_STORE (model), &child, BUY_IT, buy, QUANTITY, quantity, PRODUCT,
product, -1); /* Add the quantity to the running total if it is to be purchased. */ if (buy) { gtk_tree_model_get_iter
(model, &iter, path); gtk_tree_model_get (model, &iter, QUANTITY, &i, -1); i += quantity; gtk_tree_store_set
(GTK_TREE_STORE (model), &iter, QUANTITY, i, -1); } gtk_tree_path_free (path); g_free (category); }
gtk_widget_destroy (dialog); }
gtk_tree_model_get_iter_from_string()
gtk_tree_model_iter_next()
gtk_tree_model_get()
GtkTreeModel
gtk_tree_store_set()
PRODUCT, &name, -1); if (g_ascii_strcasecmp (name, category) == 0) break;
g_ascii_strcasecmp()
iter
GtkTreeIter
path = gtk_tree_model_get_path (model, &iter); gtk_tree_store_append (GTK_TREE_STORE
(model), &child, &iter); gtk_tree_store_set (GTK_TREE_STORE (model), &child, BUY_IT, buy, QUANTITY,
quantity, PRODUCT, product, -1);
gtk_tree_store_append()
iter
gtk_tree_store_set()
8-6 GtkComboBox
GtkComboBox
GtkTreeModel
gtk_combo_box_new_text()
GtkComboBox
gtk_combo_box_new_text()
GtkTreeModel
void gtk_combo_box_append_text
(GtkComboBox *combobox, const gchar *text); void gtk_combo_box_prepend_text (GtkComboBox *combobox,

```



```

const gchar *text); void gtk_combo_box_insert_text (GtkComboBox *combobox, gint position, const gchar *text);
gtk_combo_box_remove_text()
gtk_combo_box_get_active_text()
gtk_combo_box_new_text()
gtk_combo_box_new()
gtk_combo_box_set_model()
gtk_combo_box_new()
gtk_combo_box_get_active (GtkComboBox *combobox); gboolean
gtk_combo_box_get_active_iter (GtkComboBox *combobox, GtkTreeIter *iter);
gtk_combo_box_get_active()
gtk_combo_box_get_active_iter()
remove_row()
remove_products()
GTK_STOCK_REMOVE
static void remove_row (GtkTreeRowReference *ref,
GtkTreeModel *model) { GtkTreeIter parent, iter; GtkTreePath *path; gboolean buy; gint quantity, pnum; /*
Convert the tree row reference to a path and retrieve the iterator. */ path = gtk_tree_row_reference_get_path (ref);
gtk_tree_model_get_iter (model, &iter, path); /* Only remove the row if it is not a root row. */ if
(gtk_tree_model_iter_parent (model, &parent, &iter)) { gtk_tree_model_get (model, &iter, BUY_IT, &buy,
QUANTITY, &quantity, -1); gtk_tree_model_get (model, &parent, QUANTITY, &pnum, -1); if (buy) { pnum -=
quantity; gtk_tree_store_set (GTK_TREE_STORE (model), &parent, QUANTITY, pnum, -1); }
gtk_tree_model_get_iter (model, &iter, path); gtk_tree_store_remove (GTK_TREE_STORE (model), &iter); } }
static void remove_products (GtkButton *remove, GtkTreeView *treeview) { GtkTreeSelection *selection;
GtkTreeRowReference *ref; GtkTreeModel *model; GList *rows, *ptr, *references = NULL; selection =
gtk_tree_view_get_selection (treeview); model = gtk_tree_view_get_model (treeview); rows =
gtk_tree_selection_get_selected_rows (selection, &model); /* Create tree row references to all of the selected rows.
*/ ptr = rows; while (ptr != NULL) { ref = gtk_tree_row_reference_new (model, (GtkTreePath*) ptr->data);
references = g_list_prepend (references, gtk_tree_row_reference_copy (ref)); gtk_tree_row_reference_free (ref); ptr
= ptr->next; } /* Remove each of the selected rows pointed to by the row reference. */ g_list_foreach (references,
(GFunc) remove_row, model); /* Free the tree paths, tree row references and lists. */ g_list_foreach (references,
(GFunc) gtk_tree_row_reference_free, NULL); g_list_free (rows); g_list_free (references); }
GTK_STOCK_REMOVE
gtk_tree_selection_get_selected_rows()
gtk_tree_selection_selected_foreach()
g_list_foreach()
remove_row()
gtk_tree_model_iter_parent()
parent
FALSE
gtk_tree_model_iter_parent (model, &parent, &iter)
GtkTreeView
row-activated
Shift-
Return,
Enter
gtk_tree_view_row_activated()
static void row_activated (GtkTreeView *treeview,
GtkTreePath *path, GtkTreeViewColumn *column, gpointer data) { GtkTreeModel *model; GtkTreeIter iter; model
= gtk_tree_view_get_model (treeview); if (gtk_tree_model_get_iter (model, &iter, path)) { /* Handle the selection . .
*/ } }
8-8
gtk_tree_model_get_iter()
GtkEntry
GTK+
GtkCellRendererText
edited
textual component
8-8
8-8
Enter
edited
8-9
GtkListStore Grocery List
8-9
editable.c
static void setup_tree_view
(GtkWidget *treeview) { GtkCellRenderer *renderer; GtkTreeViewColumn *column; renderer =

```

```

gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Buy", renderer, "text",
BUY_IT, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Count", renderer, "text",
QUANTITY, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); /* Set up the third
column in the tree view to be editable. */ renderer = gtk_cell_renderer_text_new (); g_object_set (renderer, "
editable", TRUE, "editable-set", TRUE, NULL); g_signal_connect (G_OBJECT (renderer), "edited",
G_CALLBACK (cell_edited), (gpointer) treeview); column = gtk_tree_view_column_new_with_attributes
("Product", renderer, "text", PRODUCT, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview),
column); } /* Apply the changed text to the cell if it is not an empty string. */ static void cell_edited
(GtkCellRendererText *renderer, gchar *path, gchar *new_text, GtkTreeView *treeview) { GtkTreeIter iter;
GtkTreeModel *model; if (g_ascii_strcasecmp (new_text, "") != 0) { model = gtk_tree_view_get_model (treeview);
if (gtk_tree_model_get_iter_from_string (model, &iter, path)) gtk_list_store_set (GTK_LIST_STORE (model),
&iter, PRODUCT, new_text, -1); } }
// 8-9: GtkCellRendererText is made editable.
g_object_set (renderer, "editable", TRUE, "editable-set", TRUE, NULL);
g_object_set (renderer, "editable", TRUE, "editable-set", TRUE, NULL);
GtkCellRendererText edited GtkTreePath, GtkTreeIter, GtkEntry, Enter, edited
if (gtk_tree_model_get_iter_from_string (model, &iter, path))
gtk_list_store_set (GTK_LIST_STORE (model), &iter, PRODUCT, new_text, -1);
gtk_tree_model_get_iter_from_string() GtkTreePath GtkTreeIter TRUE
GTK+ GtkTreeIter
gtk_list_store_set() GtkListStore PRODUCT new_text
8-9: GtkCellRendererText text
8-9: GtkCellRendererText text
8-10: g_object_set()
256 (web-safe colors) 8-10: (celldatafunctions.c)
#include <gtk/gtk.h> enum { COLOR = 0, COLUMNS }; const gchar *clr[6] = { "00", "33", "66", "99", "CC", "FF"
}; static void setup_tree_view (GtkWidget*); static void cell_data_func (GtkTreeViewColumn*, GtkCellRenderer*,
GtkTreeModel*, GtkTreeIter*, gpointer); int main (int argc, char *argv[]) { GtkWidget *window, *treeview,
*scrolled_win; GtkListStore *store; GtkTreeIter iter; guint i, j, k; gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "Color
List"); gtk_container_set_border_width (GTK_CONTAINER (window), 10); gtk_widget_set_size_request
(window, 250, 175); treeview = gtk_tree_view_new (); setup_tree_view (treeview); store = gtk_list_store_new
(COLUMNS, G_TYPE_STRING); /* Add all of the products to the GtkListStore. */ for (i = 0; i < 6; i++) for (j = 0;
j < 6; j++) for (k = 0; k < 6; k++) { gchar *color = g_strconcat ("#", clr[i], clr[j], clr[k], NULL);
gtk_list_store_append (store, &iter); gtk_list_store_set (store, &iter, COLOR, color, -1); g_free (color); }
gtk_tree_view_set_model (GTK_TREE_VIEW (treeview), GTK_TREE_MODEL (store)); g_object_unref (store);
scrolled_win = gtk_scrolled_window_new (NULL, NULL); gtk_scrolled_window_set_policy
(GTK_SCROLLED_WINDOW (scrolled_win), GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
gtk_container_add (GTK_CONTAINER (scrolled_win), treeview); gtk_container_add (GTK_CONTAINER
(window), scrolled_win); gtk_widget_show_all (window); gtk_main (); return 0; } /* Add three columns to the
GtkTreeView. All three of the columns will be * displayed as text, although one is a gboolean value and another is *
an integer. */ static void setup_tree_view (GtkWidget *treeview) { GtkCellRenderer *renderer;
GtkTreeViewColumn *column; renderer = gtk_cell_renderer_text_new (); column =
gtk_tree_view_column_new_with_attributes ("Standard Colors", renderer, "text", COLOR, NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);

```

```

gtk_tree_view_column_set_cell_data_func (column, renderer, cell_data_func, NULL, NULL); } static void
cell_data_func (GtkTreeViewColumn *column, GtkCellRenderer *renderer, GtkTreeModel *model, GtkTreeIter
*iter, gpointer data) { gchar *text; /* Get the color string stored by the column and make it the foreground color. */
gtk_tree_model_get (model, iter, COLOR, &text, -1); g_object_set (renderer, "foreground", "#FFFFFF", "
foreground-set", TRUE, "background", text, "background-set", TRUE, "text", text, NULL); g_free (text); }

```

GTK+ uses a `GtkCellRendererText` widget to display text. The `GtkCellRendererText` widget has a `text` property that can be set to a string. The `GtkCellRendererText` widget also has a `background-set` property that can be set to a string. The `GtkCellRendererText` widget also has a `foreground-set` property that can be set to a string.

```

gtk_tree_view_column_set_cell_data_func() sets the cell_data_func property of the column to the function pointer. The function pointer is a pointer to a function that takes a GtkTreeViewColumn, a GtkCellRenderer, a GtkTreeModel, a GtkTreeIter, and a gpointer as arguments and returns a gchar*. The function pointer is set to cell_data_func. The function pointer is set to NULL if the function pointer is NULL.

```

The `GtkCellRendererText` widget has a `text` property that can be set to a string. The `GtkCellRendererText` widget also has a `background-set` property that can be set to a string. The `GtkCellRendererText` widget also has a `foreground-set` property that can be set to a string.

```

gtk_tree_view_column_set_cell_data_func() sets the cell_data_func property of the column to the function pointer. The function pointer is a pointer to a function that takes a GtkTreeViewColumn, a GtkCellRenderer, a GtkTreeModel, a GtkTreeIter, and a gpointer as arguments and returns a gchar*. The function pointer is set to cell_data_func. The function pointer is set to NULL if the function pointer is NULL.

```

The `GtkCellRendererToggle` widget has a `toggled` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `radio` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `activatable` property that can be set to a Boolean.

```

gtk_cell_renderer_toggle_new() creates a new GtkCellRendererToggle widget. The widget has a text property that can be set to a string. The widget also has a background-set property that can be set to a string. The widget also has a foreground-set property that can be set to a string. The widget also has a toggled property that can be set to a Boolean. The widget also has a radio property that can be set to a Boolean. The widget also has an activatable property that can be set to a Boolean.

```

The `GtkCellRendererToggle` widget has a `toggled` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `radio` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `activatable` property that can be set to a Boolean.

```

gtk_cell_renderer_toggle_set_radio() sets the radio property of the widget to the Boolean value. The Boolean value is TRUE if the widget is the only widget in the list with the radio property set to TRUE. The Boolean value is FALSE if the widget is not the only widget in the list with the radio property set to TRUE.

```

The `GtkCellRendererToggle` widget has a `toggled` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `radio` property that can be set to a Boolean. The `GtkCellRendererToggle` widget also has a `activatable` property that can be set to a Boolean.

```

GtkCellRendererPixbuf is a widget that displays a pixbuf. The widget has a pixbuf property that can be set to a GdkPixbuf. The widget also has a background-set property that can be set to a string. The widget also has a foreground-set property that can be set to a string.

```

The `GtkCellRendererPixbuf` widget has a `pixbuf` property that can be set to a `GdkPixbuf`. The `GtkCellRendererPixbuf` widget also has a `background-set` property that can be set to a string. The `GtkCellRendererPixbuf` widget also has a `foreground-set` property that can be set to a string.

```

static void setup_tree_view (GtkWidget *treeview) { GtkCellRenderer *renderer; GtkTreeViewColumn *column; /* Create a tree view column with two renderers, one a pixbuf and one text. */ column = gtk_tree_view_column_new (); gtk_tree_view_column_set_title (column, "Products"); renderer = gtk_cell_renderer_pixbuf_new (); gtk_tree_view_column_pack_start (column, renderer, FALSE); gtk_tree_view_column_set_attributes (column, renderer, "pixbuf", ICON, NULL); renderer = gtk_cell_renderer_text_new (); gtk_tree_view_column_pack_start (column, renderer, TRUE); gtk_tree_view_column_set_attributes (column, renderer, "text", PRODUCT, NULL);

```

```

gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); } gtk_cell_renderer_pixbuf_new()
GtkCellRendererPixbuf
gtk_tree_view_column_pack_start()
GtkCellRendererPixbuf
pixbuf
stock-id
GdkPixbuf
GTK+ 2.10
D
GtkTreeStore
retract
pixbuf
pixbuf-expander-open
pixbuf-expander-closed
GdkPixbuf
open
closed
GDK_TYPE_PIXBUF
GdkPixbuf
GdkPixbuf
1
g_object_unref()
4
GtkSpinButton
GtkCellRendererText
GtkCellRendererSpin
GtkEntry
GtkSpinButton
GtkCellRendererSpin
8-12
8-12
8-12
8-13
8-13
static void cell_data_func (GtkTreeViewColumn *column, GtkCellRenderer *renderer,
GtkTreeModel *model, GtkTreeIter *iter, gpointer data) { gfloat value; gchar *text; /* Retrieve the current value
and render it with no decimal places. */ gtk_tree_model_get (model, iter, QUANTITY, &value, -1); text =
g_strdup_printf ("%f", value); g_object_set (renderer, "text", text, NULL); g_free (text); } GtkCellRendererText
0
GtkCellRendererSpin
GtkCellRendererSpin
GtkAdjustment
8-14
Grocery List
Quantity
GtkCellRendererSpin
8-14
static void setup_tree_view (GtkWidget *treeview) { GtkCellRenderer *renderer;
GtkTreeViewColumn *column; GtkAdjustment *adj; adj = GTK_ADJUSTMENT (gtk_adjustment_new (0.0, 0.0,
100.0, 1.0, 2.0, 2.0)); renderer = gtk_cell_renderer_spin_new (); g_object_set (renderer, "editable", TRUE, "
adjustment", adj, "digits", 0, NULL); g_signal_connect (G_OBJECT (renderer), "edited", G_CALLBACK
(cell_edited), (gpointer) treeview); column = gtk_tree_view_column_new_with_attributes ("Count", renderer, "
text", QUANTITY, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); /* ... Add a
cell renderer for the PRODUCT column ... */ /* Apply the changed text to the cell. */ static void cell_edited
(GtkCellRendererText *renderer, gchar *path, gchar *new_text, GtkTreeView *treeview) { GtkTreeIter iter;
GtkTreeModel *model; GtkAdjustment *adjustment; gdouble value; /* Retrieve the current value stored by the spin
button renderer's adjustment. */ g_object_get (renderer, "adjustment", &adjustment, NULL); value =
gtk_adjustment_get_value (adjustment); model = gtk_tree_view_get_model (treeview); if
(gtk_tree_model_get_iter_from_string (model, &iter, path)) gtk_list_store_set (GTK_LIST_STORE (model), &iter,
QUANTITY, value, -1); }
gtk_cell_renderer_spin()
g_object_set()
editable, adjustment, digits
g_object_set (renderer, "editable", TRUE, "
adjustment", adj, " digits", 0, NULL);
GtkCellRendererSpin
adjustment, climb-rate, digits
GtkAdjustment
acceleration rate
climb rate
GtkCellRendererSpin
GtkCellRendererText
TRUE
editable
GtkCellRendererText
edited
Enter
cell_edited()
new_text
gtk_list_store_set()
gtk_adjustment_get_value()
QUANTITY
G_TYPE_FLOAT
GtkCellRendererCombo
GtkComboBox
GtkCellRendererCombo
GtkCellRendererText
GtkEntry
GtkComboBox
GtkCellRendererCombo
8-13
8-13
GtkCellRendererCombo
GtkTreeModel
8-15
8-15
Grocery List
QUANTITY
GtkCellRendererCombo
8-15
static

```

```

void setup_tree_view (GtkWidget *treeview) { GtkCellRenderer *renderer; GtkTreeViewColumn *column;
GtkListStore *model; GtkTreeIter iter; /* Create a GtkListStore that will be used for the combo box renderer. */
model = gtk_list_store_new (1, G_TYPE_STRING); gtk_list_store_append (model, &iter); gtk_list_store_set
(model, &iter, 0, "None", -1); gtk_list_store_append (model, &iter); gtk_list_store_set (model, &iter, 0, "One", -1);
gtk_list_store_append (model, &iter); gtk_list_store_set (model, &iter, 0, "Half a Dozen", -1);
gtk_list_store_append (model, &iter); gtk_list_store_set (model, &iter, 0, "Dozen", -1); gtk_list_store_append
(model, &iter); gtk_list_store_set (model, &iter, 0, "Two Dozen", -1); /* Create the GtkCellRendererCombo and
add the tree model. Then, add the * renderer to a new column and add the column to the GtkTreeView. */ renderer =
gtk_cell_renderer_combo_new (); g_object_set (renderer, "text-column", 0, "editable", TRUE, "has-entry", TRUE, "
model", model, NULL); column = gtk_tree_view_column_new_with_attributes ("Count", renderer, "text",
QUANTITY, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); g_signal_connect
(G_OBJECT (renderer), "edited", G_CALLBACK (cell_edited), (gpointer) treeview); renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Product", renderer, "text",
PRODUCT, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); /* Apply the
changed text to the cell. */ static void cell_edited (GtkCellRendererText *renderer, gchar *path, gchar *new_text,
GtkTreeView *treeview) { GtkTreeIter iter; GtkTreeModel *model; /* Make sure the text is not empty. If not, apply
it to the tree view cell. */ if (g_ascii_strcasecmp (new_text, "") != 0) { model = gtk_tree_view_get_model
(treeview); if (gtk_tree_model_get_iter_from_string (model, &iter, path)) gtk_list_store_set (GTK_LIST_STORE
(model), &iter, QUANTITY, new_text, -1); } }
GtkCellRendererCombo GtkRendererText: has-entry, model, text-column
g_object_set
(renderer, "text-column", 0, "editable", TRUE, "has-entry", TRUE, "model", model, NULL);
GtkCellRendererText
model GtkTreeModel, editable TRUE
GtkEntry
GtkComboBoxEntry
has-entry TRUE
GtkCellRendererCombo
text
edited
8-15
new_text
GtkCellRendererProgress
GtkProgressBar
(pulsing)
GtkCellRendererProgress
8-14
GtkCellRendererProgress
text value
value
100
100
G_TYPE_INT
GtkCellRendererProgress
text
"67% Complete", "3 of 80 Files Processed", "Installing
foo..."
GtkCellRendererProgress
GTK+ 2.10
GtkCellRendererAccel
8-15
8-15
8-16
GtkCellRendererText
(mask key)
8-16
(accelerators.c)
#include <gtk/gtk.h> #include <gdk/gdkkeysyms.h> enum { ACTION = 0, MASK, VALUE, COLUMNS }; typedef
struct { gchar *action; GdkModifierType mask; guint value; } Accelerator; const Accelerator list[] = { { "Cut",
GDK_CONTROL_MASK, GDK_X }, { "Copy", GDK_CONTROL_MASK, GDK_C }, { " Paste",
GDK_CONTROL_MASK, GDK_V }, { " New", GDK_CONTROL_MASK, GDK_N }, { "Open",
GDK_CONTROL_MASK, GDK_O }, { "Print", GDK_CONTROL_MASK, GDK_P }, { NULL, NULL, NULL }
}; static void setup_tree_view (GtkWidget*); static void accel_edited (GtkCellRendererAccel*, gchar*, guint,

```

```

GtkModifierType, guint, GtkTreeView*); int main (int argc, char *argv[]) { GtkWidget *window, *treeview,
*scrolled_win; GtkListStore *store; GtkTreeIter iter; guint i = 0; gtk_init (&argc, &argv); window =
gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW (window), "
Accelerator Keys"); gtk_container_set_border_width (GTK_CONTAINER (window), 10);
gtk_widget_set_size_request (window, 250, 250); treeview = gtk_tree_view_new (); setup_tree_view (treeview);
store = gtk_list_store_new (COLUMNS, G_TYPE_STRING, G_TYPE_INT, G_TYPE_UINT); /* Add all of the
keyboard accelerators to the GtkListStore. */ while (list[i].action ! = NULL) { gtk_list_store_append (store,
&iter); gtk_list_store_set (store, &iter, ACTION, list[i].action, MASK, (gint) list[i].mask, VALUE, list[i].value, -1);
i++; } gtk_tree_view_set_model (GTK_TREE_VIEW (treeview), GTK_TREE_MODEL (store)); g_object_unref
(store); scrolled_win = gtk_scrolled_window_new (NULL, NULL); gtk_scrolled_window_set_policy
(GTK_SCROLLED_WINDOW (scrolled_win), GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
gtk_container_add (GTK_CONTAINER (scrolled_win), treeview); gtk_container_add (GTK_CONTAINER
(window), scrolled_win); gtk_widget_show_all (window); gtk_main (); return 0; } /* Create a tree view with two
columns. The first is an action and the * second is a keyboard accelerator. */ static void setup_tree_view
(GtkWidget *treeview) { GtkCellRenderer *renderer; GtkTreeViewColumn *column; renderer =
gtk_cell_renderer_text_new (); column = gtk_tree_view_column_new_with_attributes ("Buy", renderer, " text",
ACTION, NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); renderer =
gtk_cell_renderer_accel_new (); g_object_set (renderer, "accel-mode",
GTK_CELL_RENDERER_ACCEL_MODE_GTK, "editable", TRUE, NULL); column =
gtk_tree_view_column_new_with_attributes ("Buy", renderer, "accel-mods", MASK, "accel-key", VALUE,
NULL); gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column); g_signal_connect (G_OBJECT
(renderer), "accel_edited", G_CALLBACK (accel_edited), (gpointer) treeview); } /* Apply the new keyboard
accelerator key and mask to the cell. */ static void accel_edited (GtkCellRendererAccel *renderer, gchar *path,
guint accel_key, GdkModifierType mask, guint hardware_keycode, GtkTreeView *treeview) { GtkTreeModel
*model; GtkTreeIter iter; model = gtk_tree_view_get_model (treeview); if (gtk_tree_model_get_iter_from_string
(model, &iter, path)) gtk_list_store_set (GTK_LIST_STORE (model), &iter, MASK, (gint) mask, VALUE,
accel_key, -1); }
gtk_cell_renderer_accel_new()
GtkCellRendererAccel
GtkCellRendererAccel
g_object_get()
accel-key:
accel-mode: GtkCellRendererAccelMode
GTK_CELL_RENDERER_ACCEL_MODE_OTHER
accel-mods: GdkModifierType
keycode:
GDK_SHIFT_MASK: Shift
GDK_CONTROL_MASK: Ctrl
GDK_MOD_MASK, GDK_MOD2_MASK, GDK_MOD3_MASK, GDK_MOD4_MASK,
GDK_MOD5_MASK: Alt
GDK_SUPER_MASK: 2.10
GDK_HYPER_MASK: 2.10
GDK_META_MODIFIER: 2.10
GtkCellRendererAccel
(accel-mods)
G_TYPE_INT
G_TYPE_UINT
GdkModifierType
gint
store = gtk_list_store_new (COLUMNS,
G_TYPE_STRING, G_TYPE_INT, G_TYPE_UINT);
GtkCellRendererAccel
accel-cleared
editable
TRUE
gtk_list_store_set()
8-1
GtkTreeView
8-1.
Grocery List
GtkTreeView
GtkListStore
www.gtkbook.com

```



```

This will make sure * that the accelerators will work from application load. */ static void create_popup_menu
(GtkWidget *menu, GtkWidget *progress) { GtkWidget *pulse, *fill, *clear, *separator; pulse =
gtk_menu_item_new_with_label ("Pulse Progress"); fill = gtk_menu_item_new_with_label ("Set as Complete");
clear = gtk_menu_item_new_with_label ("Clear Progress"); separator = gtk_separator_menu_item_new ();
g_signal_connect (G_OBJECT (pulse), "activate", G_CALLBACK (pulse_activated), progress); g_signal_connect
(G_OBJECT (fill), "activate", G_CALLBACK (fill_activated), progress); g_signal_connect (G_OBJECT (clear), "
activate", G_CALLBACK (clear_activated), progress); gtk_menu_shell_append (GTK_MENU_SHELL (menu),
pulse); gtk_menu_shell_append (GTK_MENU_SHELL (menu), separator); gtk_menu_shell_append
(GTK_MENU_SHELL (menu), fill); gtk_menu_shell_append (GTK_MENU_SHELL (menu), clear);
gtk_menu_attach_to_widget (GTK_MENU (menu), progress, NULL); gtk_widget_show_all (menu); }
button-press-event
GdkEventButton button member 3
GtkWidget popup-menu
GTK+
gtk_menu_new()
GtkMenuItem
GtkWidget*
gtk_menu_item_new_with_label(), gtk_menu_item_new_with_mnemonic()
gtk_menu_item_new_with_label (const gchar *label);
Alt
GtkSeparatorMenuItem
activate
activate-item
activate-item
GtkMenuItem
9-2
100%
GtkMenu
GtkMenuShell
gtk_menu_shell_append()
void gtk_menu_shell_append (GtkMenuShell *menu_shell, GtkWidget *child);
gtk_menu_shell_prepend()
gtk_menu_shell_insert()
GtkWidget
gtk_menu_attach_to_widget()
void gtk_menu_attach_to_widget (GtkMenu *menu, GtkWidget
*attach_widget, GtkWidgetDetachFunc detacher);
button-press-event
9-2
(GtkWidget
*eventbox, GdkEventButton *event, GtkWidget *menu) { if ((event->button == 3) && (event->type ==
GDK_BUTTON_PRESS)) { gtk_menu_popup (GTK_MENU (menu), NULL, NULL, NULL, NULL,
event->button, event->time); return TRUE; } return FALSE; }
static void pulse_activated (GtkMenuItem *item,
GtkProgressBar *progress) { gtk_progress_bar_pulse (progress); gtk_progress_bar_set_text (progress, " Pulse!"); }
static void fill_activated (GtkMenuItem *item, GtkProgressBar *progress) { gtk_progress_bar_set_fraction
(progress, 1.0); gtk_progress_bar_set_text (progress, "One Hundred Percent"); }
static void clear_activated
(GtkMenuItem *item, GtkProgressBar *progress) { gtk_progress_bar_set_fraction (progress, 0.0);
gtk_progress_bar_set_text (progress, "Reset to Zero"); }
button-press-event
gtk_menu_popup()
void gtk_menu_popup (GtkMenu *menu, GtkWidget *parent_menu_shell, GtkWidget
*parent_menu_item, GtkWidgetPositionFunc func, gpointer func_data, guint button, guint32 event_time);
(event->time)
(event->button)
NULL
0
gtk_get_current_event_time()
GDK_CURRENT_TIME
parent_menu_shell, parent_menu_item, func, func_data
parent_menu_shell
(menu shell)

```



```

parent_menu_item  gtk_widget_get_child(parent_menu_item);
GtkMenuPositionFunc  callback = (GtkMenuPositionFunc)gtk_widget_get_child(parent_menu_item);
func_data  (GtkMenuPositionFunc)callback;
parent_menu_item  gtk_widget_get_child(parent_menu_item);
NULL  gtk_widget_get_child(parent_menu_item);
Ctrl  Shift(modifier)  gtk_widget_get_child(parent_menu_item);
9-3  gtk_widget_get_child(parent_menu_item);
Ctrl+P  Ctrl+F  Ctrl+C  9-3.  gtk_widget_get_child(parent_menu_item);
(GtkWidget *menu, GtkWidget *window, GtkWidget
*progress) { GtkWidget *pulse, *fill, *clear, *separator; GtkAccelGroup *group; /* Create a keyboard accelerator
group for the application. */ group = gtk_accel_group_new (); gtk_window_add_accel_group (GTK_WINDOW
(window), group); gtk_menu_set_accel_group (GTK_MENU (menu), group); pulse =
gtk_menu_item_new_with_label ("Pulse Progress"); fill = gtk_menu_item_new_with_label ("Set as Complete");
clear = gtk_menu_item_new_with_label ("Clear Progress"); separator = gtk_separator_menu_item_new (); /* Add
the necessary keyboard accelerators. */ gtk_widget_add_accelerator (pulse, "activate", group, GDK_P,
GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE); gtk_widget_add_accelerator (fill, "activate", group, GDK_F,
GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE); gtk_widget_add_accelerator (clear, "activate", group,
GDK_C, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE); g_signal_connect (G_OBJECT (pulse), "activate",
G_CALLBACK (pulse_activated), progress); g_signal_connect (G_OBJECT (fill), "activate", G_CALLBACK
(fill_activated), progress); g_signal_connect (G_OBJECT (clear), "activate", G_CALLBACK (clear_activated),
progress); gtk_menu_shell_append (GTK_MENU_SHELL (menu), pulse); gtk_menu_shell_append
(GTK_MENU_SHELL (menu), separator); gtk_menu_shell_append (GTK_MENU_SHELL (menu), fill);
gtk_menu_shell_append (GTK_MENU_SHELL (menu), clear); gtk_menu_attach_to_widget (GTK_MENU
(menu), progress, NULL); gtk_widget_show_all (menu); }
GtkAccelGroup  gtk_accel_group_new()
GtkWidget  gtk_window_add_accel_group()
GtkAccelGroup  gtk_menu_set_accel_group()
GtkAccelMap  gtk_widget_add_accelerator()
void gtk_widget_add_accelerator (GtkWidget *widget, const gchar
*signal_name, GtkAccelGroup *group, guint accel_key, GdkModifierType mods, GtkAccelFlags flags);
GtkWidget  gtk_widget_add_accelerator()
signal_name  <gdk/gdkkeysyms.h>
<gtk/gtk.h>  GdkModifierType  GDK_SHIFT_LOCK,
GDK_CONTROL_MASK, GDK_MOD1_MASK,  Shift, Ctrl, Alt
1  GDK_1  GDK_KP_1
GtkWidget  GtkAccelFlags  GDK_ACCEL_VISIBLE,
GTK_ACCEL_LOCKED,  GTK_ACCEL_MASK
GtkWidget  GtkStatusbar  9-2
9-2.  GtkWidget  gtk_statusbar_new()
GtkWidget  gtk_status_bar_get_context_id()
guint gtk_status_bar_get_context_id (GtkStatusBar
*statusbar, const gchar *description);
IP  "URL"  "IP"
GtkWidget  gtk_status_bar_push()
GtkWidget  guint gtk_status_bar_push (GtkStatusBar
*statusbar, guint context_id, const gchar *message);
GtkWidget  gtk_status_bar_pop()
GtkWidget  context_id  (highest)
void gtk_status_bar_pop
(GtkStatusBar *statusbar, guint context_id);
GtkWidget  gtk_status_bar_remove()
GtkWidget  void gtk_status_bar_remove
(GtkStatusBar *statusbar, guint context_id, guint message_id);
GtkWidget  has-resize-grip

```

```

gtk_statusbar_set_has_resize_grip()
9-20
9-40
(statusbarhints.c) static void create_popup_menu (GtkWidget *menu, GtkWidget *progress, GtkWidget *statusbar)
{ GtkWidget *pulse, *fill, *clear, *separator; pulse = gtk_menu_item_new_with_label ("Pulse Progress"); fill =
gtk_menu_item_new_with_label ("Set as Complete"); clear = gtk_menu_item_new_with_label ("Clear Progress");
separator = gtk_separator_menu_item_new (); g_signal_connect (G_OBJECT (pulse), "activate", G_CALLBACK
(pulse_activated), progress); g_signal_connect (G_OBJECT (fill), "activate", G_CALLBACK (fill_activated),
progress); g_signal_connect (G_OBJECT (clear), "activate", G_CALLBACK (clear_activated), progress); /*
Connect signals to each menu item for status bar messages. */ g_signal_connect (G_OBJECT (pulse), "
enter-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_signal_connect (G_OBJECT (pulse), "
leave-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_signal_connect (G_OBJECT (fill), "
enter-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_signal_connect (G_OBJECT (fill), "
leave-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_signal_connect (G_OBJECT (clear), "
enter-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_signal_connect (G_OBJECT (clear), "
leave-notify-event", G_CALLBACK (statusbar_hint), statusbar); g_object_set_data (G_OBJECT (pulse), "
menuhint", (gpointer) "Pulse the progress bar one step."); g_object_set_data (G_OBJECT (fill), "menuhint",
(gpointer) "Set the progress bar to 100%."); g_object_set_data (G_OBJECT (clear), "menuhint", (gpointer) "Clear
the progress bar to 0%."); gtk_menu_shell_append (GTK_MENU_SHELL (menu), pulse); gtk_menu_shell_append
(GTK_MENU_SHELL (menu), separator); gtk_menu_shell_append (GTK_MENU_SHELL (menu), fill);
gtk_menu_shell_append (GTK_MENU_SHELL (menu), clear); gtk_menu_attach_to_widget (GTK_MENU
(menu), progress, NULL); gtk_widget_show_all (menu); } /* Add or remove a status bar menu hint, depending on
whether this function * is initialized by a proximity-in-event or proximity-out-event. */ static gboolean
statusbar_hint (GtkMenuItem *menuitem, GdkEventProximity *event, GtkStatusbar *statusbar) { gchar *hint; guint
id = gtk_statusbar_get_context_id (statusbar, "MenuItemHints"); if (event->type == GDK_ENTER_NOTIFY) {
hint = (gchar*) g_object_get_data (G_OBJECT (menuitem), "menuhint"); gtk_statusbar_push (statusbar, id, hint); }
else if (event->type == GDK_LEAVE_NOTIFY) gtk_statusbar_pop (statusbar, id); return FALSE; }
leave-notify-event
leave-notify-event
GDK_ENTER_NOTIFY GDK_LEAVE_NOTIFY
FALSE statusbar_hint()
9-40 "MenuItemHints"
guint id = gtk_statusbar_get_context_id (statusbar, "MenuItemHints");
GDK_ENTER_NOTIFY create_popup_menu() "menuhint"
gtk_statusbar_push (statusbar, id, hint); gtk_statusbar_push() "MenuItemHints"
GDK_LEAVE_NOTIFY gtk_statusbar_pop()
gtk_menu_item_set_submenu() gtk_menu_attach_to_widget()
GtkWidget *submenu); void gtk_menu_item_set_submenu (GtkMenuItem
*menuitem, GtkWidget *submenu);
GtkMenuItem
GtkImageMenuItem
gtk_image_menu_item_new() GtkImageMenuItem

```



```

(GTK_MENU_SHELL (editmenu), cut); gtk_menu_shell_append (GTK_MENU_SHELL (editmenu), copy);
gtk_menu_shell_append (GTK_MENU_SHELL (editmenu), paste); /* Create the Help menu content. */ contents =
gtk_image_menu_item_new_from_stock (GTK_STOCK_HELP, group); about =
gtk_image_menu_item_new_from_stock (GTK_STOCK_ABOUT, group); gtk_menu_shell_append
(GTK_MENU_SHELL (helpmenu), contents); gtk_menu_shell_append (GTK_MENU_SHELL (helpmenu), about);
gtk_container_add (GTK_CONTAINER (window), menubar); gtk_window_add_accel_group (GTK_WINDOW
(window), group); gtk_widget_show_all (window); gtk_main (); return 0; }
GtkMenuBar
gtk_menu_bar_new()
GtkPackDirection
GTK_PACK_DIRECTION_LTR,
GTK_PACK_DIRECTION_RTL, GTK_PACK_DIRECTION_TTB, GTK_PACK_DIRECTION_BTT
GtkMenuItem
GtkMenuBar
GtkMenuShell
gtk_menu_shell_append()
gtk_menu_shell_append (GTK_MENU_SHELL (menubar), file);
gtk_menu_shell_prepend()
gtk_menu_shell_insert()
GtkMenuItem
GTK+
gtk_menu_item_set_submenu()
(GtkMenuItem (file), filemenu);
GtkToolBar
select_all
(GtkEditable*); /* Create a toolbar with Cut, Copy, Paste and Select All toolbar items. */ static void create_toolbar
(GtkWidget *toolbar, GtkWidget *entry) { GtkToolItem *cut, *copy, *paste, *selectall, *separator; cut =
gtk_tool_button_new_from_stock (GTK_STOCK_CUT); copy = gtk_tool_button_new_from_stock
(GTK_STOCK_COPY); paste = gtk_tool_button_new_from_stock (GTK_STOCK_PASTE); selectall =
gtk_tool_button_new_from_stock (GTK_STOCK_SELECT_ALL); separator = gtk_separator_tool_item_new ();
gtk_toolbar_set_show_arrow (GTK_TOOLBAR (toolbar), TRUE); gtk_toolbar_set_style (GTK_TOOLBAR
(toolbar), GTK_TOOLBAR_BOTH); gtk_toolbar_insert (GTK_TOOLBAR (toolbar), cut, 0); gtk_toolbar_insert
(GTK_TOOLBAR (toolbar), copy, 1); gtk_toolbar_insert (GTK_TOOLBAR (toolbar), paste, 2); gtk_toolbar_insert
(GTK_TOOLBAR (toolbar), separator, 3); gtk_toolbar_insert (GTK_TOOLBAR (toolbar), selectall, 4);
g_signal_connect_swapped (G_OBJECT (cut), "clicked", G_CALLBACK (gtk_editable_cut_clipboard), entry);
g_signal_connect_swapped (G_OBJECT (copy), "clicked", G_CALLBACK (gtk_editable_copy_clipboard), entry);
g_signal_connect_swapped (G_OBJECT (paste), "clicked", G_CALLBACK (gtk_editable_paste_clipboard), entry);
g_signal_connect_swapped (G_OBJECT (selectall), "clicked", G_CALLBACK (select_all), entry); } /* Select all of
the text in the GtkEditable. */ static void select_all (GtkEditable *entry) { gtk_editable_select_region (entry, 0, -1);
}
gtk_toolbar_new()
create_toolbar()
GtkToolBar
GtkToolBar
gtk_toolbar_set_show_arrow()
TRUE
overflow
void gtk_toolbar_set_show_arrow (GtkToolBar *toolbar, gboolean show_arrow);
GtkToolBar
gtk_toolbar_set_style()
GtkToolBarStyle
GTK_TOOLBAR_ICONS:
GTK_TOOLBAR_TEXT:
GTK_TOOLBAR_BOTH:
GTK_TOOLBAR_BOTH_HORIZ:
important
TRUE
orientation)
gtk_toolbar_set_orientation()
GtkOrientation
GTK_ORIENTATION_HORIZONTAL
GTK_ORIENTATION_VERTICAL
GtkToolItem, GtkToolButton, GtkSeparatorToolItem
GtkToolItem
is-important

```



```

<menuitem name="EditPaste" action="Paste"/> <separator/> <menuitem name="EditSelectAll"
action="SelectAll"/> <menuitem name="EditDeselect" action="Deselect"/> </menu> <menu name="HelpMenu"
action="Help"> <menuitem name="HelpContents" action="Contents"/> <menuitem name="HelpAbout"
action="About"/> </menu> </menubar> </ui>
XML
<menubar>
  <menu>
    <menuitem name="FileOpen" action="Open"/>
    <menuitem name="FileSave" action="Save"/>
    <separator/>
    <menuitem name="EditCut" action="Cut"/>
    <menuitem name="EditCopy" action="Copy"/>
    <menuitem name="EditPaste" action="Paste"/>
    <separator/>
    <menuitem name="EditSelectAll" action="SelectAll"/>
    <menuitem name="EditDeselect" action="Deselect"/>
    <separator/>
    <menuitem name="HelpContents" action="Contents"/>
    <menuitem name="HelpAbout" action="About"/>
  </menu>
</menubar>
<toolbar name="Toolbar">
  <toolitem name="FileOpen" action="Open"/>
  <toolitem name="FileSave" action="Save"/>
  <separator/>
  <toolitem name="EditCut" action="Cut"/>
  <toolitem name="EditCopy" action="Copy"/>
  <toolitem name="EditPaste" action="Paste"/>
  <separator/>
  <toolitem name="EditSelectAll" action="SelectAll"/>
  <toolitem name="EditDeselect" action="Deselect"/>
  <separator/>
  <toolitem name="HelpContents" action="Contents"/>
  <toolitem name="HelpAbout" action="About"/>
</toolbar>
</ui>
<ui>
  <toolbar name="Toolbar">
    <toolitem name="FileOpen" action="Open"/>
    <toolitem name="FileSave" action="Save"/>
    <separator/>
    <toolitem name="EditCut" action="Cut"/>
    <toolitem name="EditCopy" action="Copy"/>
    <toolitem name="EditPaste" action="Paste"/>
    <separator/>
    <toolitem name="EditSelectAll" action="SelectAll"/>
    <toolitem name="EditDeselect" action="Deselect"/>
    <separator/>
    <toolitem name="HelpContents" action="Contents"/>
    <toolitem name="HelpAbout" action="About"/>
  </toolbar>
  <popup name="EntryPopup">
    <menuitem name="EditCut" action="Cut"/>
    <menuitem name="EditCopy" action="Copy"/>
    <menuitem name="EditPaste" action="Paste"/>
    <separator/>
    <menuitem name="EditSelectAll" action="SelectAll"/>
    <menuitem name="EditDeselect" action="Deselect"/>
  </popup>
</ui>
#include <gtk/gtk.h> /* All of the
menu item callback functions have a GtkMenuItem parameter, and * receive the same gpointer value. There is only
one callback function shown * since all of the rest will be formatted in the same manner. */
static void open(GtkMenuItem *menuitem, gpointer data);
#define NUM_ENTRIES 13
static GtkActionEntry entries[] = { { "File", NULL, "_File", NULL, NULL, NULL }, { " Open", GTK_STOCK_OPEN, NULL, NULL, "Open an existing
file", G_CALLBACK(open) }, { "Save", GTK_STOCK_SAVE, NULL, NULL, "Save the document to a file", G_CALLBACK(save) }, { "Quit", GTK_STOCK_QUIT, NULL, NULL, "Quit the application", G_CALLBACK(quit) }, { "Edit", NULL, "_Edit", NULL, NULL, NULL }, { "Cut", GTK_STOCK_CUT, NULL, NULL, "Cut the
selection to the clipboard", G_CALLBACK(cut) }, { "Copy", GTK_STOCK_COPY, NULL, NULL, "Copy the selection to the clipboard", G_CALLBACK(copy) }, { "Paste", GTK_STOCK_PASTE, NULL, NULL, "Paste text
from the clipboard", G_CALLBACK(paste) }, { "SelectAll", GTK_STOCK_SELECT_ALL, NULL, NULL, "Select all of the text", G_CALLBACK(selectall) }, { "Deselect", NULL, "_Deselect", "<control>d", "Deselect all
of the text", G_CALLBACK(deselect) }, { "Help", NULL, "_Help", NULL, NULL, NULL }, { "Contents", GTK_STOCK_HELP, NULL, NULL, "Get help on using the application", G_CALLBACK(help) }, { "About",
GTK_STOCK_ABOUT, NULL, NULL, "More information about the application", G_CALLBACK(about) } };
int main(int argc, char *argv[]) {
  GtkWidget *window, *menubar, *toolbar, *vbox;
  GtkActionGroup *group;
  GtkUIManager *uimanager;
  gtk_init(&argc, &argv);
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "UI Manager");
  gtk_widget_set_size_request(window, 250, -1); /* Create a new action group and add all of the actions to it. */
}

```

```

group = gtk_action_group_new ("MainActionGroup"); gtk_action_group_add_actions (group, entries,
NUM_ENTRIES, NULL); /* Create a new UI manager and build the menu bar and toolbar. */ uimanager =
gtk_ui_manager_new (); gtk_ui_manager_insert_action_group (uimanager, group, 0);
gtk_ui_manager_add_ui_from_file (uimanager, "menu.ui", NULL); gtk_ui_manager_add_ui_from_file (uimanager, "
toolbar.ui", NULL); /* Retrieve the necessary widgets and associate accelerators. */ menubar =
gtk_ui_manager_get_widget (uimanager, "/MenuBar"); toolbar = gtk_ui_manager_get_widget (uimanager, "
/Toolbar"); gtk_toolbar_set_style (GTK_TOOLBAR (toolbar), GTK_TOOLBAR_ICONS);
gtk_window_add_accel_group (GTK_WINDOW (window), gtk_ui_manager_get_accel_group (uimanager)); vbox
= gtk_vbox_new (FALSE, 0); gtk_box_pack_start_defaults (GTK_BOX (vbox), menubar);
gtk_box_pack_start_defaults (GTK_BOX (vbox), toolbar); gtk_container_add (GTK_CONTAINER (window),
vbox); gtk_widget_show_all (window); gtk_main (); return 0; }
// GtkUIManager
// GtkAction, GtkToggleAction, GtkRadioAction
// GtkActionEntry
typedef struct {
const gchar *name; const gchar *stock_id; const gchar *label; const gchar *accelerator; const gchar *tooltip;
GCallback callback; } GtkActionEntry;
// UI
// action
// NULL
// <Control>a", "<Shift><Control>x", "F3"
// Control "<Ctrl>" "<Ctl>"
// gtk_accelerator_parse()
// gtk_action_group_new()
// GtkActionGroup
// gtk_action_group_add_actions()
// GtkActionGroup
// void
gtk_action_group_add_actions (GtkActionGroup *group, const GtkActionEntry *entries, guint n_entries, gpointer
data);
// GtkAction
// gtk_action_group_add_action()
// gtk_action_group_add_action_with_accel()
// gtk_ui_manager_new()
// GtkUIManager
// UI
// GtkAction
// gtk_ui_manager_insert_action_group()
// GtkUIManager
// UI
// void
gtk_ui_manager_insert_action_group (GtkUIManager *uimanager, GtkActionGroup *group, gint pos);
// UI
// 9-11
// menu.ui
// toolbar.ui
// GError
// guint gtk_ui_manager_add_ui_from_file (GtkUIManager *uimanager, const gchar
*filename, GError **error);
// UI
// UI
// terminal
// UI
// "/MenuBar"
// "/Toolbar"
// GtkWidget*
gtk_ui_manager_get_widget (GtkUIManager *self, const gchar *path);
// <ui>
// name
// GTK_STOCK_OPEN
// "/MenuBar/FileMenu/FileOpen"
// GtkUIManager
// GtkToggleActionEntry
// GtkRadioActionEntry
// GtkToggleActionEntry
typedef struct { const gchar *name; const gchar *stock_id; const gchar *label; const gchar *accelerator; const gchar
*tooltip; GCallback callback; gboolean is_active; } GtkToggleActionEntry;
// UI
// (action definition)
// GtkActionEntry
// GTK+
// GtkToggleActionEntry
// member is_active
// (active)
// GtkToggleActionEntry
// gtk_action_group_add_toggle_actions()
// GtkToggleActionEntry
// void
gtk_action_group_add_toggle_actions (GtkActionGroup *group, const GtkToggleActionEntry *entries, guint
num_entries, gpointer data);
// GtkRadioActionEntry
// value
// member
// GtkRadioActionEntry
// gtk_action_group_add_radio_actions()

```

```

gtk_action_group_add_toggle_actions() void gtk_action_group_add_radio_actions (GtkActionGroup
*group, const GtkRadioActionEntry *entries, guint num_entries, gint value, GCallback on_change, gpointer data);
value UI GTK+ <placeholder> File gtk_ui_manager_add_ui() gtk_ui_manager_new_merge_id() void gtk_ui_manager_add_ui (GtkUIManager *uimanager, guint merge_id, const gchar *path, const gchar
*name, const gchar *action, GtkUIManagerItemType type, gboolean top); gtk_ui_manager_add_ui()
GTK_UI_MANAGER_MENU: GtkMenuItem GTK_UI_MANAGER_TOOLBAR: GtkMenuBar
GTK_UI_MANAGER_POPUP: GtkMenuBar
GTK_UI_MANAGER_MENUITEM: GtkMenuItem
GtkToolItem GtkToolbar
GTK_UI_MANAGER_SEPARATOR:
GTK_UI_MANAGER_ACCELERATOR: gtk_ui_manager_add_ui() Boolean TRUE
GtkIconSource,
GtkIconSet, GtkIconFactory
variant)
gtk_icon_set_new_from_pixbuf (GdkPixbuf *pixbuf);
GtkIconFactory (icon factory) GTK+ (global list)
calculator", "screenshot", "cpu", "desktop"
GtkIconFactory (iconfactory.c) #include <gtk/gtk.h> #define ICON_LOCATION "/path/to/icons/" typedef
struct { gchar *location; gchar *stock_id; gchar *label; } NewStockIcon; const NewStockIcon list[] = { {
ICON_LOCATION"checklist.png", "check-list", "Check _List" }, { ICON_LOCATION"calculator.png", "
calculator", "_ Calculator" }, { ICON_LOCATION"camera.png", "screenshot", "_ Screenshots" }, {
ICON_LOCATION"cpu.png", "cpu", "CPU _Info" }, { ICON_LOCATION"desktop.png", "desktop", "View
_Desktop" }, { NULL, NULL, NULL } }; static void add_stock_icon (GtkIconFactory*, gchar*, gchar*); int main
(int argc, char *argv[]) { GtkWidget *window, *toolbar; GtkIconFactory *factory; gint i = 0; gtk_init (&argc,
&argv); window = gtk_window_new (GTK_WINDOW_TOPLEVEL); gtk_window_set_title (GTK_WINDOW
(window), "Icon Factory"); gtk_container_set_border_width (GTK_CONTAINER (window), 10); factory =
gtk_icon_factory_new (); toolbar = gtk_toolbar_new (); /* Loop through the list of items and add new stock items.
*/ while (list[i].location != NULL) { GtkToolItem *item; add_stock_icon (factory, list[i].location, list[i].stock_id);
item = gtk_tool_button_new_from_stock (list[i].stock_id); gtk_tool_button_set_label (GTK_TOOL_BUTTON
(item), list[i].label); gtk_tool_button_set_use_underline (GTK_TOOL_BUTTON (item), TRUE); gtk_toolbar_insert
(GTK_TOOLBAR (toolbar), item, i); i++; } gtk_icon_factory_add_default (factory); gtk_toolbar_set_style
(GTK_TOOLBAR (toolbar), GTK_TOOLBAR_BOTH); gtk_toolbar_set_show_arrow (GTK_TOOLBAR
(toolbar), FALSE); gtk_container_add (GTK_CONTAINER (window), toolbar); gtk_widget_show_all (window);
gtk_main (); return 0; } /* Add a new stock icon from the given location and with the given stock id. */ static void
add_stock_icon (GtkIconFactory *factory, gchar *location, gchar *stock_id) { GtkIconSource *source; GtkIconSet
*set; source = gtk_icon_source_new (); set = gtk_icon_set_new (); gtk_icon_source_set_filename (source, location);

```



gtk\_icon\_set\_add\_source (set, source); gtk\_icon\_factory\_add (factory, stock\_id, set); }
gtk\_icon\_factory\_new(), gtk\_icon\_source\_new(), gtk\_icon\_set\_new()
9-12 gtk\_icon\_source\_set\_filename()
gtk\_icon\_source\_set\_pixbuf()
GtkPixbuf
gtk\_icon\_source\_set\_size()
void gtk\_icon\_source\_set\_size (GtkIconSource \*source, GtkIconSize size);
gtk\_icon\_source\_set\_size\_wildcarded() FALSE
gtk\_icon\_source\_set\_state\_wildcarded()
GtkIconState
gtk\_icon\_source\_set\_state()
void gtk\_icon\_source\_set\_state (GtkIconSource \*source, GtkIconState state);
GtkIconSet
void gtk\_icon\_set\_add\_source (GtkIconSet \*iconset, const GtkIconSource \*source)
GtkIconFactory
const gchar \*stock\_id, GtkIconSet \*iconset);
gtk\_icon\_factory\_add\_default()
GtkTextView
GtkButton
GtkUIManager
GtkIconFactory
GtkHandleBox
Text Editor
9-2
9-1
File
Quit
GtkStatusBar
GtkMenuShell
forward
backward
UI
Glade
XML
Libglad

Notes







000 00, 000 (address bar), 00 00 000 000. 00 000000 000 000 0000 00 000 000 00 00 0000 00 00 000 000 00 000000. 00 000 00 0000  
 0 **Handle Box** 000 0000 0 00 000 **GtkVBox** 00 0000 00. 00 00 00 0000 00 0000 00 0000 00 00. 0 000 000 0000 000 0000 0000 00 000 0000 00  
 0 00 000 000 0 00. 00000000 000 000000 00000000 000 00 000 0000 00. 00 000 0000 000 **Properties** 00 00 00 0000 **Edit** 0000 000 000 000. 0 000 0  
 0 00 00 **10-6** 00 00 0000 00 000. 0 00 0000 000 0000 000 0 0000 0000. 000 0 000 00 00 000 00 0000 000 0000 0000 00. 00 **10-6**. 00 000 00 0000  
 0 000 000000 00 000 000000 0000. 0000 000 000000 **Add** 0000 0000 00. 0000 0000 0000 0 000 000000 000 000 00 000 0000. 0000 00 000 0 000 0000 000 00  
 000 0. 000 0 000 0000 00 **Type** 00 0000 000 0000 000 000 0000. 00 000 000 00 000 000 0000 0 0000, 000, 00, 00, 000 00, 00 0 00, 0 00, 0000 0000.  
 00 0 000 0000 0 0000 000 00 000000 0000 000 00 0000. 00 00, 00 **10-6**000 0 000 000 **GtkMenuToolButton** 0 0000. 00 00 000 000 0000  
 00 0000 0 00 000 00 000 0000 000 0000. 000 0000 00 000 00 00 000 000, 000 00 0000 000000 000000 000 00 00. 00 0 000 0 000 000 000 0000 00  
 00. 0000 **Image Type** 00 0000 000 000 000 00 00 000, 000 00 000 00, 000 000 000 000 000 000 0000. 00 0000 000 0000 0000 0 0 0000 000000 000  
 00 00 00 000. **Glade** 000000 0000 00 000 000 000 0000 0000 000 000 000 0000. 0000 000 00 0000 000 00 00. **Libglade** 00 0 00 000 000000  
 0000 0000 00 0 0000 "User data" 000000 0000 000 00. 00 **10-6**000 `on_back_clicked()`00 000 00 000 00 000 **GtkToolButton** 00 `clicked` 0  
 000 0 0000. **Libglade** 000 000 0000000 000 00 **Glade** 000 000 00 000 000 000 000 00 000 0000 000 0 000 00. 000 00 000 0000 0000 00 00 000 00  
 000 00 000 0 00 0000 000000 0000 00. 000 **Libglade**000 0000 00 0000 000 000 000 **GModule** 00000000 00 0000 0000 0000 000 0000. 0 000 000000  
 0000 **Glade** 00 000 00 0000 000 000 0000 000000 00! 000 000 000 00 000 00 000 00 0000 000 **1/3**00 0000 000 000, 000000 000 0000 000 0000 0000.  
 00 **10-7**00 000 00 00 000 `expand` 000000 000 0000 00 000. 00 **10-7**. 00 00 0000 0 **3**000 `expand fill` 000000 **GtkBox** 0000 00 000000 00 00  
 0 00 000 0000 0 000 00 000000. **Glade** 000 000 0000 000 0000 0 0 000000 00 00 000 000000 000 000 0000. 000 000 00 00 00 00 000 000000 00!  
**Packing** 00 00 0 000 00 000 0 00 0 000 000000, 0 00 000 000 0000 0000 000 000 000 0000 00 000 0000. 000 000000 `gtk_box_pack_start()`  
`friends` 000 **GtkBox** 00 000 000 0 000 00 000 00(`settings`) 00 0000. 000 0000 00 000000 0000 00 00000000 00 **10-8** 000 00 000. 00 00 000000 0  
 0 000 0000 000 00 000 00 000 0000 0 00 0000 000 0000! 00 **10-8**. 00 00 0000 **10-8** 000 000 0000 0000 000000 00 0000 000000 0000 0 0000 0 00  
 00 0 000 0000. 00 000000 00, 00 00 000000, 00 00, 0 000000 00, 00 00 000 0000 0 0000 0000. 000 0 00 000 00000000 000 0000 0000 00 00 000 0000 00.  
 0 0 0000 00000000 00000 0000 00 00 000 000000 00 000 00000 0 000 000000 000 0000 0000 0000 0000 0000. 000 00 **10-9** 00 0 00 000 00 00 00 0000. 0 00 000  
**GtkEntry** 000 0000 000 0000 00, 00 000 0000 **GtkEntry** 00, 000 000 0 000 0000 000 0000. 00 **10-9**. 00 0000 000 000  
**GTK\_STOCK\_JUMP\_TO** 00 000 000 00 000 000 000000 0000 000 0000. 00 **Properties** 00000000 00 000 000000 0000 00. 000 0 0000 00 000 0  
 00 0000 000 000. 00 **10-9**000 0000 000 00 0 0 00 000 00 **GtkHBox** 00 0000 000000, **GtkImage** 000 **GTK\_STOCK\_JUMP\_TO** 00 000  
 0 0000, **GtkLabel** 000 "Go"00 000. 0000 0 0 0 00 0000 **Current Location** **GtkLabel** 000 000, 00 000(**bold**) 00 0000. 0 **2**000 **Pango**  
**Text Markup Language** 00 000 0 00. 00 **10-10**0000 000 0000 000000 "Use markup" 0 0000 00 000 **Pango Text Markup**  
**Language** 000 000 0 00 000. 00 000 0000 000 000 00 (markup tag) 0000 0000 0000 000. 00 **10-10**. **GtkLabel** 00 00000000 0000 0000 "Use underline" 00000  
 000000 000 000 0000 00 00. 00 **10-10**000 000 000 00 00 000 00 000 0000 000 000. 000 000 **GtkScrolledWindow** 00  
 0 00 000 000 00 0000, **GtkTreeView** 000 0 000000 000 0 000. 0000 00 0000 000 00000000 00 **10-11** 000000. 000 **Glade**00 00000000 000 00 000 00  
 0. 00 **10-11**. 000 00 000000 000000 000000 0000 0000 00 0 00 00 **GtkStatusBar** 000 00000000 000000! 000 000000 000 0000 00 00 000000 00 000 0  
 00 0000 000 0000 000 000 000 000. 00 00 0000 00 0 00 000 00 **GtkVBox** 000 0000 00 000 000 000 000. 00 000 00 00 000 00 000 0000.  
**Properties** 0000 **General** 00 "Number of items" 000000 000 0 00 000 0000 0 00. 00 0000 00 0 000 000 0 00 000 0 000 0 000 00 0000. 00 00  
 00 000 000 000000 00 00 00 000000 00 000 0000 00. 000 **Properties** 0 0 **Packing** 0 0000 00 000 00 0000 000 000 000 0 00. 00 000 00 0000 00 00  
 0 000 000 0000 00 000 0 00. 0000 000 00 0000 000 000000 000000 0000 000 0. 00 000 00 000 000 000000 000 000 0000 000 000 0 00. 00 00 0000 000000  
 00 0 0000 00 0 000 000 000000 000000 00000. 00 00 00 00 00 000 000 000 0 **Ctrl+X** 00 0000 00. 000 00 000 000 000 0 00 0 000 000 000. 0 00 0  
 000 00 00 0000 000 000 00 **Ctrl+V** 0000. **Glade 2** 00 0 00 00 000 000000 0000 00 0000 000 00 000 000000, 00 000000 000 0000 00 0000. 000  
 000 000 0 00 000 000 000 000 0000 000 000 00 000 00 000 000 000000 00. 00 **Glade 3**000 00000/0000 000000 000 0000 000 00. 00 000000000000  
 000 000 00 000 00 0000 000000 0000. 00 **10-12**000 **Go** 000 00 00 000000 **Signals** 00 000000. **GtkButton** 000 `clicked` 0000 00000, 0000 00 0  
**on\_button\_clicked()** 00000. 00 **10-12**. 00 000 000`clicked` 0000 000 00 00 0000 0000 00. 0000 000 0 0 000 **GtkToolButton**  
**clicked** 0000 000000 00. 0 **GtkEntry** 00 `activate` 0000 0000, 00 0000 0000 00 0 000 0 **Enter** 00 000 00000. 0 00000000 **Glade 3** 000 00000000 00  
 000 000 0000 00 0000 00 000 00 000000 0000 0000. 000 0000 0000 00 0000000000 000 000 0 **13**000 000 000. 00 00 0000 0000 `row-activated` 0000  
 0 0. 00 000000 000 00 00 000 000000 000000 0000 0000 0000 000 000. 000 000 0 0000 00 00 0 0000 0 **10-1**00 00000000 000 000 000 0 0 00 000. 00000000  
 00 00`GtkButtonGo` 00`clickedgo_on_clicked()` `GtkEntryLocation` 000 `activateon_location_activate()`  
**GtkMenuToolButton** 00`clickedon_back_clicked()` `GtkMenuToolButton` 000 `clickedon_forward_clicked()`  
**GtkToolButton** 00 `clickedon_up_clicked` `GtkToolButton` 0000 (refresh)`clickedon_refresh_clicked()` `GtkToolButton`  
**clickedon\_home\_clicked() `GtkToolButton` 00 `clickedon_delete_clicked()` `GtkToolButton` 00`clickedon_info_clicked()`  
**GtkTreeView** 00 0000 `row-activatedon_raw_activated` `GtkWindow` 00 `destroygtk_main_quit()` 0 **10-1**. 00 000 00 000000 00**

Glade 3 00 00 00 00 00. 00 10-13 00 0 0000 00 0000 00 0000 000 0000. 0000 00 000 000 000, 00 00, 000 00, 0000 0000 000 0000. 00 10-13. 00 0 000 000 0000 00 0 000 0000 0 000 00 00 000 000000 00 000 0000. 0 0000 0-000 00 0000 000 000000 000. 00 00 000 0000 0000 0 000 00 0 0000 000 000000. 0 000 00 0000 00. 0000 UI 000 0 0 GtkUIManager 000 000 0000 0000 000 000000. 0 000 00 000 00 00 000 000 0 00 000 000 00 000 000000. 00 000 000 UI 000000 0000 000 0000 000 000 0000 000 0 000 0000. 00000000 000 0 Glade 0000 00 00 000000 00 00 000000. Glade 00 0000 00 00 000 000000, 000 0000 0 000 00 0000 0 000. Libglade 000 000 000000 000 000 0000 00 000 00 00 000000 00. 000 0 0000 0 00 000 000, UI 000 000 00 0 0000 000 000 000000 000000 0000 00 00 000 000. 00 000 0 00 000 00 000, 00 00 00 00 00 0000 0000 000000 0000 0 00 0000 000 00 0000 0000. 000 00 000000 000 0 gtk\_box\_pack\_start() 000 GtkUIManager 000 000 000 000 000 0 00. 0000 0000 000 000000 000000 0000 000 000 Glade 0 0 0 00 000 0000 0000. 00 000 000000 000 0000 project.glade 000 000 0000, 000 project 000 0000 0000 00. 000 00000000 000 0000 00 00 00 00 000 000 0000. Libglade 0000 Glade 00 00000000 0000 000 Libglade 000 000 0000000 0000 00. 0 000000 Glade 000 0000000 0000 000 0 000 00 000 0000 00 000. Libglade 0 XML 000000 0000 000 000000 00 0 0000 0 0000 GladeXML 000 0000. 00 000 Glade 000 000 0000 0 000 0000 000 000 000. Libglade 0 00 000 000 000 000000 0000 000, 00 0000 00 000 000000 000 0000 00 00 00. 0000 0000 000 0000 000 000 0 00. 00 00, GladeXML 0000 000 000 0000 000 0000 000000 0000 000 000 000 00 000. Libglade 00 00 0000 0 000 0 0000 Glade 3 00 00 000 0000 00 0000 000 000 0 00. 00 000 10-100 00 00 0000 00(hand-code)000 000 000 0000 000 000. 00 000 000000 0000 000 0 000000 000 000 000 0000 0 13000 000 000 0000. 000 10-1. 000 000000 0000 (browser.c) #include <gtk/gtk.h> #include <glade/glade.h> void on\_back\_clicked (GtkToolButton\*); void on\_forward\_clicked (GtkToolButton\*); void on\_up\_clicked (GtkToolButton\*); void on\_refresh\_clicked (GtkToolButton\*); void on\_delete\_clicked (GtkToolButton\*); void on\_home\_clicked (GtkToolButton\*); void on\_info\_clicked (GtkToolButton\*); void on\_go\_clicked (GtkButton\*); void on\_location\_activate (GtkEntry\*); void on\_row\_activated (GtkTreeView\*, GtkTreePath\*, GtkTreeViewColumn\*); int main (int argc, char \*argv[]) { GtkWidget \*window; GladeXML \*xml; gtk\_init (&argc, &argv); xml = glade\_xml\_new ("browser.glade", NULL, NULL); window = glade\_xml\_get\_widget (xml, "window"); glade\_xml\_signal\_autoconnect (xml); gtk\_widget\_show\_all (window); gtk\_main (); return 0; }

000 10-10 000 000 GTK+ 00 000 0000 0000 0000 000 000. 000 00 000 000 Libglade 0 0000 0 000. gcc -export-dynamic browser.c -o browser `pkg-config --cflags --libs gtk+-2.0` \ `pkg-config --cflags --libs libglade-2.0` 00 00 0000 00 000 000000 0000 0000 0000 000 000 -export-dynamic 0000 0 000. 000 000 GModule 00 00(introspection) 00 00000000 0 00 000 0 0 00 000 000 00 0000 0000 00 000! 000 000000 0000 Glade 000 000000 glade\_xml\_new() 000 0000. 00 0000 0000 0 0 00 GladeXML 0000, gtk\_init() 000 000000 00. 0 000 XML 000 0000 000 0000000 0000, 00 0 000 00 0000, 000 0000 000 0000. GladeXML\* glade\_xml\_new (const char \*glade\_file\_name, const char \*root\_widget, const char \*translation\_domain); 00 000 0 00 0 000 0000. 0 000 Glade 000 000000 000 000 0000. 00 000 00 000 00 000000 00 000 0000 000 00 0000 000 000 0000 0000 000 000 000000 0000 00 000 000000 00. 00 000 00 000 000 GNU Autotools 00 00000000 00 0000. 00 000 00 00 00000 0000 000000, 000 0 00 000 000 000000 g\_chdir() 000 000 0000. glade\_xml\_new() 0000 0 00 000000 00 000 000000. 0000 GladeXML 000 00 000 0 00 000 0 0 000 0000 00 00. 0 000000 NULL 0000 GladeXML 00 0 00 000 000 000. glade\_xml\_new() 000 000 000 0000000 000 000 000 000. 0 0 00 00 000000 000000000 000 000000 00 000 000 00 UI 000 00 00 0000 00. 000000 0000 000 00(translation) 0000 000 0 000, 00 00 0 000 0000 0 00 000 00 000 000. 00000000 000 0000 000 00 000000 NULL 0000 00 000000. 000 GladeXML 000 0000000 000 0000000 0000000 00 glade\_xml\_get\_widget() 000 000 0000 00 0000. 0 00 0 00 0000000 000 000000, 00 Glade 00 0000 000 0000 0000. GtkWidget\* glade\_xml\_get\_widget (GladeXML \*xml, const char \*name); glade\_xml\_get\_widget() 000 000 0000 0000 Glade 000 000 0000 000 000 0000 00. 0 000 GTK+ 000 000 000 000000000 000 00 00 GtkWidget 00 00000 000 0000. 000 00 Libglade 00 00 0 000 000000, 000 000 0 0 0 0000 00 00 0000 000 0000 0000 000 00000000 0000 0000 0000 000 0 000 00 0000. 0 00 000 000 glade\_xml\_get\_widget\_prefix() 000, 00 0 00 0000 000 0000 00 000 000000 0000. 000 0000 000 00 00 00 0 00 00000 0 000 00 0000 0000. GList\* glade\_xml\_get\_widget\_prefix (GladeXML \*xml, const char \*name); 000 000000000000 000 000 000 00 00 000 Glade 000 000 000 0000 0000. 000 10-1000 glade\_xml\_signal\_autoconnect() 000 0 00 00 0000 000000. void glade\_xml\_signal\_autoconnect (GladeXML \*xml); 0000 00 00 0000 00 Libglade 0 GModule 00 NULL 000 000, 00 00000000 00 00 0(symbol table) 0000 000 000. 000 000 000 0000000 0 000 000 00 000, 0 Glade 00 0 000 0000 00000000 000 00 000 000 0000 000 000. 0 000 0000 000000 GModule 000 000 00 0 000. 0000 0000 0 00 000 glade\_xml\_signal\_autoconnect\_full() 0000 000. 0 000 0000 00 00 0000 0 00 000000 00 000 000000 000. 00 00 00 00 000 000000 0000 0000 0 00 0 00. void glade\_xml\_signal\_connect\_full (GladeXML \*self, const gchar \*handler\_name, GladeXMLConnectFunc connect\_func, gpointer data); 000 10-20 glade\_xml\_signal\_autoconnect() 0000 000 0000 GModule 0000 000 GladeXMLConnectFunc 0000. 0 000 Glade 000 000000 00 0 0 000000 0000. 000 10-2. 000 000000 static void

```

connect_func (const gchar *callback_name, GObject *object, const gchar *signal_name, const gchar *signal_data,
GObject *connect_object, gboolean connect_after, gpointer data) { static GModule *module_self = NULL; gpointer
handler_func; module_self = g_module_open (NULL, 0); g_assert (module_self != NULL); if (g_module_symbol
(module_self, callback_name, &handler_func)) { if (connect_object && connect_after) g_signal_connect_object
(object, signal_name, handler_func, connect_object, G_CONNECT_AFTER); else if (connect_object && !
connect_after) g_signal_connect_object (object, signal_name, handler_func, connect_object,
G_CONNECT_SWAPPED); else if (!connect_object && connect_after) g_signal_connect_after (object,
signal_name, handler_func, data); else g_signal_connect (object, signal_name, handler_func, data); } else
g_warning ("The callback function could not be found: %s", callback_name); }

```

10-20

g\_signal\_connect()

g\_signal\_connect\_after()

gulong g\_signal\_connect\_after (gpointer object, const gchar \*signal\_name, GCallback handler, gpointer data);

g\_signal\_connect\_object()

GObject

object

gobject

gulong g\_signal\_connect\_object (gpointer object, const gchar \*signal\_name, GCallback handler, gpointer gobject, GConnectFlags flags);

g\_signal\_connect\_object()

GConnectFlags

G\_CONNECT\_AFTER:

g\_signal\_connect\_after()

G\_CONNECT\_SWAPPED:

g\_signal\_connect\_swapped()

G\_CONNECT\_SWAPPED

GTK\_STOCK\_QUIT

gtk\_widget\_destroy()

glade\_xml\_signal\_connect()

Glade

Glade

glade\_xml\_signal\_connect()

void glade\_xml\_signal\_connect (GladeXML \*xml, const char \*signal\_name, GCallback callback\_func);

glade\_xml\_signal\_connect\_data()

void glade\_xml\_signal\_connect\_data (GladeXML \*xml, const char \*signal\_name, GCallback callback\_func, gpointer data);

Libglade

GModule

GTK+

Glade

Libglade

backend code

10-1. Glade

9-10

Glade

9-10

www.gtkbook.com

Glade 3

9-2

Glade

9-2

http://www.gtkbook.com

Glade 3

GTK+ + Glade

Glade

XML

Glade 3

Libglade

Glade

GObject

Notes

# FoundationsofGTKDevelopment:Chapter 11

## 제 11 장 커스텀 위젯 생성하기

### 커스텀 위젯 생성하기

지금까지 GTK+와 그것이 지원하는 라이브러리에 대해 많은 내용을 배웠다. 이제 GTK+가 제공하는 위젯을 이용해 자신만의 복잡한 애플리케이션을 생성하는 방법에 대한 지식은 충분히 습득하였을 것이다.

하지만 아직 학습하지 않은 주제가 하나 있는데, 바로 자신만의 위젯을 생성하는 방법이다. 따라서 이번 장에서는 GObject에서 새로운 클래스를 상속하는 데에 주력하겠다. 세 가지 예를 통해 안내하겠다.

첫 번째 예는 GtkEntry 위젯에서 MyIPAddress라고 불리는 새로운 위젯을 상속한다. 이 위젯은 사용자가 IP 주소를 입력하여 그에 따라 커서의 위치를 제어하도록 해준다. 두 번째 예제에서는 MyMarquee라고 불리는 새로운 커스텀 GtkWidget 클래스를 생성할 것인데, 이는 메시지를 명시된 속도로 스크롤 시 이용된다. 마지막으로 커스텀 인터페이스를 구현하고 이용하는 방법을 학습할 것이다.

이번 장에서 배울 내용은 다음과 같다.

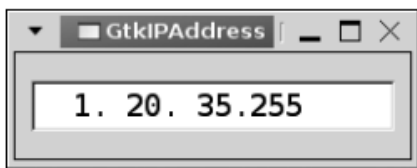
- 이미 존재하는 클래스와 위젯에서 새로운 클래스 및 위젯을 상속하는 방법
- GtkWidget에서 상속된 커스텀 위젯을 생성하는 방법. 이는 개발자 스스로 화면에 위젯을 노출시키고 그리도록 요한다.
- 커스텀 인터페이스를 구현하고 사용하는 방법.

### 새로운 위젯 상속하기

이번 장은 이미 존재하는 타입에서 새로운 GObject 타입을 생성하는 방법을 가르치는 데에 목표를 둔다. 새로운 객체의 상속 방법을 가장 잘 학습하는 방법은 예를 이용하는 것이다. 따라서 이번 절에서는 사용자가 IP 주소를 입력할 수 있는 MyIPAddress라고 불리는 새로운 위젯을 생성할 것이다. 이 위젯은 사용자가 유효한 IP 주소 외에는 입력하지 못하도록 제한한다.

**Note** 이번 절에서 새로운 위젯을 상속하면서 사용한 방식은 GObject에서 상속된 모든 객체에 적용된다. 따라서 새로운 위젯의 상속으로 국한되지 않고 GObject에서 직접적으로 혹은 간접적으로 상속된 타입에서 새로운 타입을 상속할 수도 있다.

이번 절에서 생성한 GtkIPAddress 위젯의 스크린샷을 그림 11-1에 실었다. 한 자리수와 두 자리수 모두 우측으로 정렬되어 있다.



### MyIPAddress 헤더 파일 생성하기

어떤 타입의 GObject를 상속하든 가장 먼저 할 일은 헤더 파일을 생성하는 일이다. 이 파일은 각 객체에서 필요로 하는 기본 함수의 호출을 개발자가 준비하도록 해준다. 헤더 파일은 개발자의 새로운 객체를 이용하는 코드라면 어디서든 이용 가능한 public 함수를 포함하기 때문에 위젯을 계획 시 헤더 파일의 생성부터 시작하면 되겠다.

C++ 컴파일러를 제공하려면 G\_BEGIN\_DECLS와 G\_END\_DECLS를 이용해 헤더 파일의 내용에 괄호를 둘러야 한다. 이 두 개의 매크로는 파일 내용 주변에 extern "C"를 추가하여 C에서와 마찬가지로 컴파일 시 모든 함수들이 식별명(symbol name)을 이용하도록 강요한다. 헤더 파일의 셀에 대한 예제는 리스팅 11-1에서 찾을 수 있다.

리스팅 11-1. MyIPAddress 헤더 파일 (myipaddress.h)

```

#ifndef __MY_IP_ADDRESS_H__
#define __MY_IP_ADDRESS_H__

#include <glib.h>
#include <glib-object.h>
#include <gtk/gtkentry.h>

G_BEGIN_DECLS
...
G_END_DECLS

#endif /* __MY_IP_ADDRESS_H__ */

```

■ **Note** 이번 책에 실린 대부분의 예제와 달리 이번 장에 실린 예제는 여러 줄로 이어지기 때문에 전체를 표시하지 않고 작게 나누어 표시하였다. 소스 코드의 전체 파일은 본 서적의 웹 사이트 [www.gtkbook.com](http://www.gtkbook.com)에서 다운로드할 수 있다.

개발자는 필요한 구조체와 함수를 비롯해 새로운 위젯을 생성할 때마다 리스팅 11-2에서 보이는 바와 같이 다섯 개의 매크로를 정의할 필요가 있다. 헤더 파일의 나머지 부분에 포함된 모든 함수와 구조체 선언은 `G_BEGIN_DECLS`와 `G_END_DECLS` 사이에 위치해야 한다. 리스팅 11-2의 매크로는 모든 GObject가 사용하는 표준 명명 규칙(standard naming scheme)을 따른다. 표준 명명 규칙은 객체 상속을 훨씬 간단하게 만든다.

리스팅 11-2. GObject 지시어

```

#define MY_IP_ADDRESS_TYPE (my_ip_address_get_type ())
#define MY_IP_ADDRESS(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
    MY_IP_ADDRESS_TYPE, MyIPAddress))
#define MY_IP_ADDRESS_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST ((klass), \
    MY_IP_ADDRESS_TYPE, MyIPAddressClass))
#define IS_MY_IP_ADDRESS(obj) (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
    MY_IP_ADDRESS_TYPE))
#define IS_MY_IP_ADDRESS_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE \
    ((klass), \
    MY_IP_ADDRESS_TYPE))

```

새로운 객체가 사용하는 모든 함수의 앞에는 `GTK_` 또는 `GDK_` 처럼 라이브러리명이 붙은 다음에 위젯명이 따라와야 한다. 가장 먼저 정의해야 하는 매크로는 `MY_IP_ADDRESS_TYPE`이다. 이 매크로는 객체에 해당하는 `GType` 구조체를 리턴한다. `my_ip_address_get_type()`은 헤더 파일에서 후에 정의하도록 하겠다.

다음 매크로인 `MY_IP_ADDRESS()`는 객체를 `MyIPAddress`로 캐스팅하는 데에 사용된다. 이는 `GTK_WIDGET()`, `GTK_ENTRY()`, 또는 `G_OBJECT()` 처럼 함수를 이용해 객체를 캐스팅하는 것과 비슷하다. `G_TYPE_CHECK_INSTANCE_CAST()`는 두 가지 작업을 수행한다. 우선 객체가 올바른 타입인지 검사한다. 올바른 타입이 아니면 경고가 발생한다. 올바른 타입이 맞다면 객체는 `MyIPAddress` 위젯으로 캐스팅되어 리턴된다.

`MY_IP_ADDRESS_CLASS()`는 `MY_IP_ADDRESS()`와 동일한 방식으로 사용되는데, 객체를 `MyIPAddressClass`로 캐스팅한다는 점만 다르다. 이 두 타입의 차이점은 머지않아 설명하겠다.

■ **Note** 코드는 C++ 컴파일러를 이용해서도 컴파일될 수도 있으므로 위젯 클래스 타입을 참조하려면 항상 `class` 대신 `klass`를 사용해야 하는데, `class`는 C++ 키워드에 해당하기 때문이다.

마지막으로 소개할 두 가지 매크로인 `IS_MY_IP_ADDRESS()`와 `IS_MY_IP_ADDRESS_CLASS()`는 객체가 올바른 타입인지 확인하는 데에 사용된다. 객체가 명시된 타입이면 각 함수는 `TRUE`를 리턴할 것이다.



그 다음 단계는 MyIPAddress와 MyIPAddressClass 구조체를 정의하는 것으로, 리스팅 11-3에 내용을 싣겠다. 위젯 내용은 \_MyIPAddress에서 보유한다. 새로운 위젯의 구조체에서 첫 번째 member는 항상 상속받고자 하는 타입의 인스턴스여야 한다. 리스팅에서 볼 수 있듯이 GtkEntry로부터 MyIPAddress 위젯이 상속될 것이다. 리스팅 11-3. MyIPAddress 구조체

```
typedef struct _MyIPAddress MyIPAddress;
typedef struct _MyIPAddressClass MyIPAddressClass;

struct _MyIPAddress
{
    GtkEntry entry;
};

struct _MyIPAddressClass
{
    GtkEntryClass parent_class;

    void (* ip_changed) (MyIPAddress *ipaddress);
};
```

**Caution** MyIPAddress에서 부모 객체 GtkEntry를 포인터로 정의해선 안 된다! 이를 어길 시 새로운 위젯 클래스가 부모 위젯 클래스보다 작다는 오류가 발생하고 컴파일은 실패할 것이다.

MyIPAddress의 GtkEntry 자식은 대부분 위젯의 경우와 마찬가지로 포인터가 아님을 명심하라. 이는 상속된 객체는 모든 방면에서 그것의 부모 구조체라는 사실을 증명한다. 시그널, 프로퍼티, 스타일뿐 아니라 전체 객체 자체도 상속한다. 이러한 관계는 MyIPAddressClass 구조체에서 포인터가 아닌 GtkEntryClass 객체의 선언으로 확인할 수 있다.

MyIPAddress 구조체는 GtkEntry 객체 외에는 어떤 객체도 보유하지 않는다. 위젯 구조체 GtkEntry는 본래 프로그래머가 접근해선 안 되는 private 객체를 보유하는 데에 사용되었지만 GObject는 private 프로퍼티를 구현할 수 있는 더 나은 방법을 제공하는데, 이는 소스 파일에서 다루도록 하겠다. 개발자는 위젯 구조체내에서 꼭 필요하다고 간주되는 변수를 자유롭게 배치할 수 있지만 이를 실행하기 전에는 개발자가 객체로 직접 접근성을 가져야 하는지 고려해야 한다. 기본적으로 위젯이 변수의 변경에 반응해야 할 경우 소스 파일 내 private 클래스에서 선언되어야 한다. 반응하지 않아도 된다면 public 위젯 구조체 내에 위치하는 수도 있다.

MyIPAddress 외에도 부모 클래스 타입 GtkEntryClass의 인스턴스를 포함하는 MyIPAddressClass를 정의해야 한다. 다시 말하지만 이것은 포인터 타입이 되어선 안 되는데, 부모 프로퍼티, 시그널, 함수가 개발자의 위젯 클래스로부터 상속되는 것을 허용하기 때문이다.

그 외에도 시그널에 대한 콜백 함수 프로토타입을 위젯 클래스에 정의해야 한다. 이번 예제에서는 ip-changed 시그널이 추가될 것인데, 이 시그널은 IP 주소가 성공적으로 변경되면 호출된다. 이 시그널 덕분에 개발자는 네 개의 위젯 프로퍼티에서 내용이 변경되는지 감시하지 않아도 되는데, 네 개의 숫자 각각이 자체의 위젯 프로퍼티로 정의될 것이기 때문이다.

헤더 파일을 생성하는 과정에서 마지막 단계는 개발자가 호출할 수 있는 함수에 대한 함수 프로토타입을 정의하는 일로, 리스팅 11-4에 소개하겠다. myipaddress.c 파일에서만 접근할 수 있는 private 함수들도 정의하겠다.

리스팅 11-4. 헤더 파일 함수 프로토타입

```
GType my_ip_address_get_type (void) G_GNUC_CONST;
GtkWidget* my_ip_address_new (void);
```

```
gchar* my_ip_address_get_address (MyIPAddress *ipaddress);
void my_ip_address_set_address (MyIPAddress *ipaddress, gint
address[4]);
```

리스트 11-4에 소개된 네 가지 함수는 각각 MyIPAddress와 연관된 GType을 리턴하고, 새로운 MyIPAddress 위젯을 생성하며, IP 주소를 문자열로 리턴하고, 새로운 IP 주소값을 제공한다. 이와 관련된 내용은 본 장의 뒷 부분에서 함수의 구현을 다루면서 좀 더 논하겠다.

#### 소스 파일 생성하기

헤더 파일이 완료되었다면 새로운 객체를 상속하고 MyIPAddress의 기능을 구현할 때가 되었다. 위젯에는 추적해야 하는 프로퍼티와 시그널이 많을 것인데, 이를 아래 리스트에서 정의할 것이다.

리스트 11-5는 MyIPAddress 소스 파일을 걸쳐 필요로 하는 값과 구조체를 다수 정의한다. 시그널과 프로퍼티 식별자를 비롯해 private 클래스들도 이에 포함된다.

리스트 11-5. 전역적 열거와 구조체(myipaddress.c)

```
#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
#include <stdlib.h>
#include <math.h>
#include "myipaddress.h"

#define MY_IP_ADDRESS_GET_PRIVATE(obj) (G_TYPE_INSTANCE_GET_PRIVATE
((obj), \
    MY_IP_ADDRESS_TYPE, MyIPAddressPrivate))

typedef struct _MyIPAddressPrivate MyIPAddressPrivate;

struct _MyIPAddressPrivate
{
    guint address[4];
};

enum
{
    CHANGED_SIGNAL,
    LAST_SIGNAL
};

enum
{
    PROP_0,
    PROP_IP1,
    PROP_IP2,
    PROP_IP3,
    PROP_IP4
};

static guint my_ip_address_signals[LAST_SIGNAL] = { 0 };
```

리스팅 11-5에 정의된 매크로 MY\_IP\_ADDRESS\_GET\_PRIVATE()은 현재 객체 인스턴스와 연관된 MyIPAddressPrivate 구조체를 검색한다. 이 구조체는 객체의 private 프로퍼티를 보유하는 데 사용되는데, 이러한 프로퍼티들은 각 인스턴스에 유일하다. 이번 예제에서는 MyIPAddress 가 네 개의 IP 주소값을 각각 보유한다. 이러한 값은 private하게 유지되어 해당 파일에서 정의된 함수만 수정이 가능한데, 값이 변경되면 위젯이 업데이트 되어야 하기 때문이다.

다음 단계는 위젯에 설치될 시그널과 프로퍼티를 참조하는 데에 사용될 열거를 정의하는 일이다. CHANGED\_SIGNAL은 사용자가 IP 주소의 내용을 변경하거나 프로그램적으로 변경되었을 때 발생하게 될 ip-changed 시그널을 참조한다. LAST\_SIGNAL은 얼마나 많은 시그널이 위젯에 설치되고 my\_ip\_address\_signals[]에 보관되어 있는지 알아내는 데에 사용된다. 이를 마지막 열거 값으로 정의하면 향후 시그널 계수를 업데이트해야 한다는 걱정 없이 쉽게 시그널을 추가할 수 있다.

또 다른 열거는 프로퍼티 식별자를 보유한다. 개발자의 모든 프로퍼티 식별자는 선언 시 0보다 커야 하므로 PROP\_0의 초기 열거 값을 위치시키는 것이 관행이다. 다른 열거 값들은 IP 주소를 구성하는 네 개의 정수를 참조한다. 이는 시그널을 위젯 클래스로 추가 시 사용된다. 새로운 위젯을 이용하는 프로그래머는 후에 개발자가 정의하는 프로퍼티명을 이용할 수 있다.

### 새로운 GType 등록하기

헤더 파일에서 우리는 리스팅 11-6에 구현된 my\_ip\_address\_get\_type() 함수에 대한 함수 프로토타입을 정의 하였다. 이 함수는 GType 값을 리턴하는데, 이는 등록된 타입에 유일한 수치값에 불과하다. 이런 경우 등록된 타입은 MyIPAddress 객체가 된다.

리스팅 11-6. 새로운 MyIPAddress 타입 생성하기

```
GType
my_ip_address_get_type (void)
{
    static GType entry_type = 0;

    if (!entry_type)
    {
        static const GTypeInfo entry_info =
        {
            sizeof (MyIPAddressClass),
            NULL,
            NULL,
            (GClassInitFunc) my_ip_address_class_init,
            NULL,
            NULL,
            sizeof (MyIPAddress),
            0,
            (GInstanceInitFunc) my_ip_address_init,
        };

        entry_type = g_type_register_static (GTK_TYPE_ENTRY,
        "MyIPAddress",
            &entry_info, 0);
    }

    return entry_type;
}
```

```
}

```

타입이 아직 생성되지 않았다면 객체가 초기화되는 동안 정적 식별자(static identifier)가 설정되지 않았으니 우리가 추가해야 한다는 뜻이다. 새로운 GType을 등록할 때는 먼저 타입에 대한 GTypeInfo 객체를 선언해야 한다. GTypeInfo 구조체에는 표 11-1에 정의된 10개의 member가 있는데, 모든 member가 필요한 것은 아니다.

변수	설명
guint16 class_size	위젯을 생성 시 필요한 클래스 구조체의 크기. MyIPAddressClass 구조체의 크기일 뿐이다.
GBaseInitFunc base_init	기본(base) 초기화 함수의 선택적 위치. 이 콜백 함수는 부모 클래스에서 복사한 모든 동적인 클래스 member를 재할당하는 데 사용된다.
GBaseFinalizeFunc base_finalize	기본 최종화(finalization) 함수의 선택적 위치. 이 콜백 함수는 GBaseInitFunc 함수에서 실행한 내용을 최종화하는 데 사용된다.
GClassInitFunc class_init	클래스 초기화 함수의 선택적 구현으로, 클래스에 대한 가상 함수를 채우고 시그널 및 객체 프로퍼티를 등록하는 데 사용된다.
GClassFinalizeFunc class_finalize	클래스 최종화 함수의 선택적 구현. 동적으로 할당된 자원은 GBaseInitFunc와 GBaseFinalizeFunc 함수에서 처리되어야 하므로 이 함수가 필요한 경우는 극히 드물다.
gconstpointer class_data	GClassInitFunc와 GClassFinalizeFunc의 구현으로 전달될 포인터 데이터.
guint16 instance_size	상속 중인 객체 또는 위젯의 크기. MyIPAddress 구조체의 크기일 뿐이다.
guint16 n_preallocs	GLib 2.10 배포판 이후부터 이 member는 무시되는데, 슬라이스 할당자를 이용해 인스턴스의 메모리 할당이 처리되기 때문이다.
GInstanceInitFunc instance_init	인스턴스를 준비하는 데 사용되는 선택적 함수. MyIPAddress 예제에서 이 함수는 key-press-event와 changed 시그널을 각 GtkEntry로 연결하고 위젯을 패키징한다.
const GTypeValueTable *value_table	이 타입에 대한 일반 GValue 객체를 처리하는 함수 테이블. 기본 타입을 생성할 때에만 주로 사용되므로 대부분의 경우는 정의하지 않아도 된다.

표 11-1. GTypeInfo members

새로운 클래스를 초기화하는 단계는 네 가지로 나뉘는데, 부모 클래스로부터 member를 복사하고, 나머지 member를 0으로 초기화하며, GBaseInitFunc 초기화기(initializer)를 호출하고, GClassInitFunc initializer를 호출하는 것에 해당한다. 이 단계들은 객체의 새로운 인스턴스들이 인스턴스화될 때마다 실행해야 한다.

GClassInitFunc는 새로운 GType이 유효해지기 위해 GTypeInfo에서 필요로 하는 함수다.

자신의 새 타입에 대해 GTypeInfo 객체를 준비했다면 다음으로 g\_type\_register\_static()을 이용해 GType을 등록할 차례다. 이 함수의 첫 번째 매개변수는 부모 타입을 참조하는 GType 값이다. 가령 MyIPAddress로부터 객체를 상속한다면 이는 GTK\_TYPE\_IP\_ADDRESS로 설정될 것이다. GtkEntry 위젯으로부터 새 객체를 상속하려면 GTK\_TYPE\_ENTRY를 이용할 수 있다.

```
GType g_type_register_static (GType parent_type,
    const gchar *type_name,
    const GTypeInfo *info,
    GTypeFlags flags);
```

다음으로는 새 타입의 이름으로 사용될 문자열과 그에 상응하는 GTypeInfo 객체를 명시해야 한다. 우리 예제에서는 위젯의 이름인 MyIPAddress 문자열이 사용되었다. 객체에 유일해야 하기 때문에 위젯의 이름을 사용하는 것은 좋은 생각이다. 이름은 알파벳 문자로 시작해 최소 3개의 문자로 구성되어야 한다.

마지막 매개변수는 GTypeFlags의 bitwise 조합이다. 이 열거체에서 정의하는 값에는 두 가지가 있다.

G\_TYPE\_FLAG\_ABSTRACT는 타입이 추상적임을 나타낸다. 따라서 추상적 타입의 인스턴스는 생성하지 못하도록 막을 것이다. 나머지 플래그 G\_TYPE\_FLAG\_VALUE\_ABSTRACT는 값 테이블과 같은 추상적 값의 타입을 나타내지만 g\_value\_init()와 함께 사용할 수 없다. 함수는 주어진 매개변수에 대해 새로운 GType을 리턴한다.



```

        0, 255, 0,
        G_PARAM_READWRITE));

g_object_class_install_property (gobject_class, PROP_IP3,
    g_param_spec_int ("ip-number-3",
        "IP Address Number 3",
        "The third IP address number",
        0, 255, 0,
        G_PARAM_READWRITE));

g_object_class_install_property (gobject_class, PROP_IP4,
    g_param_spec_int ("ip-number-4",
        "IP Address Number 1",
        "The fourth IP address number",
        0, 255, 0,
        G_PARAM_READWRITE));
}

```

클래스 초기화 함수에서 가장 먼저 해야 할 일은 위젯 클래스를 상속받을 GObjectClass에 필요한 함수를 모두 오버라이드하는 일이다. 이번 예제에서는 set\_property()와 get\_property()의 기본 구현을 오버라이드해야 했다. 이 함수들은 프로그래머가 각각 g\_object\_set()과 g\_object\_get()을 호출하면 호출되었다. 새로운 객체에 설치된 프로퍼티가 하나라도 있다면 이러한 함수들은 항상 오버라이드해야 한다.

■ **Note** GObjectClass에는 생성자(constructor), notify 시그널 콜백, 최종화 함수를 포함해 다른 많은 함수들이 제공된다. 오버라이드가 가능한 함수의 전체 리스트는 GObject API 문서에서 찾을 수 있다.

다음으로 MyIPAddressPrivate의 인스턴스는 g\_type\_class\_add\_private()을 이용해 위젯 클래스로 연결된다. 이 구조체는 위젯 프로퍼티에 대한 값을 보유할 것이다.

```

void g_type_class_add_private (gpointer klass,
    gsize private_size);

```

이 함수의 첫 번째 매개변수는 private 클래스와 연관될 위젯 클래스다. 그 다음에는 private 구조체의 크기가 따라오는데, sizeof()를 이용해 얻을 수 있다. GObject는 이러한 방식으로 private 데이터를 구현함으로써 C 프로그래밍 언어에서 허용하는 범위까지 데이터 숨김을 제공한다.

### 시그널 설치하기

필요한 가상 함수를 오버라이드했다면 그 다음은 위젯 클래스 초기화 함수에서 g\_signal\_new()를 이용해 자신의 객체에서 필요로 하는 시그널을 준비해야 한다. 이는 매우 길고 복잡한 함수이므로 매개변수를 한 번에 하나씩 살펴보도록 하겠다.

```

guint g_signal_new (const gchar *signal_name,
    GType class_type,
    GSignalFlags signal_flags,
    guint class_offset,
    GSignalAccumulator accumulator,
    gpointer accumulator_data,
    GSignalCMarshaller c_marshalller,
    GType return_type,
    guint n_parameters,
    ...);

```

`g_signal_new()`의 첫 번째 매개변수는 자신이 생성 중인 새로운 시그널의 이름이다. 이번 예제에서는 `ip-address` 시그널을 추가하고 있다. 이 이름은 콜백 함수로 시그널을 연결할 때 프로그래머가 `g_signal_connect()` 함수군(family of functions)에서 사용할 것이다. 프로그래머가 이름만으로 목적을 알아차리도록 가능한 한 설명적인 이름으로 만드는 것이 중요하다.

`g_signal_new()`의 다음 매개변수는 시그널을 포함하게 될 클래스의 `GType`을 제공한다. 이러한 타입은 인스턴스에서 `G_TYPE_FROM_CLASS()` 또는 `G_OBJECT_CLASS_TYPE()`을 호출하여 검색 가능하다. 그 다음에는 `GSignalFlags` 열거값에 정의된 시그널 플래그의 bitwise 리스트가 따라오는데, 그 정의는 다음과 같다.

- `G_SIGNAL_RUN_FIRST`: 첫 번째 발생단계(emission stage) 중에 이 시그널에 대한 핸들러를 호출한다. 이 플래그 집합과 함께 다른 객체 시그널이 실행 중일 때 실행될 것이다.
- `G_SIGNAL_RUN_LAST`: 세 번째 발생단계(emission stage) 중에 이 시그널에 대한 핸들러를 호출한다. 이 플래그 집합과 함께 다른 객체 시그널이 실행 중일 때 실행될 것이다.
- `G_SIGNAL_RUN_CLEANUP`: 마지막 발생단계(emission stage) 중에 이 시그널에 대한 핸들러를 호출한다. 이 플래그 집합과 함께 다른 객체 시그널이 실행 중일 때 실행될 것이다.
- `G_SIGNAL_NO_RECURSE`: 해당 객체에서 시그널이 이미 발생한 경우 재귀적 호출이 금지될 것이다. 대신 첫 번째 발생이 재시작될 뿐이다.
- `G_SIGNAL_DETAILED`: 연결과 시그널의 발생 시 시그널명에 추가된 `::detaildescriptor`의 지원을 추가한다.
- `G_SIGNAL_ACTION`: 이것을 설정 시 객체에 대한 발생전, 발생후 조정을 실행하지 않고도 `g_signal_emit()`와 그 friends를 이용해 해당 시그널을 발생시킬 수 있다. 이 객체를 이용하는 코드가 시그널을 발생하도록 허용하기 위함이다.
- `G_SIGNAL_NO_HOOKS`: 이 시그널에 대한 발생 훅(emission hook)을 지원하지 않는다.

`g_signal_new()`의 다음 매개변수는 시그널 프로토타입의 클래스에서 구조체 오프셋(structure offset)에 해당한다. 가령 `MyIPAddressClass`에서 `ip_changed()`의 오프셋을 얻기 위해 `G_STRUCT_OFFSET()`이 사용되었다. 이 함수는 아래 코드 조각에서 볼 수 있듯이 `GLib`에 의해 정의되는데, 아래 코드는 `struct_type` 내에서 `member`의 오프셋을 단순히 리턴하는 함수를 설명한다. 이는 `g_signal_new()`를 이용해 콜백 함수 프로토타입을 찾도록 해준다.

```
#define G_STRUCT_OFFSET(struct_type, member) \
    ((glong) ((guint8*) &((struct_type*) 0)->member))
```

다음으로 축적(accumulation)에 사용될 `GSignalAccumulator` 타입의 선택적 함수를 명시한 다음 그 함수로 전달될 데이터를 명시할 수 있다. 대부분의 경우는 두 가지 매개변수 모두 `NULL`로 설정될 것이다.

그 다음 매개변수 `GSignalCMarshaller`는 `closure marshal` 함수라고 부르는데, 매개변수 집합체를 `C`가 지원하는 콜백 함수로 번역하는 데에 사용된다. `GObject`는 표준 명명 규칙을 따르는 `closure marshal` 함수를 다수 제공한다. 대부분은 개발자가 별도로 생성할 필요가 없다.

가장 기본적 타입의 `closure marshal` 함수는 `g_cclosure_marshal_VOID__VOID()`다. 함수에 두 개의 밑줄 문자가 연속으로 포함되어 있는 표기법을 주목하라! 첫 번째 `VOID`는 `GObject`에게 콜백 함수의 리턴 타입이 `void`임을 알린다. 두 번째 `VOID`는 콜백 함수로 전송된 사용자 데이터와 인스턴스 외에 추가 매개변수가 없다고 말한다. 이러한 시그널 타입의 함수 프로토타입은 다음과 같다.

```
void (*callback) (gpointer instance, gpointer data);
```

또 다른 예로 `g_cclosure_marshal_VOID__BOOLEAN()`가 있다. 해당 시그널 타입에 해당하는 콜백 함수 프로토타입은 다음과 같다. 이는 `void`를 리턴하고, 객체 인스턴스와 사용자 데이터 사이에 위치한 추가 `gboolean` 매개변수를 갖고 있다.

```
void (*callback) (gpointer instance, gboolean arg, gpointer data);
```

위의 두 가지 `closure marshal` 함수 외에도 다른 기본적인 타입을 리턴하는 함수들도 많다. `GObject` 또한 `void`가 아닌 리턴값을 몇 가지 제공한다. 가령 `g_cclosure_marshal_STRING_OBJECT_POINTER()`는 `C` 문자열을 리턴

하고 두 개의 매개변수, 즉 GObject와 포인터를 수락한다. GLib 2.12에서 이용 가능한 closure marshal 함수의 전체 리스트는 다음과 같다.

- g\_cclosure\_marshal\_VOID\_\*(*void*): 이 함수들은 어떤 것도 리턴하지 않고 BOOLEAN, CHAR, UCHAR, INT, UINT, LONG, ULONG, ENUM, FLAGS, FLOAT, DOUBLE, STRING, PARAM, BOXED, POINTER, OBJECT, UINT\_POINTER 중에 하나의 추가 매개변수만 수락한다. VOID의 값은 콜백 함수로 하여금 두 개의 기본 매개변수만 갖도록 할 것이다.
- g\_cclosure\_marshal\_STRING\_OBJECT\_POINTER(): 이 함수는 문자열을 리턴하고 포인터와 GObject의 추가 매개변수를 수락한다.
- g\_cclosure\_marshal\_BOOLEAN\_FLAGS(): 이 함수는 gboolean 값을 리턴하고 G\_TYPE\_FLAGS가 정의한 bitwise 필드를 수락한다.

g\_signal\_new()에서 다음 매개변수는 콜백 함수에 사용될 리턴 타입을 제공한다. 가령 MyIPAddress 예제에서 콜백 함수는 리턴값을 갖고 있지 않으므로 G\_TYPE\_NONE이라고 부른다. 기본적으로 등록되는 기본 타입의 전체 리스트는 표 11-2에 실렸다.

GType	정의
G_TYPE_BOOLEAN	TRUE 또는 FALSE를 보유하는 표준 Boolean 타입
G_TYPE_BOXED	박스(boxed) 또는 구조체 타입을 나타내는 기본 타입
G_TYPE_CHAR G_TYPE_UCHAR	표준 C char 타입에 대한 부호가 있는(signed) 버전과 부호가 없는(unsigned) 버전
G_TYPE_DOUBLE	표준 C double 타입과 동일한 gdouble 변수
G_TYPE_ENUM	C enum 타입과 동일한 표준 열거
G_TYPE_FLAGS	Boolean 플래그를 보유하는 bitwise 필드
G_TYPE_FLOAT	표준 C float 타입과 동일한 gfloat 변수
G_TYPE_INT G_TYPE_UINT	표준 C int 타입에 대한 부호가 있는 버전과 부호가 없는 버전
G_TYPE_INT64 G_TYPE_UINT64	GLib의 64 비트 정수 구현에 대한 부호가 있는 버전과 부호가 없는 버전
G_TYPE_INTERFACE	인터페이스를 상속할 수 있는 기본 타입
G_TYPE_INVALID	일부 함수에 의해 오류 리턴값으로 사용되는 무효한(invalid) GType
G_TYPE_LONG G_TYPE_ULONG	표준 C long 타입에 대한 부호가 있는 버전과 부호가 없는 버전
G_TYPE_NONE	void와 동일하게 빈 타입
G_TYPE_OBJECT	GObject로 캐스팅되고 GObject에서 상속된 클래스를 나타내는 기본 타입
G_TYPE_PARAM	GParamSpec로부터 상속된 타입이라면 무엇이든 나타내는 기본 타입
G_TYPE_POINTER	void 포인터로 구현되는 타입이 정해지지 않은(untyped) 포인터 타입
G_TYPE_STRING	gchar 문자 배열에 대한 포인터로 저장된 NULL로 끝나는 C 문자열

표 11-2. 기본적인 GLib 타입

g\_signal\_new()에서 마지막 매개변수는 콜백 함수가 수락하는 매개변수의 개수인데, 인스턴스와 사용자 데이터에 이어 각 매개변수에 대한 타입 리스트는 제외한다. 우리 예제에는 별도의 타입이 추가되지 않았기 때문에 매개변수의 개수가 0으로 설정되었다. GtkEntry 위젯의 populate-popup 시그널에 대한 선언을 아래에서 살펴보자.

```
g_signal_new ("populate_popup",
             G_OBJECT_CLASS_TYPE (object_class), /* or G_TYPE_FROM_CLASS ()
```



```

*/
    G_SIGNAL_RUN_LAST,
    G_STRUCT_OFFSET (GtkEntryClass, populate_popup),
    NULL, NULL,
    _gtk_marshal_VOID__OBJECT, /* defined in gtkmarshal.h */
    G_TYPE_NONE, 1,
    GTK_TYPE_MENU);

```

위의 시그널 선언에서는 하나의 추가 매개변수가 콜백 함수로 전송되었는데, 이는 GtkMenu로 캐스팅되었다. GtkMenu 타입은 GTK\_TYPE\_MENU에서 정의된다. 이는 g\_cclosure\_marshal\_VOID\_\_OBJECT()가 정의한 일반 GObject 대신에 사용할 수 있는 좀 더 구체적인 매개변수 캐스트 타입을 제공한다.

프로퍼티 설치하기

이번 예제의 클래스 초기화 함수에서 마지막으로 할 일은 필요한 프로퍼티를 설치하는 일이다. MyIPAddress 위젯에는 네 개의 프로퍼티가 설치되어 있는데, 모두 정수에 해당한다.

프로퍼티는 g\_object\_class\_install\_property()를 이용해 GObjectClass에 설치된다. 이 함수에서 첫 두 개의 매개변수는 자신의 새로운 위젯 클래스와 프로퍼티 식별자에 해당하는 GObjectClass를 수락한다. 식별자는 부호가 없는 유일한 정수로서 특정 프로퍼티를 참조한다. 이러한 식별자들은 MyIPAddress에서와 마찬가지로 일반적으로 열거에 정의되어 식별자가 객체에 유일하도록 보장한다.

```

void g_object_class_install_property (GObjectClass *object_class,
    guint property_id,
    GParamSpec *pspec);

```

g\_object\_class\_install\_property()의 마지막 매개변수는 GParamSpec 객체로서, 프로퍼티가 보유하는 변수의 타입, 이름, 다양한 특성에 대한 정보를 포함한다. GParamSpec 객체를 설정하도록 많은 함수가 제공된다.

IPAddress 예제에서는 g\_param\_spec\_int()를 이용해 G\_TYPE\_INT 타입의 프로퍼티에 대한 새로운 GParamSpecInt 구현을 준비하였다. 이 함수에서 첫 세 개의 매개변수는 프로퍼티명, 프로퍼티에 대한 짧은 별칭(short nickname), 프로퍼티에 대한 설명을 각각 나타낸다. 프로퍼티명은 g\_object\_set()과 g\_object\_get()을 호출하여 접근할 것이다.

```

GParamSpec* g_param_spec_int (const gchar *name,
    const gchar *nick,
    const gchar *blurb,
    gint minimum,
    gint maximum,
    gint default_value,
    GParamFlags flags);

```

그 다음 세 개의 매개변수는 프로퍼티에 가능한 최소값, 최대값, 기본값을 정의한다. 이러한 값들은 프로퍼티 범위를 비롯해 초기 상태를 정의하는 데에 사용된다. 마지막 매개변수는 프로퍼티에 적용 가능한 아래의 GParamFlags 열거로부터 플래그를 정의하도록 해준다.

- G\_PARAM\_READABLE: 매개변수의 값을 읽는 것이 가능하다.
- G\_PARAM\_WRITABLE: 매개변수에 대해 새 값을 쓰는 것이 가능하다.
- G\_PARAM\_CONSTRUCT: 객체의 생성이 완료되면(constructed) 매개변수가 설정될 것이다.
- G\_PARAM\_CONSTRUCT\_ONLY: 객체의 생성이 완료되었을 때에만 해당 매개변수가 설정될 것이다.
- G\_PARAM\_LAX\_VALIDATION: g\_param\_value\_convert()를 이용해 매개변수를 변환하면 엄격한 검증(strict validation)이 필요하지 않을 것이다.

- `G_PARAM_STATIC_NAME`: 매개변수가 존재하는 동안 매개변수명은 절대 수정되는 일이 없고 유효하게 남을 것이다.
- `G_PARAM_STATIC_NICK`: 매개변수가 존재하는 동안 매개변수의 별칭은 절대 수정되는 일이 없고 유효하게 남을 것이다.
- `G_PARAM_STATIC_BLURB`: 매개변수가 존재하는 동안 매개변수 설명은 절대 수정되는 일이 없고 유효하게 남을 것이다.

`G_PARAM_READWRITE`라는 추가 플래그도 있는데, 이는 (`G_PARAM_READABLE | G_PARAM_WRITABLE`)에 대한 bitwise alias로 정의된다. 이는 `GParamFlags`의 열거 값 대신 매크로로서 포함된다.

GLib에서 제공되는 모든 기본적인 데이터 타입에 이용 가능한 `GParamSpec` 구조체들이 존재한다. 다른 프로퍼티 타입을 생성하기 위한 함수 프로토타입의 전체 리스트는 `GObject` API 문서를 참조하도록 한다.

### 매개변수와 값 정의

여러 가지의 기본 타입에 대해 다수의 `GParamSpec` 및 `GValue` 함수가 정의되어 있다. 아래 리스트는 이러한 함수 각각에 대한 설명을 제공한다. 아래 각 함수에서 별표 문자는 함수의 경우에 따라 `boolean`, `char`, `uchar`, `int`, `uint`, `long`, `ulong`, `int64`, `uint64`, `float`, `double`, `enum`, `flags`, `string`, `param`, `boxed`, `pointer`, `object`, `gtype` 중 하나로 대체가 가능하다.

- `G_IS_PARAM_SPEC_*`(): 주어진 객체가 주어진 타입에 유효한 매개변수 명세(parameter specification) 객체일 경우 `TRUE`를 리턴한다.
- `G_PARAM_SPEC_*`(): `GParamSpecobject` 를 특정 매개변수 명세 타입으로 캐스팅한다.
- `G_VALUE HOLDS_*`(): 주어진 `GValue`가 함수에서 정의한 타입을 보유할 수 있는 경우 `TRUE`를 리턴한다.
- `G_TYPE_PARAM_*`(): 주어진 매개변수 명세 타입에 대한 `GType`을 리턴한다.
- `g_param_spec_*`(): 주어진 타입의 새로운 매개변수 명세를 생성한다. 이 함수는 주로 `GObject`에 대한 새로운 프로퍼티를 정의할 때 사용된다. 모든 함수는 프로퍼티명, 별칭, 간략한 설명, `GParamFlags`의 bitwise 리스트, 주어진 타입에 관련된 매개변수를 수락한다. 리턴값은 새로운 `GParamSpec` 객체가 된다.
- `g_value_set_*`(): `GValue` 객체가 저장한 값을 주어진 변수로 설정한다. 새로운 값은 함수와 동일한 타입이어야 한다.
- `g_value_get_*`(): 이미 주어진 타입으로 캐스팅된 `GValue` 객체가 저장한 값을 검색한다.

**Note** 앞의 함수 리스트에서 별표 문자를 데이터 타입으로 대체할 때는 함수의 대소문자를 매치해야 한다. 예를 들어 `G_IS_PARAM_SPEC_*`에서 별표는 모두 대문자로 된 데이터 타입으로 대체되어야 한다. 상세한 예제는 API 문서를 참조한다.

### 객체 프로퍼티를 설정하고 검색하기

클래스 초기화 함수에서는 기본 함수 `set_property()`와 `get_property()`가 `GObjectClass`에서 오버라이드되었다. 새로운 `GObject`에 하나 또는 이상의 프로퍼티가 있다면 이 두 개의 함수는 오버라이드되어선 안 된다. 리스팅 11-8에서는 `g_object_set()`으로 전송된 프로퍼티마다 호출될 함수의 구현을 소개하겠다.

리스팅 11-8. 객체 프로퍼티 설정하기

```
static void
my_ip_address_set_property (GObject *object,
                           guint prop_id,
                           const GValue *value,
                           GParamSpec *pspec)
{
    MyIPAddress *ipaddress = MY_IP_ADDRESS (object);
    gint address[4] = { -1, -1, -1, -1 };
}
```

```

switch (prop_id)
{
    case PROP_IP1:
        address[0] = g_value_get_int (value);
        my_ip_address_set_address (ipaddress, address);
        break;
    case PROP_IP2:
        address[1] = g_value_get_int (value);
        my_ip_address_set_address (ipaddress, address);
        break;
    case PROP_IP3:
        address[2] = g_value_get_int (value);
        my_ip_address_set_address (ipaddress, address);
        break;
    case PROP_IP4:
        address[3] = g_value_get_int (value);
        my_ip_address_set_address (ipaddress, address);
        break;
    default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
        break;
}
}

```

프로퍼티가 도착하면 어떤 객체 타입이든 저장할 수 있는 일반 컨테이너 GValue 객체로 저장된다. GValue가 저장한 정수값을 검색 시에는 g\_value\_get\_int() 함수가 사용된다. 모든 기본 데이터 타입과 앞 절에서 정의한 GValue 객체 간 변환에 이용할 수 있는 함수들도 있다.

다음 단계는 프로퍼티가 유효할 경우 그것의 새로운 값을 저장하는 단계다. 프로퍼티 식별자는 prop\_id에 저장되는데, 이를 이용해 설치된 프로퍼티와 비교하여 수정 중인 프로퍼티 식별자를 찾을 수 있다. 변경내용을 적용하는 데에는 my\_ip\_address\_set\_address() 함수가 사용되었다. 이 함수의 구현은 이번 절의 뒷부분에 소개 하겠다.

이 함수가 제공하는 기능은 아래 코드 조각에 제시한 방법을 이용해도 구현이 가능하다. 하지만 위젯의 프로 퍼티가 모두 동일한 타입이고 배열에 저장되는 경우가 극히 드물기 때문에 확장된 형태로 제시함을 명심한다.

```

address[prop_id-1] = g_value_get_int (value);
my_ip_address_set_address (ipaddress, address);

```

이번 장의 MyIPAddress 예제는 가장 간단한 예제에 속하기 때문에 상당히 크게 확장이 가능하다. 애플리케이션에서 이 위젯을 사용하기 위해 구현하는 중이라면 프로퍼티와 시그널 뿐만 아니라 추가 기능도 제공할 것 이다. 이 점을 유념하면서 계속 살펴보도록 한다.

객체 클래스의 기본 함수 get\_property()가 오버라이드되었다. 따라서 MyIPAddress의 프로퍼티에서 g\_object\_set()이 호출되면 리스팅 11-9에서와 같이 my\_ip\_address\_get\_property()가 호출될 것이다.

리스팅 11-9. 객체 프로퍼티 검색하기

```

static void
my_ip_address_get_property (GObject *object,
    guint prop_id,
    GValue *value,

```

```

        GParamSpec *pspec)
{
    MyIPAddress *ipaddress = MY_IP_ADDRESS (object);
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (ipaddress);

    switch (prop_id)
    {
        case PROP_IP1:
            g_value_set_int (value, priv->address[0]);
            break;
        case PROP_IP2:
            g_value_set_int (value, priv->address[1]);
            break;
        case PROP_IP3:
            g_value_set_int (value, priv->address[2]);
            break;
        case PROP_IP4:
            g_value_set_int (value, priv->address[3]);
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

```

my\_ip\_address\_get\_property() 함수는 MyIPAddressPrivate 구조체에서 적절한 프로퍼티를 취해 GValue로 변환한다. 이후 새 값은 사용자의 변수로 적용되어 올바른 변수로 캐스팅된다. Private 구조체는 소스 파일의 맨 위에 정의된 MY\_IP\_ADDRESS\_GET\_PRIVATE() 함수를 이용해 검색된다.

#### 위젯 인스턴스화하기

또 구현해야 하는 초기화 함수로 my\_ip\_address\_init()가 있는데, 이는 새로운 MyIPAddress 위젯이 생성될 때마다 호출된다. 이러한 함수는 객체가 인스턴스화될 때마다 호출되는 것이 아니라 객체 클래스를 설정할 때만 호출되는 클래스 초기화 함수와는 차이가 있다. 리스팅 11-10에 표시된 인스턴스 초기화 함수는 초기 IP 주소값을 0으로 설정하고, 초기 렌더링을 실행하며, 필요한 시그널을 연결하는 일을 수행한다.

#### 리스팅 11-10. MyIPAddress 객체 인스턴스화하기

```

static void
my_ip_address_init (MyIPAddress *ipaddress)
{
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (ipaddress);
    PangoFontDescription *fd;
    guint i;

    for (i = 0; i < 4; i++)
        priv->address[i] = 0;

    fd = pango_font_description_from_string ("Monospace");
    gtk_widget_modify_font (GTK_WIDGET (ipaddress), fd);
}

```

```

my_ip_address_render (ipaddress);
pango_font_description_free (fd);

/* The key-press-event signal will be used to filter out certain
keys. We will
* also monitor the cursor-position property so it can be moved
correctly. */
g_signal_connect (G_OBJECT (ipaddress), "key-press-event",
                  G_CALLBACK (my_ip_address_key_pressed), NULL);
g_signal_connect (G_OBJECT (ipaddress), "notify::cursor-position",
                  G_CALLBACK (my_ip_address_move_cursor), NULL);
}

```

`my_ip_address_init()` 함수는 생성 및 캐스팅이 이미 완료된 `MyIPAddress` 객체를 수락한다. 여기서 개발자가 해야 할 일은 위젯이 개발자에게 리턴되어 사용자에게 표시되기 전에 위젯에서 실행되어야 하는 추가 처리를 실행하는 것이다.

이번 예제에서 함수는 먼저 네 개의 IP 주소값을 0으로 초기화한다. 위젯의 글꼴은 `Monospace`로 설정된다. 크기는 명시되지 않으므로 사용자의 테마에 따른 크기가 허용됨을 주목한다. 이는 글꼴이 크게 설정된 사용자도 위젯의 내용을 읽을 수 있도록 보장하기 위함이다.

마지막으로 `MyIPAddress` 위젯은 두 개의 시그널로 연결된다. `key-press-event` 콜백 함수는 위젯이 반응하게 될 키를 필터링할 것이다. 이후 `cursor-position`이 변경되면 위치가 업데이트되어 텍스트가 입력될 곳을 제어할 수 있게 된다. `MyIPAddress`는 `GtkEntry`에서 상속되므로 `GtkEntry`의 모든 member, 프로퍼티, 시그널, 함수 등도 상속받는다는 사실을 명심한다. 뿐만 아니라 조상 클래스에 해당하는 `GtkWidget`, `GtkObject`, `GObject`에서도 모든 것을 상속받는다.

다음으로 위젯이 사용자와 상호작용하는 방식을 처리하도록 몇 가지 `private` 함수들이 구현된다. 리스팅 11-11은 `my_ip_address_render()`라는 함수를 소개한다. 이 함수는 IP 주소값에서 문자열을 빌드하여 `GtkEntry` 위젯으로 추가한다. 이는 `GtkEntry` 위젯으로 작성하는 유일한 함수다.

리스팅 11-11. `MyIPAddress` 위젯 렌더링하기

```

/* Render the current content of the IP address in the GtkEntry widget.
*/
static void
my_ip_address_render (MyIPAddress *ipaddress)
{
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (ipaddress);
    GString *text;
    guint i;

    /* Create a string that displays the IP address content, adding
spaces if a
* number cannot fill three characters. */
    text = g_string_new (NULL);
    for (i = 0; i < 4; i++)
    {
        gchar *temp = g_strdup_printf ("%3i.", priv->address[i]);
        text = g_string_append (text, temp);
        g_free (temp);
    }
}

```

```

}

/* Remove the trailing decimal place and add the string to the
GtkEntry. */
text = g_string_truncate (text, 15);
gtk_entry_set_text (GTK_ENTRY (ipaddress), text->str);
g_string_free (text, TRUE);
}

```

이 함수는 현재 `MyIPAddressPrivate`의 인스턴스에 저장된 네 개의 정수와 세 개의 마침표(period)를 이용해 15개 문자로 된 IP 주소를 빌드하기 위해 `GString`을 이용한다. 이 문자열은 `GtkEntry` 위젯에서 사용자에게 표시된다. 정수가 세 개의 공간을 차지하지 않을 경우 하나 또는 두 개의 공백(space) 문자로 패딩되어 IP 주소는 항상 15개 문자 너비를 가질 것이다. 너비가 보장되어야만 커서가 위치해야 할 정확한 장소를 언제든지 알 수 있다.

`MyIPAddress` 위젯은 커서가 네 개의 위치 중 하나에 강제로 위치하도록 빌드된다. 각 숫자는 우측으로 정렬되고, 필요 시 좌측에 공백이 패딩된다. 이 때문에 커서는 네 개 중 하나의 숫자 우측에 강제로 위치된다. 이는 리스팅 11-12에 표시된 `notify::cursor-position` 콜백 함수에서 실행된다.

리스팅 11-12. `MyIPAddress`에 대한 콜백 함수

```

/* Force the cursor to always be at the end of one of the four numbers.
*/
static void
my_ip_address_move_cursor (GObject *entry,
                           GParamSpec *spec)
{
    gint cursor = gtk_editable_get_position (GTK_EDITABLE (entry));

    if (cursor <= 3)
        gtk_editable_set_position (GTK_EDITABLE (entry), 3);
    else if (cursor <= 7)
        gtk_editable_set_position (GTK_EDITABLE (entry), 7);
    else if (cursor <= 11)
        gtk_editable_set_position (GTK_EDITABLE (entry), 11);
    else
        gtk_editable_set_position (GTK_EDITABLE (entry), 15);
}

/* Handle key presses of numbers, tabs, backspaces and returns. */
static gboolean
my_ip_address_key_pressed (GtkEntry *entry,
                           GdkEventKey *event)
{
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (entry);
    guint k = event->keyval;
    gint cursor, value;

    /* If the key is an integer, append the new number to the address.

```

```

This is only
    * done if the resulting number will be less than 255. */
    if ((k >= GDK_0 && k <= GDK_9) || (k >= GDK_KP_0 && k <= GDK_KP_9))
    {
        cursor = floor (gtk_editable_get_position (GTK_EDITABLE
(entry)) / 4);
        value = g_ascii_digit_value (event->string[0]);

        if ((priv->address[cursor] == 25) && (value > 5))
            return TRUE;

        if (priv->address[cursor] < 26)
        {
            priv->address[cursor] *= 10;
            priv->address[cursor] += value;
            my_ip_address_render (MY_IP_ADDRESS (entry));
            gtk_editable_set_position (GTK_EDITABLE (entry), (4 *
cursor) + 3);
            g_signal_emit_by_name ((gpointer) entry, "ip-changed");
        }
    }

    /* Move to the next number or wrap around to the first. */
    else if (k == GDK_Tab)
    {
        cursor = (floor (gtk_editable_get_position (GTK_EDITABLE
(entry)) / 4) + 1);
        gtk_editable_set_position (GTK_EDITABLE (entry), (4 * (cursor %
4) + 3));
    }

    /* Delete the last digit of the current number. This just divides
the number by
    * 10, relying on the fact that any remainder will be ignored. */
    else if (k == GDK_BackSpace)
    {
        cursor = floor (gtk_editable_get_position (GTK_EDITABLE
(entry)) / 4);
        priv->address[cursor] /= 10;
        my_ip_address_render (MY_IP_ADDRESS (entry));
        gtk_editable_set_position (GTK_EDITABLE (entry), (4 * cursor) +
3);
        g_signal_emit_by_name ((gpointer) entry, "ip-changed");
    }

    /* Activate the GtkEntry widget, which corresponds to the activate
signal. */

```

```

else if ((k == GDK_Return) || (k == GDK_KP_Enter))
    gtk_widget_activate (GTK_WIDGET (entry));

return TRUE;
}

```

리스팅 11-12에는 두 번째 함수 `my_ip_address_key_pressed()`도 포함되어 있는데, 이 함수는 `key-press-event` 시그널이 발생할 때 호출된다. 해당 함수는 특정 키만 처리하고 그 외에는 모두 무시한다. 가령, 숫자 키만 처리하고 문자와 기호는 모두 무시되는 경우를 예로 들 수 있겠다. 처리되는 키 집합을 한 번에 하나씩 살펴보도록 하겠다.

첫 번째 조건문(conditional)은 `<gdk/gdkkeysyms.h>`에 정의된 바와 같이 키보드의 상단이든 키패드든 키보드에서 누른 숫자를 처리한다. `GDK_KP_#`는 숫자 패드의 숫자 키에 해당하고 `GDK_#`는 키보드 상단에 위치한 숫자 키에 해당하는데, 둘 다 조건문에서 설명되어야 한다.

커서 위치를 0과 3 사이의 숫자로 변환하기 위해 `floor()` 함수가 사용되었는데, IP 주소값이 그 값으로 편집되어야 함을 나타낸다. 이벤트 문자열 또한 `g_ascii_digit_value()`를 이용해 정수로 변환된다.

이제 필요한 값이 모두 있으니 두 개의 조건문이 새 값의 유효성을 검사한다. 새로운 값이 255를 초과하지 않을 경우에 이 새로운 정수는 현재값의 뒤에 추가된다. 숫자가 범위 내에 있다면 현재 값이 스케일링되고 (scaled), 새로운 정수는 뒤에 추가된다. 그 다음 새로운 IP 주소가 `GtkEntry` 위젯에서 렌더링되고, 커서 위치가 새로고침(refresh)된다. 마지막으로 `g_signal_emit_by_name()`을 이용해 IP 주소가 변경되었음을 사용자에게 알린다.

두 번째 조건문은 Tab 키를 처리하는데, 이 키를 누르면 네 개의 숫자를 각각 순환할 것이다. 위젯의 또 다른 구현에서는 위젯의 끝에 도달하면 탭 순으로 다음 위젯을 순환하도록 수정할 수도 있다.

다음으로 Backspace 키는 현재 값을 10으로 나눈다. 정수를 정수로 나누기 때문에 나머지 값은 무시되고, 마지막 자릿수는 없앤다(drop off). 이후 위젯이 렌더링되고 `ip-changed` 시그널이 발생한다.

마지막으로 Return 키와 Enter 키를 누르면 `gtk_widget_activate()`를 호출한다. 이는 사용자가 이러한 키를 이용해 `MyIPAddress` 위젯 내부로부터 창의 기본 위젯을 활성화하도록 해준다. 이번 절에서 다른 키를 제외한 모든 키 누름은 무시된다.

**Public MyIPAddress 함수 구현하기**

위젯을 생성하기 위한 마지막 단계는 위젯의 헤더 파일에 선언된 `public` 함수들을 구현하는 것이다. 첫 번째 함수는 `my_ip_address_new()`로, 리스팅 11-13에서 새로운 `MyIPAddress` 위젯을 생성하는 데에 사용하였다.

리스팅 11-13. 새로운 `MyIPAddress` 위젯 생성하기

```

GtkWidget *
my_ip_address_new ()
{
    return GTK_WIDGET (g_object_new (my_ip_address_get_type (), NULL));
}

```

이 함수는 `g_object_new()`가 리턴한 객체 `GtkWidget`로 캐스팅하는 작업만 제공함을 눈치챌 것이다. 이는 많은 위젯에서 편의 함수에 불과하므로 프로그래머는 `GObject` 인스턴스 자체를 생성할 필요가 없다. 초기화 함수로 매개변수를 수락하는 위젯을 사용 중이라면 이 곳에서 처리해야 할 것이다.

리스팅 11-14의 `my_ip_address_get_address()` 함수는 현재 위젯이 저장한 IP 주소의 문자열 표현을 리턴한다. 이렇게 리턴된 문자열을 모두 사용하고 나면 해제하는 일도 프로그래머의 몫인데, 문자열이 `g_strdup_printf()`를 이용해 생성되기 때문이다. 사용자는 이를 프로그램적으로 생성할 수도 있지만 대부분의 위젯은 자주 필요한 작업을 수행하는 수많은 편의 함수들을 제공하는 것이 보통이다.

리스팅 11-14. 현재 IP 주소 검색하기



```

gchar*
my_ip_address_get_address (MyIPAddress *ipaddress)
{
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (ipaddress);

    return g_strdup_printf ("%d.%d.%d.%d", priv->address[0],
priv->address[1],
    priv->address[2], priv->address[3]);
}

```

마지막 함수 `my_ip_address_set_address()`는 프로그램의 변경내용을 IP 주소로 적용하는데, 이를 리스팅 11-15에 소개하겠다. 함수가 0보다 작거나 255보다 큰 숫자를 필터링하는 모습을 확인할 수 있을 것이다. 이를 통해 프로그래머는 IP 주소 숫자마다 새로운 값을 제공하지 않아도 된다. 다시 말해, 프로그래머가 하나의 값만 업데이트하면 되기 때문에 프로그램적으로 IP 주소를 업데이트하도록 하나의 함수만 제공하면 된다는 말이다.

리스팅 11-15. 새로운 IP 주소 설정하기

```

void
my_ip_address_set_address (MyIPAddress *ipaddress,
    gint address[4])
{
    MyIPAddressPrivate *priv = MY_IP_ADDRESS_GET_PRIVATE (ipaddress);
    guint i;

    for (i = 0; i < 4; i++)
    {
        if (address[i] >= 0 && address[i] <= 255)
        {
            priv->address[i] = address[i];
        }
    }

    my_ip_address_render (ipaddress);
    g_signal_emit_by_name ((gpointer) ipaddress, "ip-changed");
}

```

위젯 테스트하기

이번 예제에서 마지막으로 해야 할 일은 위젯이 작동하는지 검사하는 일이다. 리스팅 11-16에 실린 코드는 새로운 `MyIPAddress` 위젯이 있는 창을 하나 생성한다. 초기 IP 주소로 1.20.35.255가 추가되고, `ip-changed` 시그널이 IP 주소의 현재 상태를 출력하는 콜백 함수로 연결된다.

리스팅 11-16. `MyIPAddress` 위젯 테스트하기 (`ipaddressstest.c`)

```

#include <gtk/gtk.h>
#include "myipaddress.h"

static void ip_address_changed (MyIPAddress*);

int main (int argc,
    char *argv[])

```

```

{
    GtkWidget *window, *ipaddress;
    gint address[4] = { 1, 20, 35, 255 };

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "MyIPAddress");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    g_signal_connect (G_OBJECT (window), "destroy",
                     G_CALLBACK (gtk_main_quit), NULL);

    ipaddress = my_ip_address_new ();
    my_ip_address_set_address (MY_IP_ADDRESS (ipaddress), address);
    g_signal_connect (G_OBJECT (ipaddress), "ip-changed",
                     G_CALLBACK (ip_address_changed), NULL);

    gtk_container_add (GTK_CONTAINER (window), ipaddress);
    gtk_widget_show_all (window);

    gtk_main ();
    return 0;
}

/* When the IP address is changed, print the new value to the screen. */
static void
ip_address_changed (MyIPAddress *ipaddress)
{
    gchar *address = my_ip_address_get_address (ipaddress);
    g_print ("%s\n", address);
    g_free (address);
}

```

위의 MyIPAddress 위젯은 새로운 위젯을 생성하는 가장 간단한 예제에 속한다. 따라서 실제 애플리케이션에서 사용하려면 상당 부분 확장되어야 한다. 예를 들어서 개발자는 사용자가 위젯을 오른쪽 마우스로 클릭했을 때 표시되는 팝업 메뉴를 맞춤설정하길 원할 것이다. 또 프로그래머가 커스텀 IP 주소 포맷을 정의하도록 허용하는 경우도 예로 들 수 있겠다. 이번 절에서 다른 내용을 더 잘 이해하기 위해서는 다음 절로 넘어가기 전에 MyIPAddress 위젯을 확장해볼 것을 권한다.

### 처음부터 위젯 생성하기

이미 존재하는 위젯에서 새로운 위젯을 상속하는 방법은 학습했으니 이제 처음부터 위젯을 생성하는 방법을 배워볼 차례다. 이번 절에 소개된 코드의 대부분은 앞에서 소개한 코드와 비슷함을 눈치챌 것이다. 이는 상속된 위젯과 새로운 위젯 모두 GObject의 기본 타입이어서 약간의 수고는 더 들여야 하지만 구현 방식은 동일하기 때문이다.

이번 절에서는 MyMarquee라고 불리는 위젯의 구현 방법을 학습할 것이다. 이 위젯은 위젯의 우측에서 좌측으로 메시지를 반복하여 스크롤한다. 이번 장에 실린 연습문제에서도 이 위젯을 확장할 것이기 때문에 위젯을 잘 이해하는 편이 좋을 것이다.

MyMarquee 위젯의 스크린샷은 그림 11-2에서 확인할 수 있다. 여느 예제와 마찬가지로 위젯에 대한 소스코드는 서적의 웹 사이트에서 다운로드 가능하다.



### MyMarquee 헤더 파일 생성하기

새로운 위젯을 준비하기 위한 첫 번째 단계는 헤더 파일을 생성하는 것이다. 헤더 파일을 생성하면 위젯을 제어하는 데에 사용될 프로그램적 인터페이스를 정의할 수 있도록 해준다. 리스팅 11-17(원문에 11-7로 되어 있어 수정하였습니다)은 MyMarquee 위젯에 대한 전체 헤더 파일을 제공하는데, MyIPAddress 헤더 파일과 매우 유사할 것이다.

리스팅 11-17. MyMarquee 위젯 헤더 (mymarquee.h)

```
#ifndef __MY_MARQUEE_H__
#define __MY_MARQUEE_H__

#include <glib.h>
#include <gdk/gdk.h>
#include <gtk/gtkwidget.h>

G_BEGIN_DECLS

#define MY_MARQUEE_TYPE (my_marquee_get_type ())
#define MY_MARQUEE(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
    MY_MARQUEE_TYPE, MyMarquee))
#define MY_MARQUEE_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST ((klass), \
    MY_MARQUEE_TYPE, MyMarqueeClass))
#define IS_MY_MARQUEE(obj) (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
    MY_MARQUEE_TYPE))
#define IS_MY_MARQUEE_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE ((klass), \
    MY_MARQUEE_TYPE))

typedef struct _MyMarquee MyMarquee;
typedef struct _MyMarqueeClass MyMarqueeClass;

struct _MyMarquee
```

```

{
    GtkWidget widget;
};

struct _MyMarqueeClass
{
    GtkWidgetClass parent_class;
};

GType my_marquee_get_type (void) G_GNUC_CONST;
GtkWidget* my_marquee_new (void);

void my_marquee_set_message (MyMarquee *marquee, const gchar *message);
gchar* my_marquee_get_message
(MyMarquee *marquee);

void my_marquee_set_speed (MyMarquee *marquee, gint speed);
gint my_marquee_get_speed (MyMarquee *marquee);

void my_marquee_slide (MyMarquee *marquee);

G_END_DECLS

#endif /* __MY_MARQUEE_H__ */

```

MyMarquee는 새로운 위젯이기 때문에 GtkWidget에서 직접 상속될 것이다. 이는 MyMarquee가 GtkWidget 객체를 포함하고 MyMarqueeClass가 GtkWidgetClass 클래스를 포함한다는 사실로 확인할 수 있다. 이러한 member들 중 어떤 것도 포인터로 선언되어선 안 된다는 기억을 되살려보자! GtkWidget에서 위젯을 상속하면 이벤트 처리를 비롯해 모든 위젯에 공통되는 시그널과 프로퍼티를 모두 이용할 수 있다.

위젯에는 프로그래머가 설정 및 검색할 수 있는 프로퍼티가 두 개 있다. 사용자는 my\_marquee\_set\_message()를 이용해 위젯이 스크롤할 메시지를 변경할 수 있다. 속도는 1과 50 사이의 정수에 해당한다.

my\_marquee\_slide()가 호출될 때마다 그 정수에 해당하는 수의 픽셀만큼 메시지가 좌측으로 이동할 것이다.

### MyMarquee 위젯 생성하기

헤더 파일이 생성되었으니 리스팅 11-18에서는 private 클래스의 선언, 프로퍼티 열거, 새로운 GType 생성과 같은 기본적인 초기화를 실행한다. 이 위젯에 연관된 새로운 시그널은 없으므로 시그널 열거와 시그널 식별자의 배열은 생략하였다.

리스팅 11-18. MyMarqueePrivate과 MyMarquee GType 정의하기 (mymarquee.c)

```

#include "mymarquee.h"

#define MARQUEE_MIN_WIDTH 300

#define MY_MARQUEE_GET_PRIVATE(obj) (G_TYPE_INSTANCE_GET_PRIVATE
((obj), \
    MY_MARQUEE_TYPE, MyMarqueePrivate))

typedef struct _MyMarqueePrivate MyMarqueePrivate;

```

```

struct _MyMarqueePrivate
{
    gchar *message;
    gint speed;
    gint current_x;
};

enum
{
    PROP_0,
    PROP_MESSAGE,
    PROP_SPEED
};

/* Get a GType that corresponds to MyMarquee. The first time this
function is
* called (on object instantiation), the type is registered. */
GType
my_marquee_get_type ()
{
    static GType marquee_type = 0;

    if (!marquee_type)
    {
        static const GTypeInfo marquee_info =
        {
            sizeof (MyMarqueeClass),
            NULL,
            NULL,
            (GClassInitFunc) my_marquee_class_init,
            NULL,
            NULL,
            sizeof (MyMarquee),
            0,
            (GInstanceInitFunc) my_marquee_init,
        };

        marquee_type = g_type_register_static (GTK_TYPE_WIDGET,
        "MyMarquee",
            &marquee_info, 0);
    }

    return marquee_type;
}

```

리스팅 11-18은 MyMarquee 위젯의 구현 중 첫 번째 부분만 표시한다. 필요한 위젯 프로퍼티의 값을 보유하는 데에 사용될 MyMarqueePrivate 구조체를 생성함으로써 파일을 시작하고자 한다. 파일에는 표시되는 메시지,

스크롤 속도, 현재 메시지의 수평적 위치가 포함된다. 이 위치를 기준으로 메시지의 다음 위치가 계산되어 위젯의 크기조정을 쉽게 처리하도록 해준다.

MyMarquee는 GtkWidget에서 직접 상속되기 때문에 my\_marquee\_get\_type()의 구현에서 볼 수 있듯이 위젯을 GTK\_TYPE\_WIDGET의 부모 클래스 타입으로 등록할 필요가 있겠다. my\_marquee\_get\_type() 함수의 구현은 my\_ip\_address\_get\_type()의 구현과 거의 동일하다.

리스트링 11-19에서는 MyMarquee 클래스와 인스턴스 초기화 함수를 표시하였다. my\_marquee\_class\_init()에서는 GObjectClass 뿐만 아니라 GtkWidgetClass에서도 함수를 오버라이드함을 눈치챌 것이다.

리스트링 11-19. MyMarquee 클래스와 구조체 초기화하기

```

/* Initialize the MyMarqueeClass class by overriding standard
functions,
* registering a private class and setting up signals and properties. */
static void
my_marquee_class_init (MyMarqueeClass *klass)
{
    GObjectClass *gobject_class;
    GtkWidgetClass *widget_class;

    gobject_class = (GObjectClass*) klass;
    widget_class = (GtkWidgetClass*) klass;

    /* Override the standard functions for setting and retrieving
properties. */
    gobject_class->set_property = my_marquee_set_property;
    gobject_class->get_property = my_marquee_get_property;

    /* Override the standard functions for realize, expose, and size
changes. */
    widget_class->realize = my_marquee_realize;
    widget_class->expose_event = my_marquee_expose;
    widget_class->size_request = my_marquee_size_request;
    widget_class->size_allocate = my_marquee_size_allocate;

    /* Add MyMarqueePrivate as a private data class of MyMarqueeClass.
*/
    g_type_class_add_private (klass, sizeof (MyMarqueePrivate));

    /* Register four GObject properties, the message and the speed. */
    g_object_class_install_property (gobject_class, PROP_MESSAGE,
        g_param_spec_string ("message",
            "Marquee Message",
            "The message to scroll",
            "",
            G_PARAM_READWRITE));

    g_object_class_install_property (gobject_class, PROP_SPEED,
        g_param_spec_int ("speed",

```

```

        "Speed of the Marquee",
        "The percentage of movement every second",
        1, 50, 25,
        G_PARAM_READWRITE) );
}

/* Initialize the actual MyMarquee widget. This function is used to set
up
* the initial view of the widget and set necessary properties. */
static void
my_marquee_init (MyMarquee *marquee)
{
    MyMarqueePrivate *priv = MY_MARQUEE_GET_PRIVATE (marquee);

    priv->current_x = MARQUEE_MIN_WIDTH;
    priv->speed = 25;
}

```

다음 단계는 `GTypeInfo` 객체에 의해 참조되는 클래스 및 인스턴스 초기화 함수를 구현하는 일이다. 이 예제에서는 부모 `GObjectClass`에서 함수를 오버라이드할 뿐만 아니라 `GtkWidgetClass`에서도 몇 가지 함수를 오버라이드해야 한다. 여기에는 위젯의 실현(realizing) 및 노출(exposing)을 비롯해 크기 요청과 할당(allocation)을 위한 호출의 오버라이드도 포함된다.

`GtkWidgetClass`에서 함수를 오버라이드할 때는 위젯에 결정적인 업무를 실행하므로 매우 주의를 기울여야 한다. 필요한 함수를 모두 실행하지 않으면 위젯을 사용할 수 없도록(unusable) 렌더링할 수도 있다. 이를 시도할 때는 다른 GTK+ 위젯들이 오버라이드된 함수를 어떻게 구현하는지 살펴볼 것을 권한다. 오버라이드가 가능한 함수의 전체 리스트는 `<gtk/gtkwidget.h>`에서 `GtkWidgetClass` 구조체를 참조한다.

`MyMarqueePrivate` 구조체는 `g_type_class_add_private()`을 이용해 클래스 초기화 함수에서 `MyMarqueeClass`로 추가되었다. 객체는 `MyMarqueeClass` 구조체의 member로 저장되지 않기 때문에 인스턴스 초기화 함수에서 볼 수 있듯이 `MY_MARQUEE_GET_PRIVATE()`의 정의를 이용해 `MyMarqueePrivate` 객체를 검색할 필요가 있겠다.

`my_marque_init()`에서 메시지의 현재 위치는 위젯의 우측 아래에 표시된다. `my_marquee_slide()`가 프로그램적으로 호출되면 메시지는 기본적으로 25 픽셀만큼 좌측으로 스크롤된다.

오버라이드된 `set_property()`와 `get_property()` 함수의 구현은 앞의 예제와 비슷하다. 이러한 함수들을 리스팅 11-20에 실었는데, 이들은 사용자가 위젯의 `message`와 `speed` 프로퍼티를 설정하고 검색하도록 해준다.

리스팅 11-20. `MyMarquee` 프로퍼티를 설정하고 검색하기

```

/* This function is called when the programmer gives a new value for a
widget
* property with g_object_set(). */
static void
my_marquee_set_property (GObject *object,
    guint prop_id,
    const GValue *value,
    GParamSpec *pspec)
{
    MyMarquee *marquee = MY_MARQUEE (object);
}

```

```

    switch (prop_id)
    {
        case PROP_MESSAGE:
            my_marquee_set_message (marquee, g_value_get_string
(value));
            break;
        case PROP_SPEED:
            my_marquee_set_speed (marquee, g_value_get_int (value));
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

/* This function is called when the programmer requests the value of a
widget
* property with g_object_get(). */
static void
my_marquee_get_property (GObject *object,
                        guint prop_id,
                        GValue *value,
                        GParamSpec *pspec)
{
    MyMarquee *marquee = MY_MARQUEE (object);
    MyMarqueePrivate *priv = MY_MARQUEE_GET_PRIVATE (marquee);

    switch (prop_id)
    {
        case PROP_MESSAGE:
            g_value_set_string (value, priv->message);
            break;
        case PROP_SPEED:
            g_value_set_int (value, priv->speed);
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

```

리스팅 11-21에는 `my_marquee_new()`의 구현을 실었다. 이 함수는 프로그래머가 새로운 `MyMarquee` 위젯을 생성할 때 호출할 수 있다. 단순한 편의 함수이므로 `g_object_new()`를 직접 호출하지 않아도 된다.

리스팅 11-21. 새로운 `MyMarquee` 위젯 생성하기

```

GtkWidget *
my_marquee_new ()

```



```
{
    return GTK_WIDGET (g_object_new (my_marquee_get_type (), NULL));
}
```

### 위젯 실현하기

위젯의 구현이 MyIPAddress와 다른 곳은 바로 오버라이드된 GtkWidgetClass 함수들이다. 이 함수들 중 처음으로 소개할 함수는 리스팅 11-22에 소개된 my\_marquee\_realize()다. 해당 함수는 MyMarquee 인스턴스가 처음으로 실현될 때 호출된다.

### 리스팅 11-22. MyMarquee 위젯 실현하기

```
static void
my_marquee_realize (GtkWidget *widget)
{
    MyMarquee *marquee;
    GdkWindowAttr attributes;
    gint attr_mask;

    g_return_if_fail (widget != NULL);
    g_return_if_fail (IS_MY_MARQUEE (widget));

    /* Set the GTK_REALIZED flag so it is marked as realized. */
    GTK_WIDGET_SET_FLAGS (widget, GTK_REALIZED);
    marquee = MY_MARQUEE (widget);

    /* Create a new GdkWindowAttr object that will hold info about the
    GdkWindow. */
    attributes.x = widget->allocation.x;
    attributes.y = widget->allocation.y;
    attributes.width = widget->allocation.width;
    attributes.height = widget->allocation.height;
    attributes.wclass = GDK_INPUT_OUTPUT;
    attributes.window_type = GDK_WINDOW_CHILD;
    attributes.event_mask = gtk_widget_get_events (widget);
    attributes.event_mask |= (GDK_EXPOSURE_MASK);
    attributes.visual = gtk_widget_get_visual (widget);
    attributes.colormap = gtk_widget_get_colormap (widget);

    /* Create a new GdkWindow for the widget. */
    attr_mask = GDK_WA_X | GDK_WA_Y | GDK_WA_VISUAL | GDK_WA_COLORMAP;
    widget->window = gdk_window_new (widget->parent->window, &attributes,
    attr_mask);
    gdk_window_set_user_data (widget->window, marquee);

    /* Attach a style to the GdkWindow and draw a background color. */
    widget->style = gtk_style_attach (widget->style, widget->window);
    gtk_style_set_background (widget->style, widget->window,
    GTK_STATE_NORMAL);
    gdk_window_show (widget->window);
```

```
}

```

my\_marquee\_realize()는 가장 먼저 위젯이 NULL이 아닌지 확인하고, 위젯이 MyMarquee 위젯인지 확인해야 한다. 두 테스트 중 하나라도 FALSE를 리턴할 경우 gtk\_return\_if\_fail() 함수를 이용해 함수로부터 리턴하도록 하였다. 이 테스트들은 항상 실행해야 하는데, 이를 어길 시 프로그램은 예기치 못한 반응을 보일 수 있다. 실현(realization) 함수를 사용하는 목적은 위젯의 인스턴스에 대해 GdkWindow를 준비하여 화면으로 렌더링이 가능하도록 만들기 위함이다. 이를 위해서는 먼저 새로운 GdkWindow의 원하는 프로퍼티를 보유한 GdkWindowAttr 객체가 필요하다. 표 11-3은 GdkWindowAttr 구조체의 모든 member를 설명하고 있다.

변수	설명
gchar *title	창의 제목. 최상위 수준의 창이 아닐 경우에는 NULL. 보통은 이 값을 설정할 필요가 없다.
gint event_mask	위젯이 실행하게 될 GDK 이벤트의 bitmask. gtk_widget_get_events()를 이용해 현재 위젯에 연관된 모든 이벤트를 검색한 후 자신만의 이벤트를 추가할 수 있다.
gint x, y	부모 창을 기준으로 GdkWindow 객체의 x와 y 좌표. 위젯의 할당으로부터 이 값을 검색할 수 있다.
gint width, height	GdkWindow 객체의 너비와 높이. 위젯의 할당에서 이 값을 검색할 수 있다.
GdkWindowClass wclass	대부분의 GdkWindow 객체에는 GDK_INPUT_OUTPUT으로 설정되고, 창이 눈에 보이지 않을 경우 GDK_INPUT_ONLY로 설정되어야 한다.
GdkVisual *visual	창에 사용될 GdkVisual 객체. 기본값은 gtk_widget_get_visual()로 검색 가능하다.
GdkColormap *colormap	창에 사용될 GdkColormap 객체. 기본값은 gdk_widget_get_colormap()으로 검색 가능하다.
GdkWindowType window_type	열거에 정의된 대로 표시될 창 타입.
GdkCursor *cursor	마우스가 위젯 위에 있을 때 표시될 선택적 GdkCursor 객체.
gchar *wmclass_name	이 프로퍼티는 무시해야 한다. 더 많은 정보는 gtk_window_set_wmclass()에 관한 문서를 참조한다.
gchar *wmclass_class	이 프로퍼티는 무시해야 한다. 더 많은 정보는 gtk_window_set_wmclass()에 관한 문서를 참조한다.
gboolean override_redirect	TRUE로 설정 시 위젯은 창 관리자를 건너뛴(bypass) 것이다.

표 11-3. GdkWindowAttr Members

my\_marquee\_realize()의 구현에서 우리는 먼저 부모 창의 상단 좌측 모서리를 기준으로 위젯의 가로 및 세로 위치를 설정하였다. 위젯의 할당에서 이미 그 위치를 제공하기 때문에 이 작업은 수월하다. 할당은 위젯의 초기 너비와 높이도 제공한다.

다음 member인 wclass는 두 개의 값 중 하나로 설정된다. GDK\_INPUT\_OUTPUT은 일반적인 GdkWindow 위젯을 나타내어서 대부분의 위젯에 사용된다. GDK\_INPUT\_ONLY는 이벤트를 수신하는 데에 사용되는 눈에 보이지 않는 GdkWindow 위젯에 해당한다. 그 다음으로 개발자는 창 타입을 설정할 수 있는데, 이는 아래의 GdkWindowType 열거에서 정의된 값으로 결정된다.

- GDK\_WINDOW\_ROOT: 부모 창이 없는 창으로, 전체 화면을 차지할 것이다. 창 관리자(window manager)만 이 값을 사용할 수 있다.
- GDK\_WINDOW\_TOPLEVEL: 보통 데코레이션이 있는 최상위 수준의 창이다. 이러한 창 타입을 사용하는 창의 예로 GtkWindow를 들 수 있다.
- GDK\_WINDOW\_CHILD: 최상위 수준 창의 자식 창이거나 어떤 자식 창의 자식 창에 해당한다. 최상위 수준 창이 아닌 대부분의 위젯에 사용된다.
- GDK\_WINDOW\_DIALOG: 이 타입은 오래되어 더 이상 사용되지 않으므로 사용해선 안 된다.
- GDK\_WINDOW\_TEMP: GtkMenu 위젯처럼 임시적으로만 표시될 창이다.
- GDK\_WINDOW\_FOREIGN: GdkWindow 위젯으로 래핑되어야 하는 다른 라이브러리에서 구현한 이질적인 창 타입이다.

다음 호출은 GdkWindow에 대한 이벤트 마스크를 설정한다. `gtk_widget_get_events()`를 호출하면 위젯에 이미 설치된 이벤트를 모두 리턴하고 `GDK_EXPOSURE_MASK`를 리스트로 추가한다. 그래야만 노출 함수가 호출 되도록 확보할 수 있을 것이다.

다음으로 GdkWindow 위젯에 사용될 GdkVisual 객체를 설정한다. 이 객체는 비디오 하드웨어에 관한 구체적인 정보를 설명하는 데에 사용된다. 대부분의 경우 위젯으로 할당된 기본 GdkVisual을 사용하길 원할 것인데, 이는 `gtk_widget_get_visual()`을 이용해 검색할 수 있다.

GdkWindowAttr 구조체에서 마지막으로 설정되는 프로퍼티는 색상 맵이다. 다시 말하지만 맵을 편집할 일은 별로 없을 것이기 때문에 `gdk_widget_get_colormap()`을 이용해 위젯에 대한 기본 맵을 검색하였다.

다음 단계는 GdkWindowAttr에서 어떤 필드가 honor되어야 하는지를 나타내는 구체적인

GdkWindowAttributesType 값의 마스크를 생성하는 일이다. 이번 예제에서는 명시된 x와 y 좌표인 GdkVisual, 그리고 GdkColormap이 사용될 것이다. `attributes_mask = GDK_WA_X | GDK_WA_Y | GDK_WA_VISUAL | GDK_WA_COLORMAP;`

이제 `gdk_window_new()`를 이용해 위젯에 대한 새로운 GdkWindow를 생성하기에 충분한 정보가 모였다. 이 함수는 부모 GdkWindow, GdkWindowAttr 객체, 그리고 honor해야 할 속성의 마스크를 수락한다.

```
GdkWindow* gdk_window_new (GdkWindow *parent,
    GdkWindowAttr *attributes,
    gint attributes_mask);
```

다음으로 `gdk_window_set_user_data()`를 이용해 GtkWidget를 커스텀 위젯에 대한 GdkWidget의 사용자 데이터로 저장해야 한다. 그래야만 expose-event와 같은 위젯 이벤트가 확실히 실현되도록 보장할 수 있다. 이 함수를 호출하지 않으면 이벤트는 실현되지 않을 것이다.

```
void gdk_window_set_user_data (GdkWindow *window,
    gpointer user_data);
```

창의 스타일은 `gtk_style_attach()`를 이용해 창으로 추가되는데, 이 함수를 호출하면 스타일을 위한 그래픽을 생성하는 과정이 시작된다. 리턴된 값이 새로운 스타일일 수도 있으니 항상 저장할 것을 명심한다.

```
GtkStyle* gtk_style_attach (GtkStyle *style,
    GdkWindow *window);
```

스타일이 창으로 추가되고 나면 창 배경색이 설정된다. `gtk_style_set_background()`는 GdkWindow의 배경색을 주어진 상태로 된 GtkStyle이 명시한 색상대로 설정한다.

```
void gtk_style_set_background (GtkStyle *style,
    GdkWindow *window,
    GtkStyleType state_type);
```

마지막으로 `gtk_window_show()`를 호출하여 창이 사용자에게 표시된다. 이 함수를 호출하지 않으면 위젯은 사용자에게 절대로 표시되지 않을 것이다. 이 함수는 모든 필요한 초기화가 실행되도록 보장한다.

## 크기 요청과 크기 할당 명시하기

우리는 부모 `GtkWindowClass`의 크기 요청과 할당 함수도 오버라이드하였다. 리스팅 11-23에 실린 `my_marquee_size_request()` 함수는 크기 요청에 대한 기본 너비 및 높이 값을 명시하는 데에 사용되었다.

## 리스팅 11-23. 크기 요청과 할당 처리하기

```

/* Handle size requests for the widget. This function forces the widget
   to have
   * an initial size set according to the predefined width and the font
   size. */
static void
my_marquee_size_request (GtkWidget *widget,
                        GtkRequisition *requisition)
{
    PangoFontDescription *fd;

    g_return_if_fail (widget != NULL || requisition != NULL);
    g_return_if_fail (IS_MY_MARQUEE (widget));

    fd = widget->style->font_desc;
    requisition->width = MARQUEE_MIN_WIDTH;
    requisition->height = (pango_font_description_get_size (fd) /
PANGO_SCALE) + 10;
}

/* Handle size allocations for the widget. This does the actual
   resizing of the
   * widget to the requested allocation. */
static void
my_marquee_size_allocate (GtkWidget *widget,
                        GtkAllocation *allocation)
{
    MyMarquee *marquee;

    g_return_if_fail (widget != NULL || allocation != NULL);
    g_return_if_fail (IS_MY_MARQUEE (widget));

    widget->allocation = *allocation;
    marquee = MY_MARQUEE (widget);

    if (GTK_WIDGET_REALIZED (widget))
    {
        gdk_window_move_resize (widget->window, allocation->x,
allocation->y,
                                allocation->width, allocation->height);
    }
}

```

크기 요청 함수는 초기 너비를 MARQUEE\_MIN\_WIDTH로 설정하는데, 이는 파일의 맨 위에서 설정되었다. 뿐만 아니라 높이가 글꼴 높이에 최소 10 픽셀을 더한 값이 되도록 강제로 설정한다. 그래야만 위젯에 일부 패딩과 함께 모든 메시지가 표시되도록 보장할 수 있다.

리스트 11-23의 할당 함수는 주어진 할당을 위젯으로 할당하면서 시작된다. 위젯이 실현되면 `gdk_window_move_resize()`를 호출한다. 이 함수는 `GdkWindow`의 크기를 조정하여 하나의 셀로 이동시키는 데에 사용할 수 있다. 그리고 창의 새로운 x 좌표, y 좌표, 너비, 높이를 비롯해 작업해야 할 `GdkWindow`도 수락한다.

```
void gdk_window_move_resize
(GdkWindow *window,
 gint x,
 gint y,
 gint width,
 gint height);
```

### 위젯 노출하기

`my_marquee_expose()` 함수에서 매우 흥미로운 일이 발생한다. 위젯이 먼저 사용자에게 표시될 때, 위젯의 크기가 조정되었을 때, 이전에 숨겼던 창의 일부가 표시될 때 이 함수가 호출된다. 이 함수를 리스트 11-24에 소개하였다.

### 리스트 11-24. MyMarquee 위젯 노출하기

```
static gint
my_marquee_expose (GtkWidget *widget,
                   GdkEventExpose *event)
{
    PangoFontDescription *fd;
    MyMarquee *marquee;
    MyMarqueePrivate *priv;
    PangoLayout *layout;
    PangoContext *context;
    gint width, height;

    g_return_val_if_fail (widget != NULL || event != NULL, FALSE);
    g_return_val_if_fail (IS_MY_MARQUEE (widget), FALSE);

    if (event->count > 0)
        return TRUE;

    marquee = MY_MARQUEE (widget);
    priv = MY_MARQUEE_GET_PRIVATE (marquee);
    fd = widget->style->font_desc;
    context = gdk_pango_context_get ();
    layout = pango_layout_new (context);
    g_object_unref (context);

    /* Create a new PangoLayout out of the message with the given font.
    */
    pango_layout_set_font_description (layout, fd);
```

```

pango_layout_set_text (layout, priv->message, -1);
pango_layout_get_size (layout, &width, &height);

/* Clear the text from the background of the widget. */
gdk_window_clear_area (widget->window, 0, 0,
widget->allocation.width,
widget->allocation.height);

/* Draw the PangoLayout on the widget, which is the message text.
*/
gdk_draw_layout (widget->window,
widget->style->fg_gc[widget->state],
priv->current_x,
(
widget->allocation.height - (height / PANGO_SCALE)) / 2,
layout);

return TRUE;
}

```

`pango_layout_new()`를 이용해 새로운 `PangoLayout`을 생성하는 일부터 시작했다. 이 레이아웃은 위젯에 텍스트를 그릴 때에 사용될 것이다. 해당 함수는 `PangoContext` 객체를 수락하는데, 기본 컨텍스트는 `gdk_pango_context_get()`을 이용해 검색하였다.

```
PangoLayout* pango_layout_new (PangoContext *context);
```

이러한 `PangoLayout`의 구현은 매우 간단하다. `pango_layout_set_text()`를 호출하면 레이아웃의 텍스트 내용이 `MyMarquee` 위젯의 `message` 프로퍼티로 설정된다. 텍스트의 너비와 높이는 `pango_layout_get_size()`를 호출하여 검색된다.

■ **Note** `pango_layout_get_size()`가 리턴한 너비와 높이 값은 `PANGO_SCALE`에 의해 스케일링된다. 따라서 그 값을 픽셀로 얻기 위해서는 정수를 스케일(scale)로 나누어야 할 것이다.

`PangoLayout`이 준비되고 나면 전체 위젯이 제거되고 위젯을 그릴 준비를 시킨다. 이 때 `gdk_window_clear_area()`를 호출하면 (x,y) 좌표부터 (x + width,y + height)까지 영역을 제거한다.

```

void gdk_window_clear_area (GdkWindow *window,
gint x,
gint y,
gint width,
gint height);

```

영역을 제거하고 나면 `gdk_draw_layout()`을 이용해 화면에 레이아웃을 그릴 수 있다. 이 함수는 먼저 그려야 할 `GdkDrawable` 객체, 즉 `GdkWindow`를 수락한다. 두 번째 매개변수는 사용할 그래픽 컨텍스트로, 클래스의 `GtkStyle` member에 의해 저장된다.

```

void gdk_draw_layout (GdkDrawable *drawable,
GdkGC *gc,
gint x,
gint y,
PangoLayout *layout);

```

마지막으로 레이아웃을 그릴 x와 y 좌표를 명시해야 한다. 이 위치들은 창 안에 있을 필요가 없음을 기억하라. 처음에는 레이아웃이 위젯의 우측에 그려지기 때문에 왼쪽으로 스크롤이 가능하다. 좌측의 뷰에서 완전히 숨겨지고 나서야 초기 위치로 리셋될 것이다. 따라서 스키프 주기(scrolling cycle) 끝에 x 좌표는 사실상 음수가 될 것이다.

그리기 함수

PangoLayout 객체를 GdkWindow 객체로 그리는 기능 외에도 GDK는 GdkDrawable 객체를 통해 수많은 기본 그리기 함수를 제공한다. 전체 함수의 리스트는 GDK API 문서에서 찾아볼 수 있다. 표 11-4에는 필요한 함수를 쉽게 찾을 수 있도록 몇 가지를 열거하였다

함수	설명
gdk_draw_arc()	(x,y)에서 시작해 (x + width,y + height)에서 끝나는 호(arc)를 그린다. 호에 색상을 채울 것인지 선택할 수 있다. 시작 각도와 끝 각도를 1/64도로 명시해야 한다.
gdk_draw_drawable()	때로는 자신의 GdkDrawable에 그리기가 가능한 영역의 특정 일부를 복사하는 것이 바람직하다. 이 함수는 복사할 영역에서 그리기 가능한 소스의 영역을 명시하도록 해준다.
gdk_draw_image()	소스 GdkImage 객체의 일부를 그리기가 가능한 영역에 그린다. GdkDrawable 객체를 GdkImage 객체로 변환할 수 있으므로 이 함수는 사실상 그리기가 가능한 소스가 될 수 있다.
gdk_draw_layout()	PangoLayout이 정의한 구체적인 텍스트의 문자 수만큼 그린다. GdkDrawable에 텍스트를 위치시킬 때 사용된다.
gdk_draw_layout_line()	gdk_draw_layout()과 비슷하지만 PangoLayout에서 PangoLayoutLine이라는 단일 행을 그릴 수 있다는 점에 차이가 있다.
gdk_draw_line()	시작점부터 끝점까지 직선을 그린다. 그래픽 컨텍스트의 전경색을 이용해 그려질 것이다.
gdk_draw_lines()	GdkPoint 배열에 명시된 끝점을 이용해 일련의 선을 그린다. 배열 내 점의 개수를 명시해야 한다.
gdk_draw_pixbuf()	GdkDrawable 객체에 GdkPixbuf 이미지의 일부를 그린다. 이미지를 렌더링할 때 이용될 추가 매개변수도 명시해야 한다.
gdk_draw_point()	그래픽 컨텍스트에 명시된 전경색을 이용해 화면에 하나의 점을 그린다. 개발자는 점에 대한 x 좌표와 y 좌표를 제공하면 된다.
gdk_draw_points()	GdkPoint 객체의 배열에 명시된 다수의 점을 화면으로 그린다. GdkPoint 구조체는 x 와 y 좌표를 보유한다. 개발자는 배열에서 점의 개수 또한 명시해야 한다.
gdk_draw_polygon()	GdkPoint 객체의 배열에 열거된 점을 연결하는 다각형을 그린다. 필요하다면 마지막 점이 처음 점으로 연결될 것이다. 다각형의 채움 여부를 선택할 수 있다.
gdk_draw_rectangle()	gdk_draw_polygon()과 유사하지만 결과적인 모양이 항상 직사각형이라는 점에서 차이가 있다. 개발자는 y 좌표, y 좌표, 너비, 높이를 비롯해 직사각형의 채움 여부를 명시해야 한다.
gdk_draw_segments()	연결되지 않은 라인 세그먼트(line segment)를 여러 개 그린다. 각 라인 세그먼트는 시작 좌표와 끝 좌표를 보유하는 GdkSegment 객체에 저장된다. 이 함수에 GdkSegment 객체의 배열이 제공된다.
gdk_draw_trapezoids()	GdkTrapezoid 객체의 배열에 저장된 부등변 사각형(trapezoid)을 여러 개 그린다. GdkTrapezoid 구조체는 시작점과 끝점에 대한 y 좌표를 보유한다. 부등변 사각형의 각 모서리에 해당하는 4개의 x 좌표 또한 보유한다.

표 11-4. GdkDrawable 함수

**Public** 함수 구현하기

MyMarquee 위젯에는 다수의 public 함수가 포함되어 있다. 그 중에서 가장 중요한 함수는 my\_marquee\_slide() 인데, 이 함수를 호출하면 메시지를 speed 픽셀만큼 좌측으로 이동시킬 것이다. 프로그래머는 이 함수를 timeout으로 추가함으로써 명시된 시간 간격으로 함수를 호출하여 marquee 효과를 야기할 수 있다.

리스팅 11-25. MyMarquee 메시지 슬라이딩하기

```
void
my_marquee_slide (MyMarquee *marquee)
{
    PangoFontDescription *fd;
    GtkWidget *widget;
    MyMarqueePrivate *priv;
    PangoLayout *layout;
    PangoContext *context;
    gint width, height;

    g_return_if_fail (marquee != NULL);
    g_return_if_fail (IS_MY_MARQUEE (marquee));

    widget = GTK_WIDGET (marquee);
    priv = MY_MARQUEE_GET_PRIVATE (marquee);
    fd = widget->style->font_desc;
    context = gdk_pango_context_get ();
    layout = pango_layout_new (context);
    g_object_unref (context);

    /* Create a new PangoLayout out of the message with the given font.
    */
    pango_layout_set_font_description (layout, fd);
    pango_layout_set_text (layout, priv->message, -1);
    pango_layout_get_size (layout, &width, &height);

    /* Clear the text from the background of the widget. */
    gdk_window_clear_area (widget->window, 0, 0,
widget->allocation.width,
        widget->allocation.height);

    /* Scroll the message "speed" pixels to the left or wrap around. */
    priv->current_x = priv->current_x - priv->speed;
    if ((priv->current_x + (width / PANGO_SCALE)) <= 0)
        priv->current_x = widget->allocation.width;

    /* Draw the PangoLayout on the widget, which is the message text.
    */
    gdk_draw_layout (widget->window,
        widget->style->fg_gc[widget->state],
        priv->current_x,
```



```

        (widget->allocation.height - (height / PANGO_SCALE)) / 2,
        layout);
}

```

이 함수는 앞서 구현한 노출 함수와 매우 유사함을 눈치챌 것이다. 두 함수 간 차이점을 살펴보자.

이 함수는 텍스트의 새로운 위치, 즉 현재 위치에서 speed 프로퍼티 값을 뺀 값을 계산해야 한다. 다음으로 메시지가 화면에 여전히 표시되는지 확인해야 한다. 위젯의 왼쪽 범위 밖으로 이동했다면 위치는 위젯의 우측면으로 리셋되어 스크롤링 메시지를 순환(loop)한다. 이 시점에서는 my\_marquee\_expose()에서와 동일한 방식으로 그리기가 이루어진다.

**Tip** PangoLayout으로부터 검색한 높이와 너비 값을 픽셀로 되어 있지 않음을 기억하라. 값을 픽셀로 검색하려면 값을 PANGO\_SCALE로 나누어야 한다!

마지막으로 우리는 MyMarquee 위젯의 speed 와 message 프로퍼티를 설정 및 검색하는 기능을 제공해야 한다. 이러한 프로퍼티로 접근하려면 MY\_MARQUEE\_GET\_PRIVATE()을 이용해 private 데이터 구조체를 검색해야 함을 기억해야 한다.

리스트 11-26. Message와 Speed를 설정하고 검색하기

```

/* Set the message that is displayed by the widget. */
void
my_marquee_set_message (MyMarquee *marquee,
                        const gchar *message)
{
    MyMarqueePrivate *priv = MY_MARQUEE_GET_PRIVATE (marquee);

    if (priv->message)
    {
        g_free (priv->message);
        priv->message = NULL;
    }

    priv->message = g_strdup (message);
}

/* Retrieve the message that is displayed by the widget. You must free
this
* string after you are done using it! */
gchar*
my_marquee_get_message (MyMarquee *marquee)
{
    return g_strdup (MY_MARQUEE_GET_PRIVATE (marquee)->message);
}

/* Set the number of pixels that the message will scroll. */
void
my_marquee_set_speed (MyMarquee *marquee,
                      gint speed)
{
    MyMarqueePrivate *priv = MY_MARQUEE_GET_PRIVATE (marquee);
}

```

```

    priv->speed = speed;
}

/* Retrieve the number of pixels that the message will scroll. */
gint
my_marquee_get_speed (MyMarquee *marquee)
{
    return MY_MARQUEE_GET_PRIVATE (marquee)->speed;
}

```

### 위젯 테스트하기

위젯 소스가 작성되었으니 이제 위젯을 테스트할 때가 되었다. 작은 테스트 애플리케이션을 리스팅 11-27에 실겠다. Timeout이 추가되어 150 밀리 초마다 `my_marquee_slide()`를 호출할 것이다.

`marquee`는 "Wheeeee!"라는 초기 메시지로 설정되었고, `my_marquee_slide()`를 호출할 때마다 좌측으로 10 픽셀만큼 이동할 것이다.

리스팅 11-27. MyMarquee 위젯 테스트하기 (marqueetest.c)

```

#include <gtk/gtk.h>
#include "mymarquee.h"

int main (int argc,
          char *argv[])
{
    GtkWidget *window, *marquee;
    PangoFontDescription *fd;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "MyMarquee");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    g_signal_connect (G_OBJECT (window), "destroy",
                     G_CALLBACK (gtk_main_quit), NULL);

    fd = pango_font_description_from_string ("Monospace 30");
    marquee = my_marquee_new ();
    gtk_widget_modify_font (marquee, fd);
    my_marquee_set_message (MY_MARQUEE (marquee), "Wheeeee!");
    my_marquee_set_speed (MY_MARQUEE (marquee), 10);
    pango_font_description_free (fd);

    g_timeout_add (150, (GSourceFunc) my_marquee_slide, (gpointer)
marquee);

    gtk_container_add (GTK_CONTAINER (window), marquee);
    gtk_widget_show_all (window);
}

```

```

gtk_main ();
return 0;
}

```

## 인터페이스 구현하기

앞의 여러 장에 걸쳐 GtkEditable, GtkFileChooser, GtkTreeModel, GtkRecentChooser를 포함해 여러 가지 인터페이스를 소개하였다. GObject의 인터페이스는 Java의 것과 매우 비슷하다. 새로운 인터페이스는 리스팅 11-28에서 볼 수 있듯이 GTypeInterface로부터 상속된다.

**Note** 이번 절에 실린 코드는 인터페이스를 사용 시 꼭 필요한 것이 무엇인지 설명하기 위해 매우 기본적인 인터페이스와 객체를 구현한다. 실용적으로 사용하기 위해서는 훨씬 더 많은 API를 포함하도록 크게 확장되어야 할 것이다.

리스팅 11-28. 인터페이스 헤더 파일 (myiface.h)

```

#ifndef __MY_IFACE_H__
#define __MY_IFACE_H__

#include <gtk/gtk.h>

G_BEGIN_DECLS

#define MY_TYPE_IFACE (my_iface_get_type ())
#define MY_IFACE(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
    GTK_TYPE_IFACE, MyIFace))
#define MY_IS_IFACE(obj) (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
    GTK_TYPE_IFACE))
#define MY_IFACE_GET_INTERFACE(inst) (G_TYPE_INSTANCE_GET_INTERFACE \
    ((inst), \
    MY_TYPE_IFACE, MyIFaceInterface))

typedef struct _MyIFace MyIFace;
typedef struct _MyIFaceInterface MyIFaceInterface;

struct _MyIFaceInterface
{
    GObject parent;

    void (*print_message) (MyIFace *obj, gchar *message);
};

GType my_iface_get_type ();
void my_iface_print_message (MyIFace *obj, gchar *message);

G_END_DECLS

#endif /* __MY_IFACE_H__ */

```

myiface.h 헤더 파일에는 새로운 위젯을 생성할 때와 동일한 함수와 구조체가 상당히 많이 포함되어 있음을 눈치챌 것이다. 정의부는 네 개가 있는데 이들은 각각 인터페이스의 GType을 리턴하고, 인터페이스를 캐스팅하고, 유효한 GTK\_TYPE\_IFACE인지 확인하며, 연관된 인터페이스를 리턴한다.

인터페이스를 선언할 때는 MyIFace 구조체에 대한 타입 정의를 선언해야 하지만 이는 사실 MY\_IFACE()가 작동하도록 허용하는 opaque 타입에 불과하다. MyIFaceInterface 야말로 인터페이스의 실제 내용에 해당한다. 이는 모든 인터페이스의 부모 타입인 GTypeInterface 객체를 포함해야 한다.

이는 하나 또는 이상의 함수 포인터를 포함하기도 한다. 객체가 주어진 인터페이스를 구현하면 프로그래머는 이러한 함수들을 오버라이드한다. 그래야만 각 객체가 고유의 방식으로 인터페이스를 구현하는 동시 여러 객체들 간 명명 규칙에 일관성을 제공할 수 있다.

### 인터페이스 구현하기

리스팅 11-29는 매우 기본적인 MyIFace 소스 파일의 구현을 신고 있다. 이는 새로운 인터페이스 GType의 등록, 인터페이스 클래스의 초기화, member 함수의 호출을 위한 함수들을 제공한다.

리스팅 11-29. 인터페이스 소스 파일 (myiface.c)

```
#include "myiface.h"

static void my_iface_class_init (gpointer iface);

GType
my_iface_get_type ()
{
    static GType type = 0;

    if (!type)
    {
        type = g_type_register_static_simple (G_TYPE_INTERFACE,
        "MyIFace",
        sizeof (MyIFaceInterface),
        (GClassInitFunc) my_iface_class_init,
        0, NULL, 0);

        g_type_interface_add_prerequisite (type, GTK_TYPE_WIDGET);
    }

    return type;
}

static void
my_iface_class_init (gpointer iface)
{
    GType iface_type = G_TYPE_FROM_INTERFACE (iface);

    /* Install signals & properties here ... */
}

void
```

```
my_iface_print_message (MyIFace *obj,
                        gchar *message)
{
    MY_IFACE_GET_INTERFACE (obj)->print_message (obj, message);
}
```

리스트 11-29의 첫 번째 함수는 MyIFace 타입을 등록하기 위해 사용되었다. 이 때 g\_type\_register\_static\_simple()을 이용하였다. 이 함수는 먼저 부모에 해당하는 GType과 새로운 타입의 이름을 수락한다. 부모 타입은 인터페이스에 대한 G\_TYPE\_INTERFACE에 해당한다. 세 번째 매개변수는 인터페이스 구조체의 크기로, sizeof() 함수를 이용해 얻을 수 있다.

```
GType g_type_register_static_simple (GType parent_type,
    const
    gchar *type_name,
    guint class_size,
    GClassInitFunc class_init,
    guint instance_size,
    GInstanceInitFunc instance_init,
    GTypeFlags flags);
```

다음으로는 클래스 초기화 함수를 명시해야 한다. 인스턴스 크기와 인스턴스 초기화 함수 모두 무시될 수도 있는데, 인스턴스 구조체가 opaque 타입이기 때문이다. 마지막 매개변수는 GTypeFlags의 bitwise 필드로, 인터페이스에 대해 안전하게 0으로 설정 가능하다.

인터페이스를 구현하는 객체라면 prerequisite\_type 또한 강제로 구현하도록 g\_type\_interface\_add\_prerequisite() 라는 함수를 이용하였다. 인터페이스는 기껏해야 하나의 전제조건 (prerequisite)만 가질 수 있다.

```
void g_type_interface_add_prerequisite (GType interface_type,
    GType prerequisite_type);
```

클래스 초기화 함수는 여느 GObject 클래스 초기화 함수와 다를 게 없다. 인터페이스가 필요로 하는 시그널과 프로퍼티를 준비할 때는 이 함수를 이용해야 한다. 이러한 함수들을 인터페이스로 추가한다는 것은 이 인터페이스를 구현하는 클래스라면 어디서든 이용할 수 있을 것이라 의미다.

마지막 함수 my\_iface\_print\_message()는 단순히 현재 MyIFaceInterface 인스턴스에 위치한 함수를 호출하는 public 함수다. 이는 인터페이스를 구현하고 있는 객체에 의해 추가된 함수의 인스턴스를 호출할 것이라 의미다.

### 인터페이스 사용하기

객체에서 인터페이스를 구현하기란 매우 간단한 작업이다. 가장 먼저 자신의 GType 등록 함수에 두 가지를 추가하면 된다. 리스트 11-30에는 MyObject라는 가상의 클래스에 이 함수를 사용하는 예제를 실었다. 이 객체는 인터페이스의 사용법이 얼마나 쉬운지 보이도록 객체의 가장 기본 요소들만 포함한다.

### 리스트 11-30. 객체의 GType 생성하기

```
GType
my_object_get_type (void)
{
    static GType type = 0;

    if (!type)
    {
```

```

static const GTypeInfo info =
{
    sizeof (MyObjectClass),
    NULL,
    NULL,
    (GClassInitFunc) my_object_class_init,
    NULL,
    NULL,
    sizeof (MyObject),
    0,
    (GInstanceInitFunc) my_object_init,
};

static const GInterfaceInfo iface_info =
{
    (GInterfaceInitFunc) my_object_interface_init,
    NULL,
    NULL
};

type = g_type_register_static (GTK_TYPE_WIDGET, "MyObject",
&info, 0);
g_type_add_interface_static (type, MY_TYPE_INTERFACE,
&iface_info);
}

return type;
}

```

이 함수는 가장 먼저 `GInterfaceInfo` 객체를 선언한다는 점에서 여태까지 소개한 함수와 다르다. 이 구조체는 세 가지 정보 조각을 보유한다. 두 개의 member는 인터페이스가 초기화될 때와 최종화될 때 호출되는 `GInterfaceInitFunc`와 `GInterfaceFinalizeFunc` 함수다. 세 번째 member는 각 함수로 전달되는 데이터의 포인터다. 처음 두 개의 member는 무시해도 안전하다.

두 번째 차이점은 인터페이스를 인스턴스 타입으로 추가하는 데에 사용되는 `g_type_add_interface_static()`을 호출한다는 데에 있다. 이 함수는 세 개의 매개변수를 수락하는데, 바로 인스턴스 `GType`, 인터페이스 `GType`, 이전에 정의된 `GInterfaceInfo` 객체가 그것이다.

```

void g_type_add_interface_static (GType instance_type,
    GType interface_type,
    const GInterfaceInfo *info);

```

리스트링 11-31은 `MyIFace` 인터페이스를 구현하는 과정의 마지막 두 단계를 보여준다. 첫 번째 함수 `my_object_print_message()`는 `MyIFaceInterface` member가 가리킬 `print_message()` 함수의 실제 구현이다. 해당 함수는 프로그래머가 `my_iface_print_message()`를 호출하면 호출될 것이다.

리스트링 11-31. 인터페이스 초기화하기

```

static void
my_object_print_message (MyObject *object,
    gchar *message)

```

```

{
    g_print (message);
}

static void
my_object_interface_init (gpointer iface,
                          gpointer data)
{
    MyIFaceInteface *iface = (MyIFaceInterface*) iface;
    iface->print_message =
        (void (*) (MyIFace *obj, gchar *message))
my_object_print_message;
}
    
```

리스트링 11-31에 실린 두 번째 함수는 객체의 인터페이스 초기화 함수의 구현이다. 이는 단순히 MyIFaceInterface의 print\_message() member를 함수에 대한 객체의 구현으로 가리킬 뿐이다.

위의 리스트링은 매우 간단한 인터페이스 구현 예제에 해당하지만 좀 더 복잡한 예제를 만들 때 필요한 기본 요소들을 모두 가르쳐준다. 지금쯤이면 다른 GObject에서 자신만의 객체를 상속하는 법을 비롯해 자신만의 인터페이스를 생성 및 구현할 수 있을 것인데, 이 정도면 꽤 큰 성과가 아닌가! 다음 장에서는 GTK+에 이미 빌드되어 있는 위젯에 관한 학습으로 다시 돌아가겠다.

### 자신의 이해도 시험하기

이번 장에 실린 연습문제에서는 몇 가지 새로운 기능을 포함하도록 MyMarquee 위젯을 확장시킬 것이다. 이를 위해서는 코드의 많은 부분을 편집하고 API 문서의 새로운 함수들을 살펴볼 필요가 있다. 또 message-changed 시그널과 같이 연습문제에 실리지 않았지만 위젯에 자신만의 개선사항(enhancement)을 추가하는 것도 고려해보도록 한다!

#### 연습문제 11-1. MyMarquee 확장하기

이번 연습문제에서는 MyMarquee에 몇 가지 새로운 기능을 확장한다. 먼저 프로그래머는 스크롤 방향이 좌측인지 우측인지 명시할 수 있어야 한다. 또 위젯 주변에 직사각형으로 된 테두리를 위치시켜라. 그 외 프로퍼티인 메시지는 이제 순환(cycled)되는 메시지의 리스트여야 한다. 초기 메시지는 my\_marquee\_new()에서 설정할 수 있어야 한다.

그에 이어 마우스가 위젯의 주변으로 들어서면 호출되는 오버라이드 함수를 구현하라. 마우스가 위젯 주위로 들어가면 마우스 커서가 그 주위를 벗어날 때까지 메시지의 스크롤이 중단되어야 한다. 이를 구현하기 위해서는 GdkWindow 객체에 새로운 이벤트 마스크를 추가해야 할 것이다.

### 요약

이번 장에서는 새로운 객체를 상속하는 방법을 가르치기 위해 두 가지 예를 살펴보았다. 처음에 소개한 예제는 GtkEntry에서 상속된 MyIPAddress라는 새로운 위젯을 생성하였다. 두 번째 새로운 위젯은 MyMarquee 위젯으로서 화면에 걸쳐 메시지를 스크롤한다. 이 예제는 화면에 위젯을 한 부분씩 그림으로써 말 그대로 처음부터 새 위젯을 생성하는 방법을 가르쳐주었다.

다음으로 GTK+에서 인터페이스를 구현하고 사용하는 방법을 소개하였다. 이를 통해 개발자는 본인이 생성한 새로운 위젯에 이미 존재하는 인터페이스를 사용하거나 자신만의 인터페이스를 생성할 수 있다.

다음 장에서는 앞 장에서 다루지 않았던 위젯들을 다수 학습할 것이다. 이러한 위젯으로는 인쇄(printing) 위젯, 최근 파일 지원, 캘린더(calendar), 자동 완성 엔트리, 상태 아이콘, 그리기 영역이 있다.

Notes

# FoundationsofGTKDevelopment:Chapter 12

## 제 12 장 기타 GTK+ 위젯들

### 기타 GTK+ 위젯들

본 서적에서 가르치고자 하는 내용은 대부분 마쳤다. 하지만 앞에서 다룬 주제에 해당하지 않는 위젯들도 아직 많다. 따라서 이번 장에서는 그러한 위젯들을 다루고자 한다.

가장 먼저 소개할 두 개의 위젯은 그리기에 사용되는 `GtkDrawingArea`와 `GtkLayout`이다. 이 두 위젯은 매우 비슷하지만 `GtkLayout` 위젯의 경우 임의의 위젯을 자체로 포함(embed)시키도록 허용함과 동시에 그리기용 함수를 이용한다는 점에서 차이가 있다.

또 자동 완성과 캘린더를 지원하는 `GtkEntry`에 대해서도 배워볼 참이다. 마지막으로 상태 아이콘, 인쇄 지원, 최근 파일 관리자를 포함해 GTK+ 2.10에 추가된 위젯을 소개하겠다.

이번 장에서는 다음과 같은 내용을 학습할 것이다.

- 그리기 위젯 `GtkDrawingArea`와 `GtkLayout`을 사용하는 방법
- `GtkCalendar`를 이용해 연(year)의 월(months)에 대한 정보를 추적하는 방법
- 최근 파일 추적, 인쇄 지원, 상태 아이콘을 제공하는 GTK+ 2.10에 도입된 위젯의 사용 방법
- `GtkEntryCompletion` 객체를 적용함으로써 `GtkEntry` 위젯에서 자동 완성을 구현하는 방법

### 그리기 위젯

앞 장에서 `GdkWindow`에 도형과 텍스트를 그릴 수 있도록 해주는 `GdkDrawable` 객체에 대해 학습한 바 있다. GTK+는 `GtkDrawingArea` 위젯을 제공하는데, 이는 당신이 그릴 수 있는 빈 슬레이트에 불과하다.

`GtkDrawingArea`에서는 아직 사용이 가능한(nondeprecated) 함수로 `gtk_drawing_area_new()`라는 함수 하나만 제공하는데, 이는 매개변수를 수락하지 않고 새로운 그리기 영역 위젯을 리턴한다.

```
GtkWidget* gtk_drawing_area_new ();
```

위젯을 이용하기 위해서는 앞 장에서 위젯의 `GdkWindow`에 그릴 때 이용했던 함수를 이용하면 된다.

`GdkWindow` 객체는 `GdkDrawable` 객체가 되기도 한다는 사실을 기억한다.

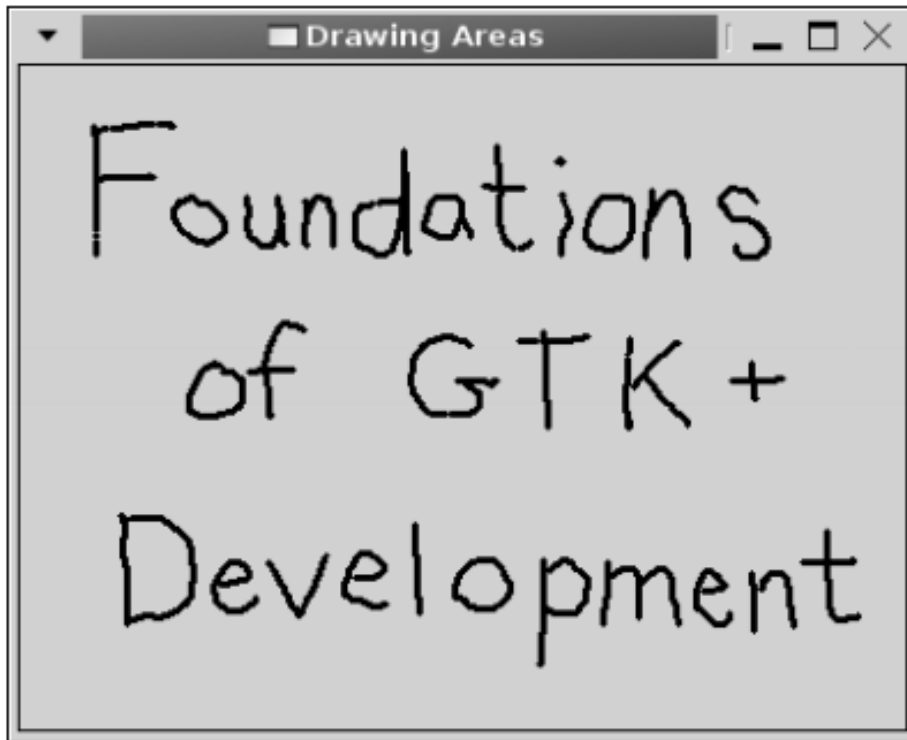
`GdkDrawingArea`는 `GtkWidget`에서 파생되어 GDK 이벤트로 연결될 수 있다는 한 가지 장점이 있다. 개발자는 그리기 영역으로 많은 이벤트를 연결하길 원할 것이다. 가장 먼저 `realize`로 연결해야 위젯이 인스턴스화될 때 GDK 자원의 생성 등 실행해야 하는 작업을 처리할 수 있을 것이다. `configure-event` 시그널은 위젯의 크기 변경을 처리해야 할 때를 알려줄 것이다. 또 `expose-event` 는 이전에 숨겨져 있던 부분이 노출되면 위젯을 다시 그리도록 해준다. `expose-event` 시그널이 특히 중요한데, 그리기 영역 내용이 `expose-event` 콜백에 걸쳐 지속 되길 원한다면 그 내용을 다시 그려야 하기 때문이다. 마지막으로 버튼과 마우스 클릭 이벤트를 연결하여 사용자와 위젯의 상호작용이 가능하게 할 수 있다.

■ **Note** 특정 타입의 이벤트를 수신하기 위해서는 `gtk_widget_add_events()`가 지원하는 위젯 이벤트의 리스트에 추가해야 한다. 또 사용자로부터 키보드 입력을 수신하기 위해서는 `GTK_CAN_FOCUS` 플래그를 설정해야 하는데, 포커스가 있는 위젯만 키 누름을 감지할 수 있기 때문이다.



## 그리기 영역의 예제

리스트 12-1은 GtkDrawingArea 위젯을 이용해 간단한 그리기 프로그램을 구현한다. 사용자가 버튼을 클릭하고 버튼을 누른 상태에서 포인터를 드래그하면 화면에 점이 그려질 것이다. 이러한 애플리케이션의 스크린샷은 그림 12-1에서 확인할 수 있다.



그리기 영역의 GdkWindow 객체에서 현재 내용은 사용자가 Delete 키를 누르면 삭제된다. 매우 간단한 프로그램이지만 GtkDrawingArea 위젯과 상호작용하는 방법을 비롯해 이 위젯과 함께 이벤트를 이용하는 방법을 보여준다.

리스트 12-1. 간단한 그리기 프로그램 (drawingareas.c)

```
#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>

static gboolean button_pressed (GtkWidget*, GdkEventButton*,
GPtArray*);
static gboolean motion_notify (GtkWidget*, GdkEventMotion*,
GPtArray*);
static gboolean key_pressed (GtkWidget*, GdkEventKey*, GPtArray*);
static gboolean expose_event (GtkWidget*, GdkEventExpose*, GPtArray*);

int main (int argc,
char *argv[])
{
    GtkWidget *window, *area;
    GPtArray *parray;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```

gtk_window_set_title (GTK_WINDOW (window), "Drawing Areas");
gtk_widget_set_size_request (window, 400, 300);

g_signal_connect (G_OBJECT (window), "destroy",
                 G_CALLBACK (gtk_main_quit), NULL);

/* Create a pointer array to hold image data. Then, add event masks
to the new
 * drawing area widget. */
parray = g_ptr_array_sized_new (5000);
area = gtk_drawing_area_new ();
GTK_WIDGET_SET_FLAGS (area, GTK_CAN_FOCUS);
gtk_widget_add_events (area, GDK_BUTTON_PRESS_MASK |
                     GDK_BUTTON_MOTION_MASK |
                     GDK_KEY_PRESS_MASK);

g_signal_connect (G_OBJECT (area), "button_press_event",
                 G_CALLBACK (button_pressed), parray);
g_signal_connect (G_OBJECT (area), "motion_notify_event",
                 G_CALLBACK (motion_notify), parray);
g_signal_connect (G_OBJECT (area), "key_press_event",
                 G_CALLBACK (key_pressed), parray);
g_signal_connect (G_OBJECT (area), "expose_event",
                 G_CALLBACK (expose_event), parray);

gtk_container_add (GTK_CONTAINER (window), area);
gtk_widget_show_all (window);

/* You must do this after the widget is visible because it must
first
 * be realized for the GdkWindow to be valid! */
gdk_window_set_cursor (area->window, gdk_cursor_new (GDK_PENCIL));

gtk_main ();
return 0;
}

/* Redraw all of the points when an expose-event occurs. If you do not
do this,
 * the drawing area will be cleared. */
static gboolean
expose_event (GtkWidget *area,
             GdkEventExpose *event,
             GPtrArray *parray)
{
    guint i, x, y;
    GdkPoint points[5];

```

```

    /* Loop through the coordinates, redrawing them onto the drawing
area. */
    for (i = 0; i < parray->len; i = i + 2)
    {
        x = GPOINTER_TO_INT (parray->pdata[i]);
        y = GPOINTER_TO_INT (parray->pdata[i+1]);

        points[0].x = x; points[0].y = y;
        points[1].x = x+1; points[1].y = y;
        points[2].x = x-1; points[2].y = y;
        points[3].x = x; points[3].y = y+1;
        points[4].x = x; points[4].y = y-1;

        gdk_draw_points (area->window,
                        area->style->fg_gc[GTK_WIDGET_STATE (area)],
                        points, 5);
    }

    return TRUE;
}

/* Draw a point where the user clicked the mouse and points on each of
the
* four sides of that point. */
static gboolean
button_pressed (GtkWidget *area,
               GdkEventButton *event,
               GPtrArray *parray)
{
    gint x = event->x, y = event->y;
    GdkPoint points[5] = { {x,y}, {x+1,y}, {x-1,y}, {x,y+1}, {x,y-1} };

    gdk_draw_points (area->window,
                    area->style->fg_gc[GTK_WIDGET_STATE (area)],
                    points, 5);

    g_ptr_array_add (parray, GINT_TO_POINTER (x));
    g_ptr_array_add (parray, GINT_TO_POINTER (y));

    return FALSE;
}

/* Draw a point where the moved the mouse pointer while a button was
* clicked along with points on each of the four sides of that point. */
static gboolean
motion_notify (GtkWidget *area,

```

```

        GdkEventMotion *event,
        GPtrArray *parray)
{
    gint x = event->x, y = event->y;
    GdkPoint points[5] = { {x,y}, {x+1,y}, {x-1,y}, {x,y+1}, {x,y-1} };

    gdk_draw_points (area->window,
                    area->style->fg_gc[GTK_WIDGET_STATE (area)],
                    points, 5);

    g_ptr_array_add (parray, GINT_TO_POINTER (x));
    g_ptr_array_add (parray, GINT_TO_POINTER (y));

    return FALSE;
}

/* Clear the drawing area when the user presses the Delete key. */
static gboolean
key_pressed (GtkWidget *area,
            GdkEventKey *event,
            GPtrArray *parray)
{
    if (event->keyval == GDK_Delete)
    {
        gdk_window_clear (area->window);
        g_ptr_array_remove_range (parray, 0, parray->len);
    }

    return FALSE;
}

```

리스팅 12-1에서 몇 가지 눈에 띄는 점이 발견될 것이다. 먼저 그리기 영역에 추가된 점을 추적하기 위해 `GPtrArray`가 사용되었다. 점이 화면에 추가되면 처음 점의 네 면에 있는 점이 모두 활성화된다. 이후 수평 및 수직 위치가 배열로 추가된다. `expose-event` 콜백 함수가 호출되면 점들이 모두 다시 그려진다. 그리기 영역의 내용을 개발자가 다시 그리지 않으면 `expose-event` 발생 중에 삭제될 것이다.

점을 그리기 위해 이 애플리케이션은 `gdk_draw_points()`를 이용한다. 이 함수는 그리기 가능한 객체에 `npoints` 점의 배열을 그린다. 이는 현재 위젯의 상태에 대한 기본 전경색을 이용한다.

```

void gdk_draw_points (GdkDrawable *drawable,
                    GdkGC *gc,
                    GdkPoint *points,
                    gint npoints);

```

`gdk_draw_points()` 외에도 제 11장에 열거된 그리기 함수 중 어느 것이든 그리기 영역과 함께 이용할 수 있다.

레이아웃 위젯

GtkDrawingArea 외에도 GTK+는 GtkLayout이라는 그리기 위젯을 제공한다. 이 위젯은 사실상 컨테이너로서, 그리기 요소(drawing primitive) 뿐만 아니라 자식 위젯도 지원한다는 점에서 GdkDrawingArea와 다르다. 게다가 GtkLayout은 본래부터 스크롤링 지원을 제공하므로 스크롤이 있는 창에 추가 시 뷰포트가 필요 없다.

■ **Note** 레이아웃과 관련해 한 가지 주목해야 할 차이점은 GtkWidget의 window 대신 GtkLayout의 bin\_window member로 그려야 한다는 점이다. 예를 들자면, GTK\_WIDGET(layout)->window 대신 GTK\_LAYOUT(layout)->bin\_window로 그려야 할 것이다. 이런 경우 위젯에 자식 위젯을 올바르게 포함하도록 해준다.

새로운 GtkLayout 위젯은 gtk\_layout\_new()를 이용해 생성되는데, 이 함수는 수평 조정과 수직 조정을 수락한다. 조정(adjustment)은 개발자가 두 가지 함수 매개변수로 NULL을 전달하면 알아서 생성될 것이다.

GtkLayout에는 네이티브 스크롤링 지원이 있기 때문에 스크롤이 있는 창과 함께 이용 시 GtkDrawingArea를 이용하는 것보다 훨씬 유용할 것이다.

```
GtkWidget* gtk_layout_new (GtkAdjustment *hadjustment,
                           GtkAdjustment *vadjustment);
```

하지만 GtkLayout은 위젯도 포함할 수 있기 때문에 오버헤드가 어느 정도 부가된다. 이 때문에 위젯의 GdkWindow에만 그려야 한다면 GtkDrawingArea가 더 나은 선택이 되겠다.

자식 위젯은 gtk\_layout\_put()을 이용해 GtkLayout 컨테이너로 추가되는데, 이 함수를 호출하면 컨테이너의 상단 좌측 모서리를 기준으로 자식 위젯을 위치시킬 것이다. GtkLayout은 GtkContainer에서 직접 상속되기 때문에 다수의 자식을 지원할 수 있다.

```
void gtk_layout_put (GtkLayout *layout,
                    GtkWidget *child_widget,
                    gint x,
                    gint y);
```

gtk\_layout\_move()의 호출은 후에 자식 위젯을 GtkLayout 컨테이너에서 다른 위치로 이동시킬 때 사용할 수 있다.

■ **Caution** 개발자는 자식 위젯을 구체적인 수평 및 수직 위치에 두기 때문에 GtkLayout은 GtkFixed와 동일한 문제를 제시한다. 레이아웃 위젯을 이용 시에는 이러한 문제를 주의해야 한다! GtkFixed 위젯에서 발생하는 문제에 대해서는 제 3장을 참고한다.

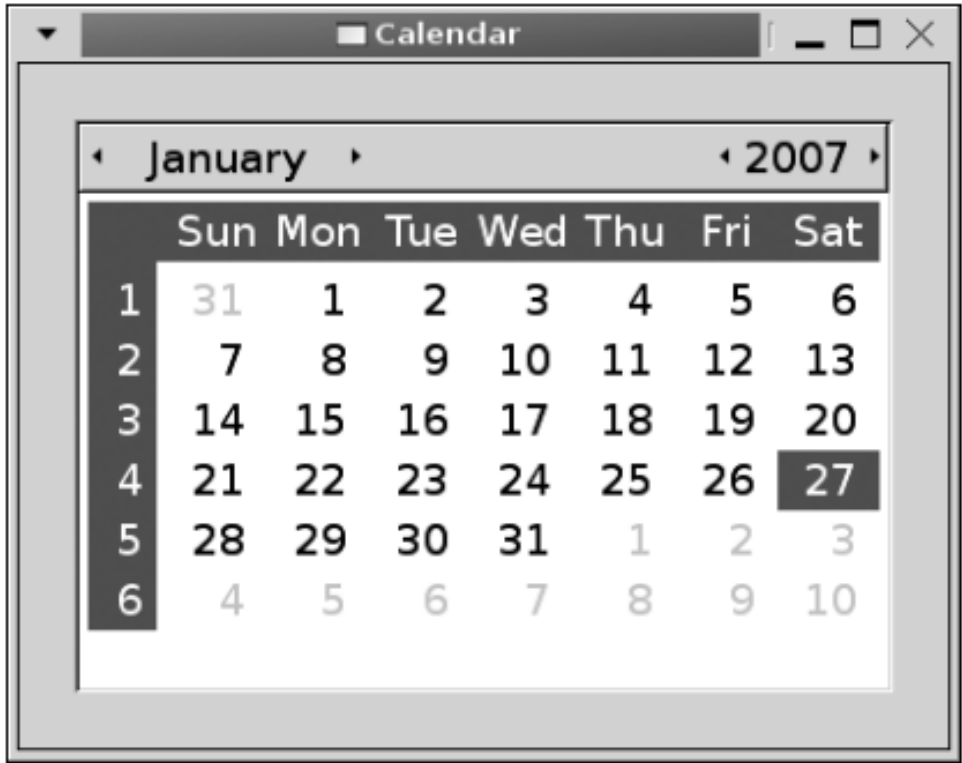
마지막으로 레이아웃의 크기를 강제로 지정하고 싶다면 새로운 너비와 높이 매개변수를 gtk\_layout\_set\_size()로 전송할 수 있다. gtk\_widget\_set\_size\_request() 대신 gtk\_layout\_set\_size()를 이용해야 하는데, 그래야만 조정 매개변수 또한 조정하기 때문이다.

```
void gtk_layout_set_size (GtkLayout *layout,
                          guint width,
                          guint height);
```

뿐만 아니라 크기 요청과 달리 레이아웃 크기조정 함수는 부호가 없는 숫자를 필요로 한다. 즉, 레이아웃 위젯에 대한 절대적인 크기를 명시해야 한다는 의미다. 이 크기는 레이아웃의 총 크기로, 스크롤 영역의 범위를 벗어나거나 화면에 표시되지 않는 위젯의 부분도 포함해야 한다! GtkLayout 위젯의 기본 크기는 100x100 픽셀이다.

### 캘린더

GTK+는 특정 월(month)을 표시하는 GtkCalendar 위젯도 제공한다. 이 위젯은 사용자가 그림 12-2에 표시된 바와 같이 스크롤 화살표를 이용해 월과 연도를 움직이도록 해준다. 선택한 연도에 대한 주(week)의 개수와 요일명을 세 자리 문자로 줄여 표시할 수도 있다.



GtkCalendar 구조체에는 이용 가능한 member의 수가 많지만 모두 읽기만 가능한데, 이러한 객체들을 설명하도록 하겠다. 현재 월과 연도가 프로그램적으로 또는 사용자에게 의해 변경되면 이러한 값들은 모두 리셋될 것 이란 점을 명심해야 한다. 따라서 아래와 같은 변경내용을 처리해야 할 것이다.

- num\_marked\_dates: 당월에 표시된 일수. 이 값은 0과 당월의 총 일자수 사이 값이어야 한다.
- marked\_date: 당월에 표시된 num\_marked\_dates 일수를 포함하는 부호가 없는 정수의 배열.
- month: 사용자가 보고 있는 현재 월. 월의 값은 0부터 11 사이의 값이어야 한다. 월이 변경되면 month-changed 시그널이 발생할 것이다. 캘린더가 다음 월이나 이전 월로 이동하면 next-month 또는 previous-month 시그널이 발생할 것이다.
- year: 표시된 월에 해당하는 현재 연도. 캘린더가 다음 연도 또는 이전 연도로 이동하면 next-year 또는 previous-year 시그널이 발생할 것이다.
- selected\_day: 현재 선택된 일자로, 하나 이상의 일자를 표시할 수는 있지만 항상 하나의 일자에 해당한다. 일자는 1부터 당월에 포함된 일자 수 사이의 값이어야 한다.

새로운 GtkCalendar 위젯은 gtk\_calendar\_new()를 이용해 생성된다. 기본적으로 현재 일자가 선택된다. 따라서 컴퓨터가 저장한 현재 월과 연도도 표시될 것이다. 선택된 일자는 gtk\_calendar\_get\_date()를 이용해 검색하고, gtk\_calendar\_select\_day()를 이용하면 새로운 일자를 선택할 수 있다. 현재 선택한 일자를 선택 해제하려면 gtk\_calendar\_select\_day()를 이용해 일자(date) 값을 0으로 한다.

GtkCalendar 위젯이 어떻게 표시되는지와 어떻게 사용자와 상호작용하는지를 맞춤설정하기 위해서는 gtk\_calendar\_set\_display\_options()를 이용해 GtkCalendarDisplayOptions 값의 bitwise 리스트를 설정해야 한다. 이 열거에서 아직 사용되고 있는 값은 다음과 같다.

- GTK\_CALENDAR\_SHOW\_HEADING: 설정 시 월과 연도의 이름이 표시될 것이다.
- GTK\_CALENDAR\_SHOW\_DAY\_NAMES: 설정 시 각 일자에 대한 3자리 문자로 된 축약어가 해당 일자의 열 위에 표시될 것이다. 주요 캘린더 내용과 헤더 사이에서 렌더링된다.

- `GTK_CALENDAR_NO_MONTH_CHANGE`: 사용자가 캘린더의 현재 월을 변경하지 못하도록 한다. 이 플래그를 설정하지 않으면 다음 또는 이전 월로 이동하도록 화살표가 표시될 것이다. 기본적으로 화살표는 활성화된다.
- `GTK_CALENDAR_SHOW_WEEK_NUMBERS`: 현재 연도에 해당하는 캘린더의 좌측을 따라 주(week) 번호를 표시한다. 기본적으로 주 번호는 숨겨진다.

하나의 일자를 선택하는 기능 외에도 `gtk_calendar_mark_day()`를 이용해 각 월에서 원하는 수만큼 일자를 한번에 하나씩 선택할 수 있다. 일자가 성공적으로 표시되면 `TRUE`를 리턴할 것이다.

```
gboolean gtk_calendar_mark_day (GtkCalendar *calendar,
                               guint day);
```

마크(mark)는 어떤 월에서 연관된 이벤트를 가진 일자를 모두 선택하는 경우 등 여러 용도를 갖고 있다. 일자를 표시하고 나면 일자는 `marked_data` 배열로 추가될 것이다. 일자를 표시하고 나면 `gtk_calendar_unmark_day()`를 이용해 표시를 해제할 수도 있으며, 일자가 성공적으로 표시 해제되면 `TRUE`를 리턴할 것이다. `gtk_calendar_clear_marks()`를 이용하면 모든 일자의 표시를 해제할 수 있다.

```
gboolean gtk_calendar_unmark_day (GtkCalendar *calendar,
                                  guint day);
```

사용자가 일자를 선택할 때를 감지 시 이용할 수 있는 시그널로는 두 가지가 있다. 첫 번째 시그널인 `day-selected`는 사용자가 마우스나 키보드로 새로운 일자를 선택하면 발생할 것이다. `day-selected-double-click` 시그널은 사용자가 더블 클릭을 통해 일자를 선택하면 발생할 것이다. 즉, 대부분의 경우 `GtkCalendar` 위젯과 함께 `button-press-event` 시그널이 필요한 일은 없을 것이란 의미다.

### 상태 아이콘

`GtkStatusIcon` 위젯은 GTK+ 2.10에서 소개되었고, 플랫폼 독립적인 방식으로 시스템 트레이(알림 영역)에 아이콘을 표시하는 데에 사용된다. 시스템 트레이 아이콘은 종종 일부 타입의 이벤트를 사용자에게 비개입적인 방식으로 알려거나 최소화된 애플리케이션으로 쉽게 접근할 수 있도록 제공된다.

시스템 트레이 아이콘의 `GtkStatusIcon` 구현은 툴팁을 추가하고, 아이콘과 상호작용을 위한 팝업 메뉴를 추가하며, 사용자에게 특정한 이벤트 타입을 알리기 위해 아이콘을 깜빡이도록 만드는 기능을 제공한다. 사용자가 아이콘을 클릭하여 아이콘을 활성화시키는 것도 가능하다.

**Note** `GtkStatusIcon`은 `GtkWidget`에서 상속되는 것이 아니라 사실상 `GObject`에 해당한다! Microsoft Windows에서 시스템 트레이 아이콘은 위젯으로 추가될 수 없기 때문에 꼭 필요하다.

새로운 상태 아이콘을 생성하도록 다섯 가지의 함수가 제공된다. 빈 `GtkStatusIcon` 인스턴스는 `gtk_status_icon_new()`를 이용해 생성된다. 해당 초기화 함수를 사용할 경우 객체를 `visible`로 설정하기 전에 시스템 트레이 아이콘에 해당하는 이미지를 명시해야 한다.

```
GtkStatusIcon* gtk_status_icon_new ();
GtkStatusIcon* gtk_status_icon_new_from_pixbuf (GdkPixbuf *pixbuf);
GtkStatusIcon* gtk_status_icon_new_from_file (const gchar *filename);
GtkStatusIcon* gtk_status_icon_new_from_stock (const gchar *stock_id);
GtkStatusIcon* gtk_status_icon_new_from_icon_name (const gchar *icon_name);
```

나머지 네 개의 함수는 각각 `GdkPixbuf` 객체, 시스템 내 파일, 스톡 항목, 혹은 현재 아이콘 테마의 이미지로부터 상태 아이콘을 생성한다. 이러한 함수들은 모두 필요 시 알림 영역에 맞도록 이미지를 스케일링할 것이다.

`gtk_status_icon_new()`를 이용해 상태 아이콘을 초기화했다면 `gtk_status_icon_set_from_pixbuf()`와 `gtk_status_icon_set_from_stock()`를 이용해 이미지를 설정할 수 있다. `GdkPixbuf` 객체, 파일, 스톡 항목, 현재 아이콘 테마의 이미지로부터 이미지를 설정하기 위한 함수들이 제공된다. 이러한 함수들은 후에 애플리케이션의 현재 상태를 반영하도록 이미지

를 변경할 때에도 사용 가능하다. 가령 애플리케이션이 이메일 클라이언트라면 시스템 트레이 아이콘을 애플리케이션의 아이콘에서 봉투모양(envelop)으로 변경하여 새 메시지가 도착했음을 표시하도록 설정할 수 있겠다.

■Tip 기본적으로 상태 아이콘은 visible로 설정된다. gtk\_status\_icon\_set\_visible()을 이용해 뷰에서 아이콘을 숨기거나 visible로 설정 가능하다.

사용자가 시스템 트레이 아이콘 위에서 마우스를 움직이면 추가 정보를 제공하는 툴팁을 표시하고자 할 경우에는 gtk\_status\_icon\_set\_tooltip()을 이용한다. 예를 들어, 다운로드하는 애플리케이션에서 진행 경과의 퍼센트 또는 이메일 클라이언트에서 새로운 메시지의 개수 등의 정보를 제공할 수 있겠다.

```
void gtk_status_icon_set_tooltip (GtkStatusIcon *icon,
    const gchar *tooltip_text);
```

사용자가 알아야 하는 이벤트가 애플리케이션에서 발생하면 gtk\_status\_icon\_set\_blinking()을 이용해 상태 아이콘을 깜빡이도록 만들 수 있다. 이 기능은 사용자의 개인설정에 따라 비활성화할 수도 있다. 이런 경우 함수는 어떤 효과도 보이지 않을 것이다. 이 함수를 이용할 때는 깜빡임 기능을 끌 것을 잊지 말라! 깜빡임 기능이 더 이상 필요하지 않은데도 끄지 않을 시 어떤 사람들에게는 애플리케이션의 사용을 포기할 만큼 큰 골칫거리가 되기도 한다.

```
void gtk_status_icon_set_blinking (GtkStatusIcon *icon,
    gboolean blinking);
```

GtkStatusIcon은 세 개의 시그널을 제공한다. activate 시그널은 사용자가 상태 아이콘을 활성화하면 발생한다. size-changed 시그널은 아이콘에 이용 가능한 크기가 변경되면 발생한다. 이 덕분에 개발자는 아이콘의 크기를 변경하거나 새로운 아이콘이 새로운 크기에 맞도록 로딩할 수 있는데, 이런 경우 TRUE를 리턴해야 한다. FALSE를 리턴하면 GTK+는 현재 아이콘을 새로운 크기에 맞도록 스케일링할 것이다.

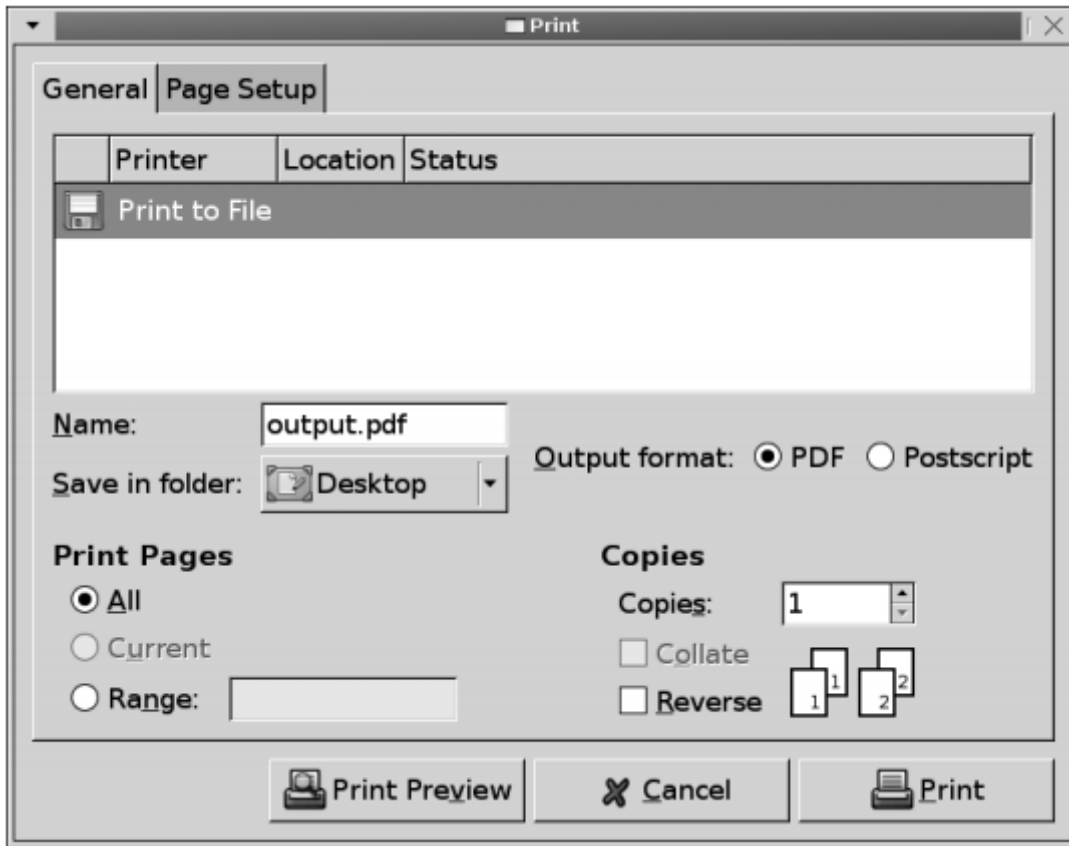
마지막으로 popup-menu 시그널은 사용자가 메뉴를 표시해야 한다고 나타내면 발생한다. 주로 아이콘을 오른쪽 마우스로 클릭하면 되는데, 사용자 플랫폼에 따라 다르다. 이 함수는 각각 어떤 버튼을 눌러야 하는지, 언제 버튼이 활성화되는지를 나타내는 두 개의 부호가 없는 정수를 수락한다. 메뉴를 표시하려면 이 두 개의 값은 gtk\_menu\_popup()으로 전송되어야 한다. gtk\_menu\_popup()의 네 번째 매개변수에 대해서는 gtk\_status\_icon\_position\_menu()를 사용하길 원할 것이다. 이는 메뉴 위치 지정 함수로서 메뉴를 화면 어디에 위치시킬 것인지를 계산할 것이다.

### 인쇄 지원

GTK+ 2.10에는 라이브러리에 인쇄 지원을 추가하기 위해 다수의 새로운 위젯과 객체가 도입되었다. 대부분의 경우에는 해당 API에 많은 객체가 포함되어 있기 때문에 여러 플랫폼에 걸쳐 사용이 가능한 고수준 인쇄 API인 GtkPrintOperation과 직접적으로 상호작용만 하면 된다. 이는 대부분의 인쇄 연산을 처리하는 데에 있어 사용자가 직접 이용하는(front-end) 인터페이스 역할을 한다.

이번 절에서는 사용자가 GtkFileChooserButton 위젯에서 선택하는 텍스트 파일의 내용을 인쇄하는 애플리케이션을 구현할 것이다. Linux 시스템에 표시되는 기본 인쇄 대화상자의 스크린샷을 그림 12-3에 소개하겠다. 사용자는 GtkFileChooserButton 위젯을 이용해 디스크에서 파일을 선택한 후 메인 메뉴의 Print 버튼을 클릭하여 대화상자를 열 수 있다.





리스트 12-2는 애플리케이션에 필요한 데이터 구조체를 정의하고 사용자 인터페이스를 준비시킴으로써 시작한다. 최종 결과물의 렌더링에 도움이 되는 현재 인쇄 작업에 관한 정보를 보유하기 위해 `PrintData` 구조체를 이용할 것이다. `Widgets`는 콜백 함수에서 인쇄 작업의 정보와 여러 개의 위젯으로 접근성을 제공하는 간단한 구조체에 해당한다.

리스트 12-2. GTK+ 인쇄 예제 (`printing.c`)

```
#include <gtk/gtk.h>
#include <math.h>

#define HEADER_HEIGHT 20.0
#define HEADER_GAP 8.5

/* A structure that will hold information about the current print job.
*/
typedef struct
{
    gchar *filename;
    gdouble font_size;
    gint lines_per_page;
    gchar **lines;
    gint total_lines;
    gint total_pages;
} PrintData;

typedef struct
{
```

```
    GtkWidget *window, *chooser;
    PrintData *data;
} Widgets;

GtkPrintSettings *settings;

static void print_file (GtkButton*, Widgets*);
static void begin_print (GtkPrintOperation*, GtkPrintContext*,
Widgets*);
static void draw_page (GtkPrintOperation*, GtkPrintContext*, gint,
Widgets*);
static void end_print (GtkPrintOperation*, GtkPrintContext*, Widgets*);

int main (int argc,
          char *argv[])
{
    GtkWidget *hbox, *print;
    Widgets *w;

    gtk_init (&argc, &argv);

    w = g_slice_new (Widgets);
    w->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (w->window), "Printing");
    gtk_container_set_border_width (GTK_CONTAINER (w->window), 10);

    g_signal_connect (G_OBJECT (w->window), "destroy",
                      G_CALLBACK (gtk_main_quit), NULL);

    w->chooser = gtk_file_chooser_button_new ("Select a File",
      GTK_FILE_CHOOSER_ACTION_OPEN);
    gtk_file_chooser_set_current_folder (GTK_FILE_CHOOSER (w->chooser),
      g_get_home_dir ());

    print = gtk_button_new_from_stock (GTK_STOCK_PRINT);

    g_signal_connect (G_OBJECT (print), "clicked",
                      G_CALLBACK (print_file), (gpointer) w);

    hbox = gtk_hbox_new (FALSE, 5);
    gtk_box_pack_start (GTK_BOX (hbox), w->chooser, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (hbox), print, FALSE, FALSE, 0);

    gtk_container_add (GTK_CONTAINER (w->window), hbox);
    gtk_widget_show_all (w->window);

    gtk_main ();
}
```

```

return 0;
}

```

리스트 12-2의 윗부분에서 `HEADER_HEIGHT`와 `HEADER_GAP`이라는 두 개의 값이 정의된다.

`HEADER-HEIGHT`는 렌더링될 헤더 텍스트에 이용할 수 있는 공간의 양이다. 이것은 파일명이나 페이지 번호와 같은 정보를 표시하는 데에 사용될 것이다. `HEADER_GAP`은 헤더와 실제 페이지 내용 사이에 위치할 패딩이다.

현재 인쇄 작업에 관한 정보를 저장하기 위해서는 `PrintData` 구조체를 이용할 것이다. 이러한 정보로는 디스크 상에서 파일의 위치, 글꼴 크기, 한 페이지에 렌더링 가능한 행의 수, 파일의 내용, 총 행의 수, 총 페이지 수가 포함된다.

#### 인쇄 연산

다음 단계는 `GTK_STOCK_PRINT` 버튼을 클릭하면 실행될 콜백 함수를 구현하는 일이다. 이 함수는 리스트 12-3에서 구현하였다. 이는 `PrintData` 객체를 생성하고, 필요한 시그널을 모두 연결하며, 인쇄 연산을 생성하는 일을 처리할 것이다.

#### 리스트 12-3. 인쇄와 인쇄 미리보기

```

/* Print the selected file with a font of "Monospace 10". */
static void
print_file (GtkButton *button,
            Widgets *w)
{
    GtkPrintOperation *operation;
    GtkWidget *dialog;
    GError *error = NULL;
    gchar *filename;
    gint res;

    /* Return if a file has not been selected because there is nothing
to print. */
    filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER
(w->chooser));
    if (filename == NULL)
        return;

    /* Create a new print operation, applying saved print settings if
they exist. */H
    operation = gtk_print_operation_new ();
    if (settings != NULL)
        gtk_print_operation_set_print_settings (operation, settings);

    w->data = g_slice_new (PrintData);
    w->data->filename = g_strdup (filename);
    w->data->font_size = 10.0;

    g_signal_connect (G_OBJECT (operation), "begin_print",
                      G_CALLBACK (begin_print), (gpointer) w);
    g_signal_connect

```

```

(G_OBJECT (operation), "draw_page",
    G_CALLBACK (draw_page), (gpointer) w);
g_signal_connect (G_OBJECT (operation), "end_print",
    G_CALLBACK (end_print), (gpointer) w);

/* Run the default print operation that will print the selected
file. */
res = gtk_print_operation_run (operation,
GTK_PRINT_OPERATION_ACTION_PRINT_DIALOG,
    GTK_WINDOW (w->window), &error);

/* If the print operation was accepted, save the new print
settings. */
if (res == GTK_PRINT_OPERATION_RESULT_APPLY)
{
    if (settings != NULL)
        g_object_unref (settings);
    settings = g_object_ref (gtk_print_operation_get_print_settings
(operation));
}
/* Otherwise, report that the print operation has failed. */
else if (error)
{
    dialog = gtk_message_dialog_new (GTK_WINDOW (w->window),
GTK_DIALOG_DESTROY_WITH_PARENT,
    GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE,
    error->message);

    g_error_free (error);
    gtk_dialog_run (GTK_DIALOG (dialog));
    gtk_widget_destroy (dialog);
}

g_object_unref (operation);
g_free (filename);
}

```

인쇄의 첫 번째 단계는 새로운 인쇄 연산을 생성하는 일로, `gtk_print_operation_new()`를 이용하면 가능하다. `GtkPrintOperation`이 독특한 이유는 플랫폼의 네이티브 인쇄 대화상자가 있을 경우 그것을 사용하기 때문이다. 그러한 대화상자를 제공하지 않는 UNIX와 같은 플랫폼에서는 `GtkPrintUnixDialog`가 사용될 것이다.

■ **Note** 대부분의 애플리케이션에서는 인쇄(print) 객체와 직접 상호작용하는 대신 가능하면 `GtkPrintOperation` API를 사용해야 한다. `GtkPrintOperation`은 플랫폼 독립적인 인쇄 해답으로 생성되어 많은 양의 코드가 없이는 쉽게 구현할 수 없다.

다음 단계는 연산에 인쇄 설정을 적용하도록 `gtk_print_operation_print_settings()`를 호출해야 한다. 이 애플리케이션에서 `GtkPrintSettings` 객체는 `settings`라고 불리는 전역 변수로 저장된다. 인쇄 연산이 성공적으로 실행 되면 개발자는 현재 인쇄 설정을 저장해야만 추후 인쇄 작업에 동일한 설정을 적용할 수 있다.

다음으로 `g_slice_new()`를 이용하면 새로운 객체를 할당하여 `PrintData` 구조체를 준비할 수 있다. 파일명은 이미 존재가 확인된 `GtkFileChooserButton`에서 현재 선택된 파일로 설정된다. 인쇄 글꼴 크기 또한 10.0 포인트로 설정된다. 텍스트 편집 애플리케이션에서 이러한 글꼴은 현재 `GtkTextView`의 글꼴에서 검색하는 것이 보통이다. 좀 더 복잡한 인쇄 애플리케이션에서는 글꼴 크기가 문서마다 다양하지만 이번 예제는 시작 단계의 개발자들을 대상으로 하기 때문에 간단하게 구현하였다.

이제 세 개의 `GtkPrintOperation` 시그널을 연결해야 하는데, 이에 관해서는 이번 절의 뒷부분에서 상세히 논하겠다. 간략하게 말하자면, `begin-print`는 페이지가 렌더링되기 전에 호출되며, 페이지 수를 설정하고 필요한 준비작업을 실행하는 데에 사용할 수 있다. `draw-page` 시그널은 페이지의 렌더링을 가능하게 해주기 때문에 인쇄 작업에 있는 모든 페이지마다 호출된다. 마지막으로 `end-print` 시그널은 인쇄 연산의 성공이나 실패 여부와 상관없이 인쇄 연산이 완료되고 나면 호출된다. 이러한 콜백 함수는 인쇄 작업이 끝난 후 정리하는 데에 사용된다. 인쇄 연산에 걸쳐 사용 가능한 그 외 시그널도 많이 존재하는데, 전체 리스트는 부록 B에서 찾을 수 있다.

인쇄 연산이 준비되고 나면 `gtk_print_operation_run()`을 호출하여 인쇄를 시작해야 한다. 이 함수에서는 인쇄 연산이 어떠한 작업을 실행할 것인지 정의한다.

```
GtkPrintOperationResult gtk_print_operation_run (GtkPrintOperation
*operation,
        GtkPrintOperationAction action,
        GtkWindow *parent,
        GError **error);
```

아래 표시된 `GtkPrintOperationAction` 열거는 인쇄 연산이 실행하는 인쇄 작업을 정의한다. 문서를 인쇄하기 위해서는 `GTK_PRINT_OPERATION_ACTION_PRINT_DIALOG`를 이용해야 한다.

- `GTK_PRINT_OPERATION_ACTION_PRINT_DIALOG`: 플랫폼의 기본 인쇄 대화상자를 표시하고, 기본 대화상자가 없다면 `GtkPrintUnixDialog`를 사용한다. 대부분의 인쇄 연산에서 일반적인 액션에 해당한다.
- `GTK_PRINT_OPERATION_ACTION_PRINT`: 인쇄 대화상자를 표시하지 않고 현재 인쇄 설정을 이용해 인쇄를 시작한다. 사용자가 이 액션을 승인할 것이라고 100% 확실할 때만 실행해야 한다. 사용자에게 이미 확인 대화상자를 표시한 경우를 예로 들 수 있겠다.
- `GTK_PRINT_OPERATION_ACTION_PREVIEW`: 현재 설정으로 실행될 인쇄 작업을 미리보기한다. 인쇄 연산과 동일한 렌더링용 콜백을 사용하므로 준비 및 실행에 약간의 작업이 필요하다.
- `GTK_PRINT_OPERATION_ACTION_EXPORT`: 인쇄 작업을 파일로 내보내기(`export`)한다. 이 설정을 이용하기 위해서는 연산을 실행하기 전에 `export-filename` 프로퍼티를 설정해야 한다.

`gtk_print_operation_run()`의 마지막 두 매개변수는 인쇄 대화상자에 사용할 부모 창 또는 `GError` 구조체를 정의하도록 해주기도 하고, `NULL`을 이용해 매개변수를 무시할 수도 있다. 이 함수는 모든 페이지가 렌더링되고 프린터로 전송되고 나서야 리턴할 것이다.

함수가 제어를 다시 돌려주면 `GtkPrintOperationResult` 열거 값을 리턴할 것이다. 이 값은 개발자가 그 다음으로 어떤 작업을 실행해야 하는지, 인쇄 연산이 성공했는지 실패했는지에 대한 설명을 제공한다. 네 가지 열거 값을 아래 소개하겠다.

- `GTK_PRINT_OPERATION_RESULT_ERROR`: 인쇄 연산에서 특정 타입의 오류가 발생하였다. 상세한 정보는 `GError` 객체를 이용해야 한다.
- `GTK_PRINT_OPERATION_RESULT_APPLY`: 인쇄 설정이 변경되었다. 따라서 변경 내용을 손실하지 않기 위해서는 즉시 저장해야 한다.
- `GTK_PRINT_OPERATION_RESULT_CANCEL`: 사용자가 인쇄 연산을 취소했으며, 개발자는 변경 내용을 인쇄 설정으로 저장해선 안 된다.
- `GTK_PRINT_OPERATION_RESULT_IN_PROGRESS`: 인쇄 연산이 아직 완료되지 않았다. 작업을 비동기 식으로 실행 중일 때만 이 값을 얻을 것이다.

인쇄 연산을 비동기식으로 실행하는 것도 가능한데, 이는 페이지의 렌더링이 완료되기 전에 `gtk_print_operation_run()`이 리턴할 수도 있다는 의미다. 이러한 실행은 `gtk_print_operation_set_allow_async()`를 통해 설정된다. 모든 플랫폼이 이 연산을 허용하는 것은 아니므로 혹시 작동하지 않을 수도 있으니 마음의 준비를 하도록 한다!

인쇄 연산을 비동기식으로 실행할 경우 `done` 시그널을 이용해 인쇄가 완료되면 알림을 복구시킬 수 있다. 이 시점에서 개발자에게 인쇄 연산 결과가 제공되는데, 결과는 알림에 따라 처리해야 할 것이다.

인쇄 연산의 결과를 처리하고 난 후에 결과적인 오류가 설정되어 존재한다면 오류 또한 처리해야 한다. `GtkPrintError` 도메인에서 발생 가능한 오류의 리스트는 부록 B에서 찾아볼 수 있다. 인쇄 연산에서 가장 최근에 발생한 `GError`를 검색하기 위해서는 `gtk_print_operation_get_error()`를 이용할 수도 있다. 이는 `GTK_PRINT_OPERATION_RESULT_ERROR`를 리턴한 인쇄 작업에 관한 추가 정보를 검색하기 위해 비동기식으로 인쇄 연산을 실행할 때 사용 가능하다.

`GtkPrintOperation`이 제공하는 한 가지 독특한 기능으로, 인쇄 연산이 실행되는 동안 진행 대화상자를 표시하는 기능을 들 수 있다. 기본적으로는 꺼져 있지만 `gtk_print_operation_set_show_progress()`를 이용해 켤 수 있다. 사용자로 하여금 여러 개의 인쇄 연산을 동시에 실행하도록 허용할 때 특히 유용한 기능이다.

```
void gtk_print_operation_set_show_progress (GtkPrintOperation
*operation,
      gboolean show_progress);
```

때로는 현재 인쇄 작업을 취소해야 하는 경우가 있는데, 이럴 때는 `gtk_print_operation_cancel()`을 호출하면 된다. 이 함수는 주로 `begin-print`, `paginate`, 또는 `draw-page` 콜백 함수에서 사용된다. 이 함수는 사용자가 활성화된 인쇄 연산 도중에 중단할 수 있도록 `Cancel` 버튼을 제공하도록 해주기도 한다.

```
void gtk_print_operation_cancel (GtkPrintOperation *operation);
```

인쇄 작업에 유일한 이름을 제공할 수도 있는데, 그 이름은 외부 인쇄 모니터링 애플리케이션에서 작업을 식별하는 데에 사용될 것이다. 인쇄 작업의 이름은 `gtk_print_operation_set_job_name()`을 이용해 주어진다. 이 값이 설정되지 않으면 `GTK+`는 자동으로 인쇄 작업에 이름을, 연속된 인쇄 작업에는 숫자를 지정한다.

인쇄 작업을 비동기식으로 실행 중이라면 인쇄 작업의 현재 상태를 검색하길 원할 것이다.

`gtk_print_operation_get_status()`를 호출하면 인쇄 작업의 상태에 관한 추가 정보를 제공하는 `GtkPrintStatus` 열거 값이 리턴될 것이다. 인쇄 작업 상태에 가능한 값의 리스트는 다음과 같다.

- `GTK_PRINT_STATUS_INITIAL`: 인쇄 연산이 아직 시작되지 않았다. 이것은 기본 초기값이므로 인쇄 대화상자가 아직 표시되어 있을 때 이 상태가 리턴될 것이다.
- `GTK_PRINT_STATUS_PREPARING`: 인쇄 연산이 두 페이지로 나뉘고, `begin-print` 시그널이 발생하였다.
- `GTK_PRINT_STATUS_GENERATING_DATA`: 페이지가 렌더링 중이다. 이 값은 `draw-page` 시그널이 발생하는 동안 설정될 것이다. 이 때는 어떤 데이터도 프린터로 전송되지 않을 것이다.
- `GTK_PRINT_STATUS_SENDING_DATA`: 인쇄 작업에 관한 데이터가 프린터로 전송 중이다.
- `GTK_PRINT_STATUS_PENDING`: 모든 데이터가 프린터로 전송되었지만 작업은 처리되지 않았다. 프린터를 중단시키는 것이 가능하다.
- `GTK_PRINT_STATUS_PENDING_ISSUE`: 인쇄 도중에 문제가 발생하였다. 프린터에 용지가 부족한 경우 혹은 용지가 걸리는 경우를 예로 들 수 있다.
- `GTK_PRINT_STATUS_PRINTING`: 프린터가 현재 인쇄 작업을 처리 중이다.
- `GTK_PRINT_STATUS_FINISHED`: 인쇄 작업이 성공적으로 완료되었다.
- `GTK_PRINT_STATUS_FINISHED_ABORTED`: 인쇄 작업이 취소되었다. 작업을 다시 실행하지 않는 한 어떤 액션도 취하지 않을 것이다.

`gtk_print_operation_get_status()`가 리턴한 값은 수치값이기 때문에 애플리케이션 내에서 사용 가능하다. 하지만 `GTK+`는 `gtk_print_operation_get_status_string()`을 이용해 문자열을 검색하는 기능도 제공하는데, 이 문자열은 인쇄 작업 상태를 설명하는 사람이 읽을 수 있는 문자열에 해당한다. 이 문자열은 출력의 디버깅 또는 사용

자에게 인쇄 작업에 관한 추가 정보를 표시하는 데에 사용할 수 있다. 가령 상태 바 위에 혹은 메시지 대화상자 안에 문자열을 표시할 수 있겠다.

#### 인쇄 연산 시작하기

인쇄 연산이 모두 준비되었으니 필요한 시그널 콜백 함수를 구현할 차례다. `begin-print` 시그널은 사용자가 인쇄를 시작 시, 즉 사용자의 관점에서 모든 설정이 완료되면 발생한다.

리스트링 12-4에서는 `begin_print()` 콜백 함수를 이용해 먼저 파일의 내용을 검색하여 여러 개의 행으로 나누었다. 그리고 총 행의 수가 계산되는데, 이는 페이지 수를 검색하는 데에 사용할 수 있다.

#### 리스트링 12-4. `begin-print` 시그널에 대한 콜백 함수

```

/* Begin the printing by retrieving the contents of the selected files
and
* splitting it into single lines of text. */
static void
begin_print (GtkPrintOperation *operation,
             GtkPrintContext *context,
             Widgets *w)
{
    gchar *contents;
    gdouble height;
    gsize length;

    /* Retrieve the file contents and split it into lines of text. */
    g_file_get_contents (w->data->filename, &contents, &length, NULL);
    w->data->lines = g_strsplit (contents, "\n", 0);

    /* Count the total number of lines in the file. */
    w->data->total_lines = 0;
    while (w->data->lines[w->data->total_lines] != NULL)
        w->data->total_lines++;

    /* Based on the height of the page and font size, calculate how
many lines can be
* rendered on a single page. A padding of 3 is placed between lines
as well. */
    height = gtk_print_context_get_height (context) - HEADER_HEIGHT -
HEADER_GAP;
    w->data->lines_per_page = floor (height / (w->data->font_size + 3));
    w->data->total_pages = (w->data->total_lines - 1) /
w->data->lines_per_page + 1;
    gtk_print_operation_set_n_pages (operation, w->data->total_pages);
    g_free (contents);
}

```

인쇄 연산에서 필요로 하는 페이지 수를 계산하기 위해서는 모든 페이지에 얼마나 많은 행이 렌더링될 수 있는지 알아내야 한다. 모든 페이지의 총 높이는 `gtk_print_context_get_height()`를 이용해 검색할 수 있는데, 이는 `GtkPrintContext` 객체에 저장된다. `GtkPrintContext`는 페이지를 어떻게 그리는지에 관한 정보를 저장하는 데에 사용된다. 예를 들어 이 객체는 페이지 셋업, 너비와 높이 치수, 양방향으로 인치당 도트 수를 저장한다.

draw-page 콜백 함수는 이번 장의 뒷부분에서 상세히 살펴보겠다.

텍스트의 렌더링에 사용될 페이지의 총 높이를 얻었다면 텍스트 글꼴 크기에서 각 행 사이에 3 픽셀의 공간을 더한 값을 높이에서 나누어야 한다. 페이지별 행의 수를 내림(round down)하도록 floor() 함수를 이용하여 전면(full page)의 하단에 클리핑(clipping)이 발생하지 않도록 하였다.

페이지별 행의 수를 얻었다면 이제 총 페이지 수를 계산할 수 있다. 개발자는 이 값을 gtk\_print\_operation\_set\_n\_pages() 콜백 함수가 끝날 무렵 이 함수로 전송해야 한다. GTK+가 draw-page 콜백 함수를 얼마나 많이 호출해야 하는지 알도록 페이지 수가 사용될 것이다. 페이지 수는 양수 값이어야 하는데, 그 래야만 기본값 -1에서 변경되기 전에는 렌더링이 시작되지 않도록 확보할 수 있기 때문이다.

페이지 렌더링하기

그 다음은 렌더링되어야 하는 페이지마다 한 번씩 호출될 draw-page 콜백 함수를 구현할 차례다. 이 콜백 함수를 이용하려면 Cairo라고 불리는 또 다른 라이브러리를 소개할 필요가 있다. Cairo는 벡터 그래픽 라이브러리로서 무엇보다 인쇄 연산을 렌더링하는 데에 사용된다.

리스트 12-5에서는 가장 먼저 gtk\_print\_context\_get\_cairo\_context()를 이용해 현재 GtkPrintContext에 대한 Cairo 드로잉 컨텍스트를 검색하였다. cairo\_t 객체를 이용해 인쇄 내용을 렌더링한 후 PangoLayout으로 적용하였다.

이 콜백 함수가 시작되면 GtkPrintContext로부터 두 개의 다른 값을 검색해야 한다. 첫 번째는 gtk\_print\_context\_get\_widget()로, 문서의 너비를 리턴하는 함수다. 각 페이지에 일치하는 행의 수는 이미 계산했으므로 페이지 높이를 검색할 필요는 없음을 기억한다. 텍스트가 페이지보다 넓으면 클리핑될 것이다. 문서의 클리핑을 피하려면 이 예제를 수정해야 할 것이다.

**Caution** GtkPrintContext가 리턴한 너비는 픽셀로 되어 있다. 다른 함수들은 Pango 단위(unit)나 포인트(point)와 같은 다른 대안적 스케일을 사용할 수 있으므로 주의해야 한다.

다음 단계에서는 인쇄 컨텍스트에 사용 가능한 gtk\_print\_context\_create\_layout()을 이용해 PangoLayout을 생성해야 한다. 인쇄 연산에도 이런 방식으로 Pango 레이아웃을 생성해야 하는데, 그 이유는 인쇄 연산에 이미 올바른 글꼴 메트릭스(font metrics)가 적용되어 있기 때문이다.

리스트 12-5. draw-page 시그널에 대한 콜백 함수

```

/* Draw the page, which includes a header with the file name and page
number along
 * with one page of text with a font of "Monospace 10". */
static void
draw_page (GtkPrintOperation *operation,
           GtkPrintContext *context,
           gint page_nr,
           Widgets *w)
{
    cairo_t *cr;
    PangoLayout *layout;
    gdouble width, text_height;
    gint line, i, text_width, layout_height;
    PangoFontDescription *desc;
    gchar *page_str;

    cr = gtk_print_context_get_cairo_context (context);
    width = gtk_print_context_get_width (context);
    layout = gtk_print_context_create_pango_layout (context);

```



```

desc = pango_font_description_from_string ("Monospace");
pango_font_description_set_size (desc, w->data->font_size *
PANGO_SCALE);

/* Render the page header with the filename and page number. */
pango_layout_set_font_description (layout, desc);
pango_layout_set_text (layout, w->data->filename, -1);
pango_layout_set_width (layout, -1);
pango_layout_set_alignment (layout, PANGO_ALIGN_LEFT);
pango_layout_get_size (layout, NULL, &layout_height);
text_height = (gdouble) layout_height / PANGO_SCALE;

cairo_move_to (cr, 0, (HEADER_HEIGHT - text_height) / 2);
pango_cairo_show_layout (cr, layout);

page_str = g_strdup_printf ("%d of %d", page_nr + 1,
w->data->total_pages);
pango_layout_set_text (layout, page_str, -1);
pango_layout_get_size (layout, &text_width, NULL);
pango_layout_set_alignment (layout, PANGO_ALIGN_RIGHT);

cairo_move_to (cr, width - (text_width / PANGO_SCALE),
              (HEADER_HEIGHT - text_height) / 2);
pango_cairo_show_layout (cr, layout);

/* Render the page text with the specified font and size. */
cairo_move_to (cr, 0, HEADER_HEIGHT + HEADER_GAP);
line = page_nr * w->data->lines_per_page;
for (i = 0; i < w->data->lines_per_page && line < w->data->total_lines; i++)
{
    pango_layout_set_text (layout, w->data->lines[line], -1);
    pango_cairo_show_layout (cr, layout);
    cairo_rel_move_to (cr, 0, w->data->font_size + 3);
    line++;
}

g_free (page_str);
g_object_unref (layout);
pango_font_description_free (desc);
}

```

그 다음으로 이 함수는 파일명을 페이지의 상단 좌측 모서리에 추가하는 연산을 실행하였다. 먼저 `pango_layout_set_text()`는 레이아웃이 저장한 현재 텍스트를 파일명으로 설정한다. 레이아웃의 너비는 -1로 설정되어야만 파일명이 사선(/) 문자에서 래핑(wrap)하지 않는다. 텍스트는 `pango_layout_set_alignment()`를 이용해 레이아웃의 좌측으로 정렬된다.

텍스트가 레이아웃으로 추가되었으니 `cairo_move_to()`를 이용해 Cairo에서 현재 포인트를 페이지의 좌측과 헤더의 중앙으로 이동한다. PangoLayout의 높이는 먼저 PANGO\_SCALE의 인수(factor)만큼 감소해야 한다!

```
void cairo_move_to (cairo_t *cairo_context,
                  double x,
                  double y);
```

다음으로 Cairo 컨텍스트에 PangoLayout을 그리기 위해 pango\_cairo\_show\_layout()을 호출한다. 레이아웃의 상단 좌측 모서리는 Cairo 컨텍스트의 현재 포인트에서 렌더링된다. 이 때문에 cairo\_move\_to()를 이용해 바람직한 위치로 먼저 이동해야 했다.

```
void pango_cairo_show_layout (cairo_t *cairo_context,
                             PangoLayout *layout);
```

파일명을 렌더링한 후에는 동일한 방법을 이용해 페이지 계수를 각 페이지의 상단 우측 모서리로 추가한다. PangoLayout이 리턴한 너비는 PANGO\_SCALE로 축소되어야만 다른 Cairo 값들과 동일한 단위가 될 것이라는 사실을 명심한다.

다음으로는 현재 페이지에 대한 모든 행을 렌더링해야 한다. 먼저 페이지 좌측으로 헤더 아래의 HEADER\_GAP 단위를 이동시켜야 한다. 그러면 각 행은 pango\_cairo\_show\_layout()을 이용해 Cairo 컨텍스트로 점차적으로(incrementally) 렌더링된다. 한 가지 흥미로운 점은 루프에서 커서의 위치는 cairo\_rel\_move\_to()를 이용해 이동된다는 것이다.

```
void cairo_rel_move_to (cairo_t *cairo_context),
                    double dx,
                    double dy);
```

이 함수는 이전 위치를 기준으로 현재 위치를 이동하는 데에 사용된다. 따라서 한 행이 렌더링되고 나면 현재 위치가 한 줄 밑으로 이동하는데, 글꼴은 Monospace이기 때문에 텍스트의 글꼴 크기와 동일하다.

**Tip** 이전 위치를 기준으로 커서를 이동시키면 텍스트의 각 행과 주위의 행 사이에 임의의 공간을 추가하기가 쉬운데, 다만 begin-print 콜백 함수에서 페이지 수를 계산할 때 이 추가높이를 계산에 넣은 경우에 한해서만 해당된다.

GTK+를 이용해 개발할 때는 전체 Cairo 라이브러리를 이용할 수 있다. 이번 장에서 Cairo를 다룬 절에서 몇 가지 기본적인 내용을 소개하겠다. 하지만 자신만의 애플리케이션에서 인쇄를 구현 중이라면 Cairo API 문서에서 라이브러리를 좀 더 학습하도록 한다.

### 인쇄 연산 최종화하기

모든 페이지를 렌더링했다면 end-print 시그널이 발생할 것이다. 리스팅 12-6은 이 시그널에 사용될 콜백 함수를 보여준다. 이는 PrintData 객체에서 동적으로 할당된 모든 메모리를 해제하고 나서 객체 자체도 해제한다.

리스팅 12-6. end-print 시그널에 대한 콜백 함수

```
/* Clean up after the printing operation since it is done. */
static void
end_print (GtkPrintOperation *operation,
          GtkPrintContext *context,
          Widgets *w)
{
    g_strfreev (w->data->lines);
    g_slice_free1 (sizeof (PrintData), w->data);
    w->data = NULL;
}
```

GTK+가 제공하는 인쇄 API는 어차피 크기가 방대하기 때문에 PangoLayout 과 Cairo에 대해 규모가 큰 API를 고려하지 않아도 된다. 따라서 이번 예제는 시작 단계에 불과하며 API가 제시하는 학습 곡선을 수월하게 완

료하도록 도와주는 간단한 예제에 불과하다. 자신만의 애플리케이션에 인쇄를 구현할 때 이 예제를 이용해 시작해도 좋지만 대부분의 경우 이 주제를 좀 더 깊게 연구해야 할 것이다.

### Cairo 드로잉 컨텍스트

Cairo는 GTK+ 라이브러리에 걸쳐 사용된 그래픽 렌더링 라이브러리다. 본 서적에서는 인쇄 연산 중에 페이지를 렌더링하도록 Cairo를 이용하였다. 이번 절에서는 cairo\_t 객체를 비롯해 이와 연관된 몇 가지 그리기 함수를 소개하겠다.

GTK+에서 인쇄 연산의 페이지는 cairo\_t 객체로 렌더링된다. 이 객체는 텍스트를 렌더링하고, 다양한 도형과 선(line)을 그리도록 해주며, 클리핑된 영역을 색상으로 채우도록 해준다. Cairo 드로잉 컨텍스트를 조작할 수 있도록 Cairo가 제공하는 함수를 몇 가지 살펴보겠다.

#### 패스(path) 그리기

Cairo 컨텍스트에서 도형은 패스를 이용해 렌더링된다. 새로운 패스는 cairo\_new\_path()를 이용해 생성된다. 이후 cairo\_copy\_path()를 통해 새로운 패스의 복사본을 검색하고 패스에 새로운 선과 도형을 추가할 수 있다.

```
cairo_path_t* cairo_copy_path (cairo_t *cairo_context);
```

패스를 그리는 데에 제공되는 함수에는 다수가 있는데, 표 12-1에 열거하겠다. 이 함수에 관한 추가 정보는 Cairo API 문서에서 찾을 수 있다.

함수	설명
cairo_arc()	현재 패스에 호(arc)를 그린다. 호의 반경, 중심(center)의 가로와 세로 위치, 곡선의 시작 각도와 끝 각도(라디안)를 제공한다.
cairo_curve_to()	현재 패스에 Bezier 곡선을 생성한다. 개발자는 곡선을 계산하는 데에 사용될 두 개의 제어점과 곡선의 끝 위치를 제공해야 한다.
cairo_line_to()	현재 위치에서 명시된 점까지 직선을 그린다. 첫 지점이 존재하지 않으면 현재 위치는 단순히 이동할 것이다.
cairo_move_to()	컨텍스트 내 새로운 위치로 이동하면 새로운 하위패스가 생성될 것이다.
cairo_rectangle()	현재 패스 내에 직사각형을 그린다. 개발자는 직사각형의 상단 좌측 모서리의 좌표와 직사각형의 너비 및 높이를 제공해야 한다.
cairo_rel_curve_to()	이 함수는 현재 위치를 기준으로 그려진다는 점을 제외하면 cairo_curve_to()와 동일하다.
cairo_rel_line_to()	이 함수는 현재 위치를 기준으로 그려진다는 점을 제외하면 cairo_line_to()와 동일하다.
cairo_rel_move_to()	이 함수는 현재 위치를 기준으로 그려진다는 점을 제외하면 cairo_move_to()와 동일하다.

표 12-1. Cairo 패스 그리기 함수

하위패스의 사용을 완료하면 cairo\_path\_close()를 이용해 닫아야 한다. 이 함수를 이용하면 필요 시 현재 패스를 색상으로 채울 수 있도록 패스를 에워쌀(enclose) 것이다.

### 렌더링 옵션

소스에서 그리기 옵션에 사용된 현재 색상은 `cairo_set_source_rgb()`에 해당한다. 색상은 새로운 색상이 설정될 때까지 사용될 것이다. 색상의 선택 외에도 `cairo_set_source_rgba()`를 이용하면 다섯 번째 알파 매개변수를 수락한다. 각 색상 매개변수는 0.0과 1.0 사이의 부동 소수점수다.

```
void cairo_set_source_rgb (cairo_t *cairo_context,
                          double red,
                          double green,
                          double blue);
```

현재 패스를 구체적인 지점으로 옮겨 소스의 색상을 선택하고 나면 컨텍스트만 수락하는 `cairo_fill()`을 이용해 패스를 채울 수 있다. 아니면 `cairo_fill_extents()`를 이용해 직사각형 영역을 채우는 수도 있다. 이 함수는 (x1,y1)과 (x2,y2)의 모서리로 된 영역을 계산하여 두 점 사이의 영역이면서 현재 패스에 의해 포함된 영역을 모두 채운다.

```
void cairo_fill_extents (cairo_t *cairo_context,
                        double *x1,
                        double *y1,
                        double *x2,
                        double *y2);
```

곡선과 같은 그리기 연산은 도형의 변을 들쭉날쭉하게 만드는 경우가 있다. 이를 수정하기 위해 Cairo는 `cairo_set_antialias()`를 통해 그리기에 안티 앨리어징(antialiasing)을 제공한다.

```
void cairo_set_antialias (cairo_t *cairo_context,
                         cairo_antialias_t antialias);
```

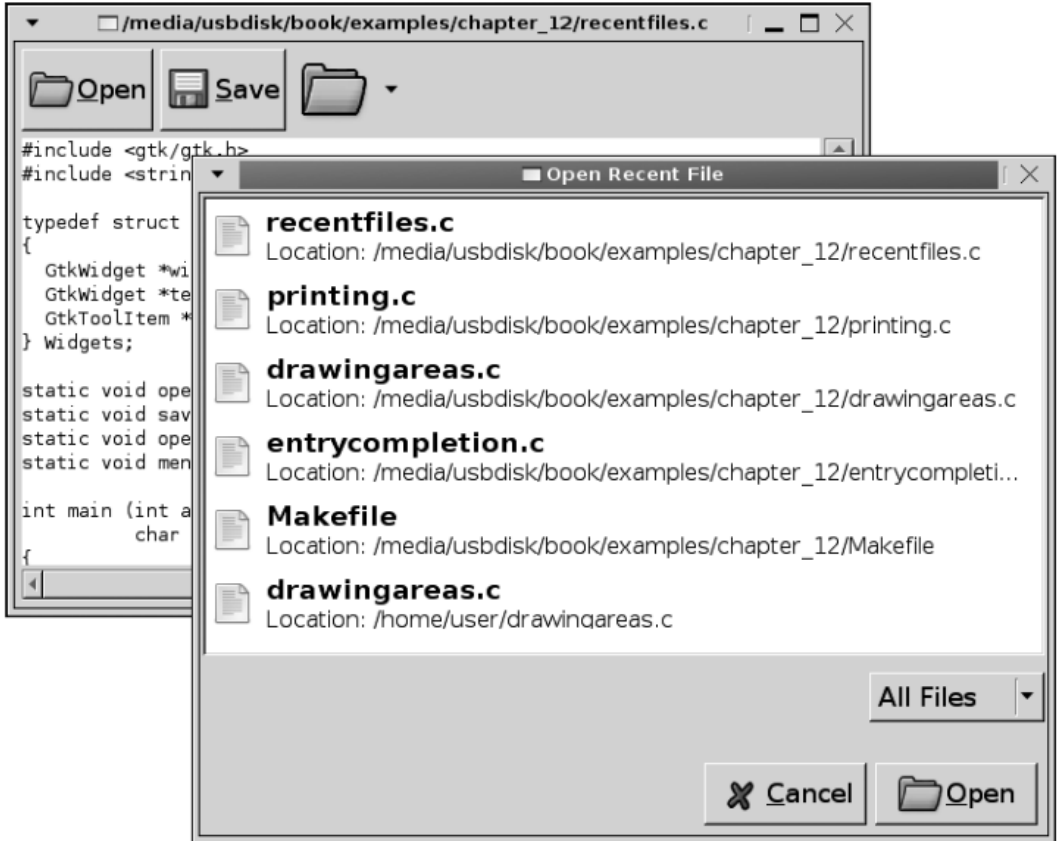
안티 앨리어징은 `cairo_antialias_t` 열거에서 제공된다. 이 열거에서 정의하는 값은 다음과 같다.

- `CAIRO_ANTIALIAS_DEFAULT`: 기본 안티 앨리어징 알고리즘이 사용될 것이다.
- `CAIRO_ANTIALIAS_NONE`: 안티 앨리어징이 발생하지 않고, 대신 알파 마스크가 사용될 것이다.
- `CAIRO_ANTIALIAS_GRAY`: 안티 앨리어징에 단일 색만 사용하라. 꼭 회색일 필요는 없지만 전경색과 배경색을 기준으로 선택된다.
- `CAIRO_ANTIALIAS_SUBPIXEL`: LCD 화면에서 제공되는 하위 픽셀 셰딩(subpixel shading)을 사용하라.

Cairo에 대해 약간의 맛만 보도록 Cairo의 드로잉 컨텍스트를 간략하게 소개해보았다. Cairo에 관한 세부적인 정보는 [www.cairographics.org](http://www.cairographics.org)에 실린 API 문서를 참조하도록 한다.

### 최근 파일

GTK+ 2.10에서는 새로운 API가 소개되어 여러 애플리케이션에 걸쳐서 최근에 연 파일을 추적하도록 해준다. 이번 절에서는 간단한 텍스트 편집 애플리케이션에서 이러한 기능을 구현해볼 것이다. 최근 파일 선택자가 있는 애플리케이션의 모습은 그림 12-4에서 확인할 수 있다. 뒷부분에 실린 연습문제를 통해 개발자의 텍스트 에디터에 최근 파일 지원을 추가해볼 것이다.



리스트 12-7에

실린 코드는 텍스트 편집 애플리케이션을 준비한다. 두 개의 버튼은 GtkFileChooserDialog를 이용해 이미 존재하는 파일을 열고 변경내용을 저장하도록 해준다. 그 다음으로 GtkMenuToolButton이 있는데, 이는 두 개의 함수를 제공한다. 버튼을 클릭하면 GtkRecentChooserDialog가 표시되면서 리스트에서 최근 파일을 선택하도록 해준다. GtkMenuToolButton 위젯 내 메뉴는 가장 최근에 사용한 10개의 파일을 표시하는

GtkRecentChooserMenu 타입이다.  
리스트 12-7. 최근에 연 파일 기억하기 (recentfiles.c)

```
#include <gtk/gtk.h>

typedef struct
{
    GtkWidget *window;
    GtkWidget *textview;
} Widgets;

static void open_file (GtkButton*, Widgets*);
static void save_file (GtkButton*, Widgets*);
static void open_recent_file (GtkButton*, Widgets*);
static void menu_activated (GtkMenuShell*, Widgets*);

int main (int argc,
```

```

    char *argv[])
{
    GtkWidget *vbox, *hbox, *open, *save, *swin, *icon, *menu;
    PangoFontDescription *fd;
    GtkRecentManager *manager;
    Widgets *w;

    gtk_init (&argc, &argv);

    w = g_slice_new (Widgets);
    w->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (w->window), "Recent Files");
    gtk_container_set_border_width (GTK_CONTAINER (w->window), 5);
    gtk_widget_set_size_request (w->window, 600, 400);

    g_signal_connect (G_OBJECT (w->window), "destroy",
        G_CALLBACK (gtk_main_quit), NULL);

    w->textview = gtk_text_view_new ();
    fd = pango_font_description_from_string ("Monospace 10");
    gtk_widget_modify_font (w->textview, fd);
    pango_font_description_free (fd);

    swin = gtk_scrolled_window_new (NULL, NULL);
    open = gtk_button_new_from_stock (GTK_STOCK_OPEN);
    save = gtk_button_new_from_stock (GTK_STOCK_SAVE);
    icon = gtk_image_new_from_stock (GTK_STOCK_OPEN,
GTK_ICON_SIZE_BUTTON);
    w->recent = gtk_menu_tool_button_new (icon, "Recent Files");

    /* Load the default recent chooser menu and create a menu from it.
    */
    manager = gtk_recent_manager_get_default ();
    menu = gtk_recent_chooser_menu_new_for_manager (manager);
    gtk_menu_tool_button_set_menu (GTK_MENU_TOOL_BUTTON (w->recent),
menu);

    gtk_recent_chooser_set_show_not_found (GTK_RECENT_CHOOSER (menu),
FALSE);
    gtk_recent_chooser_set_local_only (GTK_RECENT_CHOOSER (menu),
TRUE);
    gtk_recent_chooser_set_limit (GTK_RECENT_CHOOSER (menu), 10);
    gtk_recent_chooser_set_sort_type (GTK_RECENT_CHOOSER (menu),
GTK_RECENT_SORT_MRU);

    g_signal_connect (G_OBJECT (menu), "selection-done",
        G_CALLBACK (menu_activated), (gpointer) w);

```

```

    /* ... Connect other signals and populate the window ... */

    gtk_container_add (GTK_CONTAINER (w->window), vbox);
    gtk_widget_show_all (w->window);

    gtk_main ();
    return 0;
}

/* Save the changes that the user made to the file to disk. */
static void
save_file (GtkButton *save,
           Widgets *w)
{
    const gchar *filename;
    gchar *content;
    GtkTextBuffer *buffer;
    GtkTextIter start, end;

    filename = gtk_window_get_title (GTK_WINDOW
(w->window));
    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (w->textview));
    gtk_text_buffer_get_bounds (buffer, &start, &end);
    content = gtk_text_buffer_get_text (buffer, &start, &end, FALSE);

    if (!g_file_set_contents (filename, content, -1, NULL))
        g_warning ("The file '%s' could not be written!", filename);
    g_free (content);
}

```

GtkRecentManager라고 불리는 주요(central) 클래스는 최근 파일 정보를 처리한다. 처음부터 자신만의 파일을 생성할 수도 있지만 최근 파일을 여러 애플리케이션에 걸쳐 공유하고 싶다면 `gtk_recent_manager_get_default()`를 이용해 기본값을 검색할 수 있다. 이는 GEdit, GNOME의 최근 문서 메뉴, GtkRecentManager API를 활용하는 다른 애플리케이션들과 최근 파일을 공유하도록 해준다.

다음으로 기본 GtkRecentManager로부터 새로운 GtkRecentChooserMenu 위젯을 생성한다. 이 메뉴는 최근 파일을 표시하고, `gtk_recent_chooser_menu_new_for_manager()`를 이용해 생성된 메뉴 항목에 선택적으로 번호를 매길 것이다. 파일에 기본적으로 번호가 매겨지지 않지만 이 프로퍼티는 `show-numbers`를 TRUE로 설정하거나 `gtk_recent_chooser_menu_set_show_numbers()`를 호출하여 변경이 가능하다.

GtkRecentChooserMenu는 위젯과 상호작용에 필요하게 될 기능을 제공하는 GtkRecentChooser 인터페이스를 구현한다. 리스팅 12-7에서는 메뉴를 맞춤설정하기 위해 다수의 GtkRecentChooser 프로퍼티가 사용되었다. 이들은 GtkRecentChooser 인터페이스를 구현하는 또 다른 위젯 두 개에도 적용되는데, 바로 GtkRecentChooserDialog와 GtkRecentChooserWidget이다.

리스트에 최근 파일이 추가되면 리스트에서 제거하는 것도 가능하다. 이런 경우 리스트에 파일이 표시되는 것은 원치 않을 것이다. 더 이상 존재하지 않는 최근 파일은 `gtk_recent_chooser_set_show_not_found()`를 이용해 숨길 수 있다. 이 프로퍼티는 로컬 머신에 위치한 파일에만 작용할 것이다.

■Tip 찾을 수 없는 파일을 사용자에게 표시하길 원할 수도 있다. 사용자가 존재하지 않는 파일을 선택하면 개발자는 사용자에게 문제를 알린 후 리스트에서 해당 파일을 쉽게 제거할 수 있다.

기본적으로는 로컬 파일만 사용자에게 표시되는데, 즉 file:// 가 앞에 붙은 인터넷 식별자(URI)를 갖게 될 것 이란 뜻이다. URI는 접두사를 기반으로 파일 위치나 인터넷 주소 등을 참조하는 데에 사용된다. file:// 접두사 만 사용할 경우 로컬 머신에 위치함을 보장할 것이다. 이 프로퍼티를 FALSE로 설정하면 원격 위치에 존재하 는 최근 파일을 표시할 수 있다. 원격 파일은 더 이상 존재하지 않을 경우 필터링되지 않음을 명심한다!

리스트에 최근 파일의 수가 많은 경우 메뉴에 모두 열거하는 일은 원치 않을 것이다. 메뉴에 수백 개의 항목이 포함되어 있다면 얼마나 커질지 생각해보라! 따라서 gtk\_recent\_chooser\_set\_limit()를 이용하면 메뉴에 표시 될 최근 항목의 최대 수를 설정할 수 있다.

```
void gtk_recent_chooser_set_limit (GtkRecentChooser *chooser,
    gint limit);
```

요소의 개수를 제한할 때 어떤 파일을 표시할 것인지는 gtk\_recent\_chooser\_set\_sort\_type()을 이용해 정의한 정 렬 타입에 따라 좌우된다. 기본적으로는 GTK\_RECENT\_SORT\_NONE으로 설정된다. GtkRecentSortType 열 거에서 이용 가능한 값은 다음과 같다.

- GTK\_RECENT\_SORT\_NONE: 최근 파일의 리스트가 전혀 정렬되지 않고, 요소가 표시되는 순서대로 리턴 될 것이다. 어떤 파일이 표시될 것인지 예측할 수 없으므로 표시할 요소의 개수를 제한한다면 이 값을 사용 해선 안 된다.
- GTK\_RECENT\_SORT\_MRU: 리스트에 가장 최근에 추가한 파일을 먼저 정렬한다. 가장 최근의 파일을 리 스트 처음에 위치시키므로 대부분 이 값을 사용할 것이다.
- GTK\_RECENT\_SORT\_LRU: 리스트에 가장 늦게 추가한 파일을 먼저 정렬한다.
- GTK\_RECENT\_SORT\_CUSTOM: 최근 파일의 정렬에 커스텀 정렬 함수를 이용한다. 이 값을 이용하려면 사용할 정렬 함수를 정의하도록 gtk\_recent\_manager\_set\_sort\_func()를 이용해야 한다.

이 예제에서 마지막 부분은 명시된 이름으로 파일을 저장하는 것이다. 텍스트 에디터에서 파일이 열리면 창 제목이 파일명으로 설정된다. 이 파일명은 파일을 저장 시에 사용된다. 이에 따라 이렇게 간단한 텍스트 에디 터는 새로운 파일을 생성할 때에는 사용할 수 없으므로 주의하길 바란다!

최근 선택자 메뉴

지금까지 GtkRecentChooserMenu 위젯에 대해 학습하였다. 리스팅 12-8은 리스팅 12-7에서 연결했던 selection-done 콜백 함수를 구현한다. 이 함수는 선택된 URI을 검색하고, 파일이 존재할 경우 파일을 연다.

리스팅 12-8. GtkRecentChooserMenu 이용하기

```
/* A menu item was activated. So, retrieve the file URI and open it. */
static void
menu_activated (GtkMenuShell *menu,
    Widgets *w)
{
    GtkTextBuffer *buffer;
    gchar *filename, *content, *fn;
    gsize length;

    filename = gtk_recent_chooser_get_current_uri (GTK_RECENT_CHOOSER
(menu));

    if (filename != NULL)
    {
        /* Remove the "file://" prefix from the beginning of the URI if
```



```

it exists. */
    fn = g_filename_from_uri (filename, NULL, NULL);

    if (g_file_get_contents (fn, &content, &length, NULL))
    {
        gtk_window_set_title (GTK_WINDOW (w->window), fn);
        buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW
(w->textview));
        gtk_text_buffer_set_text (buffer, content, -1);
        g_free (content);
    }
    else
        g_warning ("The file '%s' could not be read!", filename);

    g_free (filename);
    g_free (fn);
}
}

```

하나의 항목만 선택 가능하므로 현재 선택된 최근 파일을 검색할 때는 `gtk_recent_chooser_get_current_uri()`를 이용할 수 있다. 메뉴는 로컬 파일만 표시하도록 제한하였기 때문에 URI에서 `file://` 접두사를 제거해야 한다. 원격 파일의 표시도 허용한다면 URI에서 다른 접두사, 즉 `http://` 같은 접두사는 제거해야 할 것이다. URI 접두사를 제거하려면 `g_filename_from_uri()`를 이용하면 된다.

```

gchar* g_filename_from_uri (const gchar *uri,
    gchar **hostname,
    GError **error);

```

접두사가 제거되고 나면 GLib는 파일 열기를 시도한다. 파일이 성공적으로 열리면 창 제목이 파일명으로 설정되고 파일이 열린다. 파일 열기가 실패하면 파일을 열 수 없다는 경고가 사용자에게 표시된다.

### 최근 파일 추가하기

Open 버튼을 누르면 사용자가 `GtkFileChooserDialog`로부터 열 파일을 선택할 수 있기를 바랄 것이다. 파일이 열리면 리스팅 12-9에서 볼 수 있듯이 기본 `GtkRecentManager`로 추가될 것이다.

리스팅 12-9. 파일을 열고 최근 파일 리스트로 추가하기

```

/* Open a file selected by the user and add it as a new recent file. */
static void
open_file (GtkButton *open,
    Widgets *w)
{
    GtkWidget *dialog;
    GtkRecentManager *manager;
    GtkRecentData *data;
    GtkTextBuffer *buffer;
    gchar *filename, *content, *uri;
    gsize length;

    static gchar *groups[2] = {

```

```

        "testapp",
        NULL
    };

    dialog = gtk_file_chooser_dialog_new ("Open File", GTK_WINDOW
(w->>window),
        GTK_FILE_CHOOSER_ACTION_OPEN,
        GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
        GTK_STOCK_OPEN, GTK_RESPONSE_OK,
        NULL);

    if (gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_OK)
    {
        filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER
(dialog));

        if (g_file_get_contents (filename, &content, &length, NULL))
        {
            /* Create a new recently used resource. */
            data = g_slice_new (GtkRecentData);
            data->display_name = NULL;
            data->description = NULL;
            data->mime_type = "text/plain";
            data->app_name = (gchar*) g_get_application_name ();
            data->app_exec = g_strjoin (" ", g_get_prpname (), "%u",
NULL);

            data->groups = groups;
            data->is_private = FALSE;
            uri = g_filename_to_uri (filename, NULL, NULL);

            /* Add the recently used resource to the default recent
manager. */
            manager = gtk_recent_manager_get_default ();
            gtk_recent_manager_add_full (manager, uri, data);

            /* Load the file and set the filename as the title of the
window. */
            gtk_window_set_title (GTK_WINDOW (w->>window), filename);
            buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW
(w->textview));
            gtk_text_buffer_set_text (buffer, content, -1);

            g_free (content);
            g_free (uri);
            g_free (data->app_exec);
            g_slice_free (GtkRecentData, data);
        }
    }

```

```

else
    g_warning ("The file '%s' could not be read!", filename);

    g_free (filename);
}

gtk_widget_destroy (dialog);
}

```

파일이 성공적으로 열리면 `gtk_recent_manager_add_full()`을 이용해 파일을 기본 `GtkRecentManager`에 새로운 최근 항목으로 추가한다. 이 함수를 이용하려면 두 가지 항목이 필요하다. 첫째는 URI가 필요한데, 이는 파일이 로컬 파일임을 나타내도록 `file://` 뒤에 파일명을 붙여서 생성된다. 파일명은 `g_filename_to_uri()`를 이용해 빌드된다.

```

gchar* g_filename_to_uri (const gchar *filename,
    const gchar *hostname,
    GError **error);

```

두 번째로 `GtkRecentData` 구조체의 인스턴스가 필요하다. 이 구조체의 내용은 아래 코드 조각에 표시하였는데, 7개의 매개변수가 포함되어 있다. `display_name`은 파일명 대신 표시해야 할 짧은 이름에 해당하며, `description`은 파일의 간략한 설명에 해당한다. 두 값 모두 안전하게 NULL로 설정 가능하다.

```

typedef struct
{
    gchar *display_name;
    gchar *description;
    gchar *mime_type;
    gchar *app_name;
    gchar *app_exec;
    gchar **groups;
    gboolean is_private;
} GtkRecentData;

```

다음은, 파일에 대한 MIME 타입, 애플리케이션명, 파일을 여는 데 사용한 명령행을 명시해야 한다. 애플리케이션의 이름은 `g_get_application_name()`을 호출하여 검색 가능하다. 이후 `g_get_prgname()`을 이용해 프로그램명을 얻을 수 있다. `%f`와 `%u` 문자는 각각 리소스에 대한 파일 경로와 URI를 얻는 데에 사용된다.

다음으로 `groups`는 리소스가 속한 그룹을 지정하는 문자열 리스트다. 개발자는 이를 이용해 특정 그룹에 속하지 않은 파일을 필터링할 수 있다. 가령 `testapp` 그룹에 대한 필터가 `GtkRecentChooser`로 추가되면 이 애플리케이션에 의해 추가된 최근 파일만 표시될 것이다.

마지막 member인 `is_private`은 리소스를 등록하지 않은 애플리케이션에서도 리소스를 이용할 수 있는지 여부를 명시한다. 이를 TRUE로 설정하면 `GtkRecentManager`를 이용하는 다른 애플리케이션들은 해당하는 최근 파일을 표시하지 못하도록 방지할 수 있다.

`GtkRecentData` 인스턴스를 생성했다면 최근 파일 URI와 함께 새로운 리소스로서 추가할 수 있는데, 이는 `gtk_recent_manager_add_item()`을 통해 이루어진다. `gtk_recent_manager_add_item()`을 이용해 새로운 최근 항목을 추가할 수도 있는데, 이를 호출하면 알아서 `GtkRecentData` 객체가 생성될 것이다.

최근 항목을 제거하려면 `gtk_recent_manager_remove_item()`을 호출하라. 명시된 URI로 된 파일이 성공적으로 제거되면 이 함수는 TRUE를 리턴할 것이다. 만일 파일 제거가 실패하면 `GtkRecentManagerError` 도메인에서 오류가 설정될 것이다. `gtk_recent_manager_purge_items()`를 이용해 리스트로부터 최근 파일을 모두 제거할 수도 있다.

```
gboolean gtk_recent_manager_remove_item (GtkRecentManager *manager,
    const gchar *uri,
    GError **error);
```

**Caution** 기본 `GtkRecentManager`에서 모든 항목을 제거(purging)하는 것은 피하도록 한다! 이는 모든 애플리케이션에서 등록된 최근 항목을 제거하는데, 자신의 애플리케이션에서는 보통 다른 애플리케이션의 최근 리소스를 변경해서는 안 되므로 사용자가 이를 필요로 하는 경우는 드물기 때문이다.

최근 선택자 대화상자

GTK+는 최근 파일을 편의(convenient) 대화상자에 표시하는 `GtkRecentChooserDialog`라는 위젯도 제공한다. 이 위젯은 `GtkRecentChooser` 인터페이스를 구현하기 때문에 `GtkRecentChooserMenu`와 그 기능이 매우 비슷하다. 리스팅 12-10은 사용자가 열어야 할 최근 파일을 선택하기 위해 이 위젯을 이용하는 방법을 보여준다.

리스팅 12-10. `GtkRecentChooserDialog` 이용하기

```
/* Allow the user to choose a recent file from the list in the dialog.
 */
static void
open_recent_file (GtkButton *recent,
    Widgets *w)
{
    GtkWidget *dialog;
    GtkRecentManager *manager;

    GtkTextBuffer *buffer;
    GtkRecentFilter *filter;
    gchar *filename, *content, *fn;
    gsize length;

    manager = gtk_recent_manager_get_default ();
    dialog = gtk_recent_chooser_dialog_new_for_manager ("Open Recent
File",
        GTK_WINDOW (w->window), manager,
        GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
        GTK_STOCK_OPEN, GTK_RESPONSE_OK, NULL);

    /* Add a filter that will display all of the files in the dialog.
 */
    filter = gtk_recent_filter_new ();
    gtk_recent_filter_set_name (filter, "All Files");
    gtk_recent_filter_add_pattern (filter, "*");
    gtk_recent_chooser_add_filter (GTK_RECENT_CHOOSER (dialog),
filter);

    /* Add another filter that will only display plain text files. */
    filter = gtk_recent_filter_new ();
    gtk_recent_filter_set_name (filter, "Plain Text");
    gtk_recent_filter_add_mime_type (filter, "text/plain");
    gtk_recent_chooser_add_filter (GTK_RECENT_CHOOSER (dialog),
```

```

filter);

    gtk_recent_chooser_set_show_not_found (GTK_RECENT_CHOOSER (dialog),
FALSE);
    gtk_recent_chooser_set_local_only (GTK_RECENT_CHOOSER (dialog),
TRUE);
    gtk_recent_chooser_set_limit (GTK_RECENT_CHOOSER (dialog), 10);
    gtk_recent_chooser_set_sort_type (GTK_RECENT_CHOOSER (dialog),
GTK_RECENT_SORT_MRU);

    if (gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_OK)
    {
        filename = gtk_recent_chooser_get_current_uri
(GTK_RECENT_CHOOSER (dialog));

        if (filename != NULL)
        {
            /* Remove the "file://" prefix from the beginning of the
URI if it exists. */
            fn = g_filename_from_uri (filename, NULL, NULL);

            if (g_file_get_contents (fn, &content, &length, NULL))
            {
                gtk_window_set_title (GTK_WINDOW (w->window), fn);
                buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW
(w->textview));
                gtk_text_buffer_set_text (buffer, content, -1);
                g_free (content);
            }
            else
                g_warning ("The file '%s' could not be read!",
filename);

            g_free (filename);
            g_free (fn);
        }
    }

    gtk_widget_destroy (dialog);
}

```

새로운 `GtkRecentChooserDialog` 위젯은 `gtk_recent_chooser_dialog_new_for_manager()`를 이용해 대화상자와 비슷한 방식으로 생성된다. 이 함수는 대화상자 제목, 부모 창, 표시할 `GtkRecentManager` 위젯, 버튼과 응답 식별자 쌍을 수락한다.

리스트링 12-10에서는 최근 파일 필터를 소개한다. 새로운 `GtkRecentFilter` 객체는 `gtk_recent_chooser_new()`를 이용해 생성된다. 설치된 패턴을 따르는 최근 파일만 표시하도록 필터를 이용하였다.

```
gtk_recent_filter_set_name (filter, "All Files");
gtk_recent_filter_add_pattern (filter, "*");
gtk_recent_chooser_add_filter (GTK_RECENT_CHOOSER (dialog), filter);
```

다음 단계는 필터의 이름을 설정하는 일이다. 이 이름은 사용자가 사용할 필터를 선택하는 콤보 박스에 표시 될 것이다. 필터를 생성하는 방법에는 여러 가지가 있는데, 일치하는 패턴으로 필터를 찾는 `gtk_recent_filter_add_pattern()`도 그 중 하나다. 별표 문자를 와일드카드로 이용할 수도 있다. MIME 타입, 이미지 타입, 애플리케이션명, 그룹명, 일자 수(ages in days)를 매칭하기 위한 함수들도 존재한다. 다음으로 `gtk_recent_chooser_add_filter()`를 이용해 `GtkRecentFilter`를 최근 선택자로 추가한다.

`GtkRecentChooserDialog` 위젯의 경우 `gtk_recent_chooser_set_select_multiple()`을 이용해 다중 파일을 선택하는 것이 가능하다. 사용자가 다중 파일을 선택할 경우, 선택된 파일을 모두 검색하기 위해서는 `gtk_recent_chooser_get_uris()`를 이용해야 할 것이다.

```
gchar** gtk_recent_chooser_get_uris (GtkRecentChooser *chooser,
                                     gsize *length);
```

이 함수는 NULL로 끝나는 문자열 리스트에 요소의 개수를 리턴하기도 한다. 리스트 사용이 끝나면 `g_strfreev()`를 이용해 해제해야 한다.

### 자동 완성

제 4장에서 `GtkEntry` 위젯에 대해 학습한 바 있지만 GTK+는 `GtkEntryCompletion` 객체를 제공하기도 한다. `GtkEntryCompletion`은 `GObject`에서 파생되며, `GtkEntry` 위젯에서 사용자에게 자동 완성 기능을 제공할 때 사용 가능하다. 그림 12-5는 사용자에게 다중 선택 기능을 제공하는 `GtkEntry`의 예를 보여준다. 사용자는 선택을 무시하고 임의의 문자열을 입력할 수도 있음을 주목한다.



리스트 12-11은 GTK+ 위젯의 이름을 입력하도록 요청하는 `GtkEntry` 위젯을 구현한다. 입력된 텍스트와 동일한 접두사를 가진 `GtkEntryCompletion` 위젯 내 문자열은 모두 선택(choice)으로 표시된다. 이 예제는 자동 완성을 준비시키고 실행하는 것이 얼마나 쉬운지를 보여준다.

리스트 12-11. 자동 완성 (entrycompletion.c)

```
#include <gtk/gtk.h>

#define NUM_ELEMENTS 4

static gchar *widgets[] = { "GtkDialog", "GtkWindow", "GtkContainer",
                             "GtkWidget" };
```

```
int main (int argc,
          char *argv[])

{
    GtkWidget *window, *vbox, *label, *entry;
    GtkEntryCompletion *completion;
    GtkListStore *store;
    GtkTreeIter iter;
    unsigned int i;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Automatic Completion");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    label = gtk_label_new ("Enter a widget in the following
GtkEntry:");
    entry = gtk_entry_new ();

    /* Create a GtkListStore that will hold autocompletion
possibilities. */
    store = gtk_list_store_new (1, G_TYPE_STRING);
    for (i = 0; i < NUM_ELEMENTS; i++)
    {
        gtk_list_store_append (store, &iter);
        gtk_list_store_set (store, &iter, 0, widgets[i], -1);
    }

    completion = gtk_entry_completion_new ();
    gtk_entry_set_completion (GTK_ENTRY (entry), completion);
    gtk_entry_completion_set_model (completion, GTK_TREE_MODEL
(store));
    gtk_entry_completion_set_text_column (completion, 0);

    vbox = gtk_vbox_new (FALSE, 5);
    gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), entry, FALSE, FALSE, 0);

    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show_all (window);

    g_object_unref (completion);
    g_object_unref (store);

    gtk_main ();
}
```

```

return 0;
}
    
```

GtkEntryCompletion을 구현하기 위해서는 먼저 선택을 표시하는 새로운 GtkListStore를 생성해야 한다. 이 예제에 실린 모델은 하나의 텍스트 열만 포함하지만 그 하나의 열이 G\_TYPE\_STRING에 해당하는 한 줄 더 복잡한 GtkListStore를 제공하는 것도 허용된다.

새로운 GtkEntryCompletion 객체는 gtk\_entry\_completion\_new()를 이용해 생성된다. 이렇게 생긴 객체는 후에 gtk\_entry\_set\_completion()을 이용해 기존 GtkEntry 위젯으로 적용할 수도 있다. GTK+는 매치된 내용을 표시하고 선택을 적용하는 일을 기본적으로 알아서 처리할 것이다.

다음은 gtk\_entry\_completion\_set\_model()을 이용해 트리 모델을 GtkEntryCompletion 객체로 적용한다. 객체로 적용된 모델이 이미 존재한다면 대체될 것이다. gtk\_entry\_completion\_set\_text\_column()을 이용하면 어떤 열이 문자열을 포함하는지 지정할 수 있는데, 모델에는 꼭 하나의 열만 있으란 법은 없기 때문이다. 텍스트 열을 설정하지 않으면 텍스트 열이 기본적으로 -1로 설정되어 자동 완성이 작동하지 않을 것이다.

gtk\_entry\_completion\_set\_inline\_completion()을 이용하면 모든 매치에 공통된 접두사를 원하는 만큼 표시할 수가 있다. 인라인 완성(inline completion)은 대·소문자를 구분하지만 자동 완성에서는 구분되지 않음을 명심해야 한다! 인라인 완성을 이용 시 하나의 매치만 있을 때에는 팝업 메뉴가 표시되지 않도록 하려면 gtk\_entry\_completion\_set\_popup\_single\_match()를 설정해야 할 것이다.

gtk\_entry\_completion\_set\_popup\_set\_width()를 이용하면 강제로 팝업 메뉴의 너비를 GtkEntry 위젯과 동일한 너비로 만든다. 이는 GtkEntryCompletion의 popup-set-width 프로퍼티와 동일하다.

매치가 너무 많다면 gtk\_entry\_completion\_set\_minimum\_key\_length()를 이용해 최소 매치 길이를 설정하길 원할 것이다. 이 함수는 리스트에 요소의 수가 너무 많아서 화면에 렌더링하는 데에 오랜 시간이 소요되는 경우에 유용하겠다.

### 자신의 이해도 시험하기

이번 장에 실린 연습문제에서는 앞의 여러 장에 실린 연습문제에서 집중적으로 살펴본 텍스트 편집 애플리케이션을 완성할 것이다. 그러기 위해서는 자동 완성, 인쇄, 최근 파일 기능을 애플리케이션으로 통합해야 할 것이다.

#### 연습문제 12-1. 전체 텍스트 에디터 생성하기

이번 연습문제에서는 앞의 몇 장에 걸쳐 생성한 텍스트 에디터를 완성할 것이다. 먼저 애플리케이션에 세 가지 새로운 기능을 추가할 것이다.

첫 번째로 검색 톨바에 지난 검색을 기억하기 위해 구현되어야 하는 자동 완성 기능을 추가하라. 애플리케이션은 애플리케이션 런타임의 현재 인스턴스에 대한 과거의 검색만 기억해야 한다.

다음으로 인쇄 지원을 추가해야 하는데, 인쇄 지원에는 인쇄하기와 인쇄 미리보기 기능이 포함된다. 인쇄 지원은 고수준의 GtkPrintOperation API로 쉽게 구현이 가능하다. 마지막으로 GtkRecentManager API를 이용해 텍스트 에디터로 하여금 마지막으로 로딩된 5개의 파일을 기억하도록 지시하라.

앞에서 살펴본 애플리케이션의 측면은 재작성할 필요가 없으므로 제 10장의 연습문제 해답을 이용하거나 본 서적의 웹 사이트에서 해답을 다운로드하도록 한다.





GTK 3.0 100% clear\_flag, sqrt\_clicked() GtkEntry GtkEntry (negate) (power) 3-3 GtkDrawingArea 13-3. GtkDrawingArea expose-event (clear) expose-event gdk\_draw\_lines() gdk\_draw\_polygon() (scaffolding) TRUE PangoLayout GtkDrawingArea gdk\_draw\_layout() GdkWindow gdk\_draw\_line() gdk\_draw\_polygon() 5 GIOChannel 13-4 ping) 13-4. g\_spawn\_async\_with\_pipes() g\_shell\_parse\_argv() Ping GIOChannel GtkTreeView Stop kill() kill (pid, SIGINT); watch GIOChannel (shut down) XML XML 13-1 test.cal <calendar>

```
<event> <name>Release of the Book</name> <location>Everywhere</location> <day>16</day>
<month>3</month> <year>2007</year> <start>All Day</start> <end></end> </event> </calendar>
```

New Save Open GLib XML XML 13-1 XML passthrough() passthrough\_text() void (\*passthrough) (GMarkupParseContext \*context, const gchar \*passthrough\_text, gsize text\_len, gpointer data, GError \*\*error); error() GMarkupParser void (\*error) (GMarkupParseContext \*context, GError \*error, gpointer data); XML GMarkupParseContext g\_markup\_parse\_context\_new() GMarkupParser GMarkupParseFlags G\_MARKUP\_TREAT\_CDATA\_AS\_TEXT. g\_markup\_parse\_context\_new() GMarkupParser XML g\_market\_parse\_context\_parse() TRUE gboolean g\_markup\_parse\_context\_parse (GMarkupParseContext \*context, const gchar \*text, gsize

```
<event> <name>Release of the Book</name> <location>Everywhere</location> <day>16</day>
<month>3</month> <year>2007</year> <start>All Day</start> <end></end> </event> </calendar>
```

New Save Open GLib XML XML 13-1 XML passthrough() passthrough\_text() void (\*passthrough) (GMarkupParseContext \*context, const gchar \*passthrough\_text, gsize text\_len, gpointer data, GError \*\*error); error() GMarkupParser void (\*error) (GMarkupParseContext \*context, GError \*error, gpointer data); XML GMarkupParseContext g\_markup\_parse\_context\_new() GMarkupParser GMarkupParseFlags G\_MARKUP\_TREAT\_CDATA\_AS\_TEXT. g\_markup\_parse\_context\_new() GMarkupParser XML g\_market\_parse\_context\_parse() TRUE gboolean g\_markup\_parse\_context\_parse (GMarkupParseContext \*context, const gchar \*text, gsize

text\_len, GError \*\*error); 이 코드는 g\_markup\_parse\_context\_free()를 호출하여, GMarkupParse 객체를 해제하고, GTK+ 라이브러리를 초기화합니다. 이 코드는 GTK+ 라이브러리를 초기화하고, http://www.gtkbook.com에서 GTK+에 대한 정보를 얻을 수 있습니다. GTK+에 대한 정보는 http://www.gtk.org에서 얻을 수 있습니다. GTK+에 대한 정보는 http://developer.gnome.org에서 얻을 수 있습니다. GTK+에 대한 정보는 http://developer.gnome.org에서 얻을 수 있습니다. Libgnomeui는 GTK+에 대한 정보를 얻을 수 있습니다. Libgnomeprint는 GTK+에 대한 정보를 얻을 수 있습니다. GConf: GNOME에 대한 정보를 얻을 수 있습니다. GConf에 대한 정보는 http://developer.gnome.org에서 얻을 수 있습니다. Libgnomeui는 GTK+에 대한 정보를 얻을 수 있습니다. Libgnomeprint는 GTK+에 대한 정보를 얻을 수 있습니다. GConf: GNOME에 대한 정보를 얻을 수 있습니다. GConf에 대한 정보는 http://developer.gnome.org에서 얻을 수 있습니다. GnomeVFS: GNOME에 대한 정보를 얻을 수 있습니다. MIME: MIME에 대한 정보를 얻을 수 있습니다. ORBit: CORBA에 대한 정보를 얻을 수 있습니다. Libart: GnomeCanvas에 대한 정보를 얻을 수 있습니다. GNOME: GNOME에 대한 정보를 얻을 수 있습니다. Bonobo: GNOME에 대한 정보를 얻을 수 있습니다. CROBRA: GNOME에 대한 정보를 얻을 수 있습니다. VTE: GNOME에 대한 정보를 얻을 수 있습니다. Terminal: GNOME에 대한 정보를 얻을 수 있습니다. GtkWidget: GNOME에 대한 정보를 얻을 수 있습니다. 13은 GTK+에 대한 정보를 얻을 수 있습니다. 6은 GTK+에 대한 정보를 얻을 수 있습니다. API에 대한 정보는 http://www.gtk.org에서 얻을 수 있습니다. Notes

# Foundations of GTK Development: Appendix A

## 부록 A GTK+ 프로퍼티

### GTK+ 프로퍼티

GObject는 프로퍼티 시스템을 제공하여 위젯이 사용자와 어떻게 상호작용하고 화면에 어떻게 그려지는지 개발자가 맞춤설정할 수 있도록 해준다. 지금부터는 GTK+ 2.10에서 이용 가능한 위젯과 자식 프로퍼티에 대해 완전한 참고자료를 제공할 것이다.

### GTK+ 프로퍼티

GObject에서 파생된 모든 클래스는 원하는 수만큼의 프로퍼티를 가질 수 있다. GTK+에서 이러한 프로퍼티는 위젯의 현재 상태에 관한 정보를 저장한다. 가령 GtkButton은 relief라는 프로퍼티를 갖고 있는데, 이 프로퍼티는 정상 상태에서 버튼이 사용하는 양감 테두리(relief border)의 타입을 정의한다.

아래 코드에서는 g\_object\_get()을 이용해 버튼의 relief 프로퍼티가 저장한 현재 값을 검색하였다. 이 함수는 리턴된 값을 저장하기 위해 NULL로 끝나는 프로퍼티와 변수의 리스트를 수락한다. g\_object\_set()을 이용해 각 객체(object) 프로퍼티도 설정이 가능하다.

```
g_object_get (button, "relief", &value, NULL);
```

위젯에 이용 가능한 프로퍼티에는 다수가 있는데, 표 A-1부터 A-90까지는 GTK+ 2.10의 위젯과 객체에 관련된 프로퍼티 리스트를 모두 제공한다. 객체 프로퍼티는 부모 위젯으로부터 상속되므로 프로퍼티의 전체 리스트는 위젯의 상속구조를 확인해야 함을 기억하도록 한다. 각 객체에 대한 추가 정보는 API 문서를 참조하라.

프로퍼티	타입	설명
artists	GStrv	애플리케이션에서 사용되는 아트워크를 생성하도록 도와준 사람들의 리스트. 종종 링크로 된 이메일 주소 또는 URL과 같은 정보가 포함된다.
authors	GStrv	애플리케이션의 프로그래밍을 도와준 사람들의 리스트. 종종 프로그래머마다 링크로 된 이메일 주소 또는 URL과 같은 정보가 포함된다.
comments	gchar array	프로그램의 일반적인 기능을 설명하는 짧은 문자열. 메인 대화창에 표시되므로 너무 길어선 안 된다.
copyright	gchar array	애플리케이션에 관한 저작권 정보. 메인 대화창에 표시되므로 너무 길어선 안 된다. 저작권 문자열에 관한 예로 "(C) Copyright 2007 Author"를 들 수 있다.
documents	GStrv	애플리케이션에 관한 문서를 작성하도록 도와준 사람들의 리스트. 종종 링크로 된 각 문서에 대한 이메일 주소 또는 URL이 포함된다.
license	gchar array	애플리케이션의 라이선스 내용. 2차(secondary) 대화상자에 GtkTextView 위젯을 이용해 표시되므로 문자열 길이는 중요하지 않다.
logo	GdkPixbuf	메인 창에 애플리케이션의 로고로 표시할 이미지. 이 값이 설정되지 않으면 gtk_window_get_default_icon_list()가 사용될 것이다.
logo-icon-name	gchar array	아이콘 테마 중에서 메인 About 대화상자의 로고로 사용할 아이콘명. 이 값을 설정 시 logo 프로퍼티보다 우선할 것이다.
name	gchar array	메인 About 대화상자에 표시할 애플리케이션명. 이 프로퍼티를 설정하지 않으면 g_get_application_name()이 사용될 것이다.
translator-credits	gchar array	현재 언어에 대한 번역가(들)에 관한 정보를 보유하는 문자열. Translatable(번역 가능)으로 설정되어야만 각 번역가가 커스텀 문자열을 제공할 수 있다. 주로 번역가마다 링크로 된 이메일 주소 또는 URL이 포함된다.
version	gchar array	사용자가 실행 중인 애플리케이션의 버전.
website	gchar array	애플리케이션에 대한 홈페이지의 URL. 문자열 앞에는 http:// 가 붙어야 한다.
website-label	gchar array	웹 사이트 URL 대신 표시할 라벨. 이 값을 설정하지 않을 경우, website가 URL 라벨로 설정될 것이다.
wrap-license	gboolean	TRUE로 설정 시, license 내용이 래핑될 것이다.

표 A-1. GtkAboutDialog 프로퍼티

프로퍼티	타입	설명
accel-closure	GClosure	키보드 가속기에 대한 변경 내용을 감시해야 하는 closure.
accel-widget	GtkWidget	키보드 가속기에 대한 변경 내용을 감시해야 하는 위젯.

표 A-2. GtkAccelLabel 프로퍼티

프로퍼티	타입	설명
action-group	GtkActionGroup	액션이 속한 액션 그룹. 액션이 액션 그룹에 속하지 않을 경우, NULL로 설정해도 좋다.
hide-if-empty	gboolean	TRUE로 설정 시, 빈 메뉴 프로키는 뷰에서 숨겨질 것이다.
icon-name	gchararray	아이콘 테마에서 사용할 아이콘명. 이 프로퍼티는 stock-id 프로퍼티로 오버라이드된다.
is-important	gboolean	툴바가 GTK_TOOLBAR_BOTH_HORIZ 모드로 되어 있는 경우, 이 프로퍼티를 TRUE로 설정하면 항목에 해당하는 라벨이 표시될 것이다. 그 외의 경우, 어떤 효과도 없다.
label	gchararray	메뉴 항목이나 버튼에 표시할 텍스트. 툴바 항목은 short-label 프로퍼티를 이용한다.
name	gchararray	액션을 구별하는 유일한 문자열.
sensitive	gboolean	TRUE로 설정 시, 액션이 활성화될 것이다. 그 외의 경우, 사용자는 액션과 상호작용할 수 없을 것이다.
short-label	gchararray	툴 항목에 표시할 텍스트. 메뉴 항목과 버튼은 label 프로퍼티를 이용한다.
stock-id	gchararray	액션을 이용하는 위젯에 표시할 스톡 아이콘. 이 프로퍼티는 icon-name보다 우선한다.
tooltip	gchararray	사용자가 툴바 항목 위를 왔다갔다하면 표시되는 액션에 관한 툴팁.
visible	gboolean	TRUE로 설정 시, 액션이 사용자에게 표시되지 않을 것이다.
visible-horizontal	gboolean	TRUE로 설정 시, 툴바가 가로 방향으로 설정되면 액션이 툴바에 표시되지 않을 것이다.
visible-overflown	gboolean	TRUE로 설정 시, 액션이 툴바 오버플로 메뉴에 표시될 것이다. 그 외의 경우, 뷰에서 숨겨질 것이다.
visible-vertical	gboolean	TRUE로 설정 시, 툴바가 세로 방향으로 설정되면 액션이 툴바에 표시될 것이다.

표 A-3. GtkAction 프로퍼티

프로퍼티	타입	설명
name	gchararray	액션 그룹을 구별하는 문자열.
sensitive	gboolean	TRUE로 설정 시, 액션 그룹이 활성화(active 또는 enabled)로 설정된다.
visible	gboolean	TRUE로 설정 시, 액션 그룹이 사용자에게 표시될 것이다.

표 A-4. GtkActionGroup 프로퍼티

프로퍼티	타입	설명
lower	gdouble	조정이 도달할 수 있는 최소 gdouble 값.
page-increment	gdouble	앞으로 또는 뒤로 한 페이지를 이동 시 전환할 증가값(increment).
page-size	gdouble	조정에 대한 페이지 크기. GtkSpinButton에 대해 GtkAdjustment를 이용 시에는 0으로 설정해야 한다.
step-increment	gdouble	각 단계에서 이동하게 될 증가값. GtkSpinButton의 경우를 예로 들면, 화살표 버튼을 누를 때마다 하나의 단계를 취할 것이다.
upper	gdouble	조정이 도달할 수 있는 최대 gdouble 값.
value	gdouble	조정의 현재 값으로, 항상 lower와 upper 사이의 값이어야 한다.

표 A-5. GtkAdjustment 프로퍼티

프로퍼티	타입	설명
bottom-padding	guint	자식 위젯의 하단면을 따라 추가된 패딩.
left-padding	guint	자식 위젯의 좌측면을 따라 추가된 패딩.
right-padding	guint	자식 위젯의 우측면을 따라 추가된 패딩.
top-padding	guint	자식 위젯의 상단면을 따라 추가된 패딩.
xalign(yalign)	gfloat	자식 위젯의 정렬을 정의하는 값으로, 0.0과 1.0 사이 값이어야 하며 1.0은 컨테이너의 하단이나 우측면을 따라 정렬됨을 의미한다.
xscale(yscale)	gfloat	자식 위젯이 추가 공간을 차지하도록 확장시키는 데에 사용되는 값으로, 0.0과 1.0 사이의 값이어야 한다.

표 A-6. GtkAlignment 프로퍼티

프로퍼티	타입	설명
arrow-type	GtkArrowType	GtkArrow가 가리킬 방향.
shadow-type	GtkShadowType	화살표 주변에 위치시킬 그림자(shadow) 타입.

표 A-7. GtkArrow 프로퍼티

프로퍼티	타입	설명
obey-child	gboolean	TRUE로 설정 시, ratio 프로퍼티 대신 자식 위젯이 정의한 영상비를 이용한다.
ratio	gfloat	영상비를 정의하는 0.0001과 10,000 사이의 숫자.
xalign(yalign)	gfloat	컨테이너 내에서 자식 컨테이너의 정렬을 정의하는 값으로, 0.0과 1.0 사이의 값이어야 하며 0.5는 중앙 정렬을 의미한다.

표 A-8. GtkAspectFrame 프로퍼티

프로퍼티	타입	설명
homogeneous	gboolean	TRUE로 설정 시, 모든 자식들이 동일한 크기로 설정될 것이다.
spacing	gint	각 자식과 주위 자식들(neighbors) 사이에 추가할 공간.

표 A-9. GtkBox 프로퍼티

프로퍼티	타입	설명
focus-on-click	gboolean	TRUE로 설정 시, 버튼을 마우스로 클릭하면 버튼이 포커스를 잡을(grab) 것이다.
image	GtkWidget	버튼의 테스트 옆에 표시할 위젯.
image-position	GtkPositionType	라벨을 기준으로 한 image의 위치.
label	gchararray	버튼이 라벨을 포함할 경우 버튼 내에 표시할 텍스트 라벨.
relief	GtkReliefStyle	버튼 주변에 위치시킬 테두리 타입.
use-stock	gboolean	TRUE로 설정 시, 스톡 항목이 버튼의 내용으로 사용될 것이다.
use-underline	gboolean	TRUE로 설정 시, 밑줄 다음에 오는 문자에 니모닉 키보드 가속기가 사용될 것이다.

xalign(yalign)	gfloat	0.0과 1.0 사이의 부동 소수점으로 GtkMisc 또는 GtkAlignment 위젯의 경우 자식 위젯을 정렬하는데 사용되며, 0.5는 중앙 정렬을 의미한다.
표 A-10. GtkButton 프로퍼티		

프로퍼티	타입	설명
layout-style	GtkButtonBoxStyle	자식 버튼에 사용되는 레이아웃의 타입.
표 A-11. GtkButtonBox 프로퍼티		

프로퍼티	타입	설명
day	gint	1과 31 사이에 현재 선택된 일자(day). 0을 선택하면 현재 일자의 선택을 해제할 것이다.
month	gint	0과 11 사이에 현재 선택된 월로, 0은 1월을 나타낸다.
no-month-change	gboolean	TRUE로 설정 시, 사용자는 월을 변경하지 못할 것이다.
show-day-names	gboolean	TRUE로 설정 시, 요일명이 일자(day) 위에 표시될 것이다.
show-heading	gboolean	TRUE로 설정할 경우, 캘린더 헤딩이 표시될 것이다.
show-week-numbers	gboolean	TRUE로 설정할 경우, 현재 월과 연도에 대한 주(week) 번호가 캘린더 좌측면을 따라 표시될 것이다.
year	gint	현재 선택된 연도.
표 A-12. GtkCalendar 프로퍼티		

프로퍼티	타입	설명
cell-background	gchararray	"Red" 또는 "#00CC00"과 같이 배경색을 나타내는 문자열. 이 프로퍼티가 효과를 발휘하기 위해서는 cell-background-set 또한 TRUE로 설정해야 한다.
cell-background-gdk	GdkColor	셀의 배경색.
height	gint	셀의 높이. 셀의 기본 높이를 사용하려면 이 프로퍼티를 -1로 설정한다.
is-expanded	gboolean	행에 자식 행이 있을 경우 행이 확장되면 이 프로퍼티는 TRUE로 설정될 것이다.
is-expander	gboolean	행에 자식 행이 있을 경우 TRUE로 설정하라.
mode	GtkCellRendererMode	셀의 상호작용 모드.
sensitive	gboolean	TRUE로 설정 시, 사용자는 셀과 상호작용을 할 수 있을 것이다.
visible	gboolean	TRUE로 설정 시, 셀이 사용자에게 표시될 것이다.
width	gint	셀의 너비. 셀의 기본 너비를 사용하려면 이 프로퍼티를 -1로 설정한다.
xalign(yalign)	gfloat	셀 내에서 내용의 정렬을 정의하는 값으로, 0.0과 1.0 사이 값이어야 하며 0.5는 중앙 정렬을 의미한다.
xpad(ypad)	guint	셀의 자식 내용에서 각 면(side)마다 위치시켜야 할 수평 및 수직 패딩.
표 A-13. GtkCellRenderer 프로퍼티		

프로퍼티	타입	설명
accel-key	guint	가속기에 대한 키 값. 키 코드의 리스트는 gdkkeysyms.h에서 찾을 수 있다.
accel-mode	GtkCellRendererAccelMode	가속기가 GTK+ 가속기인지 결정하는 플래그 값. GTK_CELL_RENDERER_ACCEL_MODE_GTK의 값을 선택할 경우 이미 사용 중인 가속기를 입력하지 못할 것이다.
accel-mods	GdkModifierType	가속기에 사용할 수정자(modifier).
keycode	guint	키보드 가속기에 대한 하드웨어 키 코드. 키에 이용 가능한 키 값이 있는 경우 accel-key 프로퍼티를 이용해야 한다.

표 A-14. GtkCellRendererAccel 프로퍼티

프로퍼티	타입	설명
has-entry	gboolean	TRUE로 설정 시, 셀을 편집하는 동안 GtkComboBoxEntry 위젯이 표시될 것이다.
model	GtkTreeModel	GtkComboBox 위젯에서 선택(choice)을 정의하는 트리 모델.
text-column	gint	셀을 편집하지 않을 때 표시할 model 내의 열 번호.

표 A-15. GtkCellRendererCombo 프로퍼티

프로퍼티	타입	설명
follow-state	gboolean	TRUE로 설정 시, pixbuf는 GtkCellRendererState를 바탕으로 색이 칠해질 것이다.
icon-name	gchararray	아이콘 테마로부터 표시할 아이콘. stock-id와 pixbuf 프로퍼티가 이 설정보다 우선한다.
pixbuf	GdkPixbuf	셀에 표시할 이미지. 이 프로퍼티는 icon-name 보다 우선한다.
pixbuf-expander-closed	GdkPixbuf	자식 행이 숨겨질 때 익스팬더로 표시할 이미지.
pixbuf-expander-open	GdkPixbuf	자식 행이 표시될 때(visible) 익스팬더로 표시할 이미지.
stock-detail	gchararray	테마 엔진으로 전송된 문자열. 스톡 항목의 렌더링에 관한 추가 정보를 제공한다.
stock-id	gchararray	아이콘으로 사용할 스톡 식별자. 이 프로퍼티는 icon-name보다 우선한다.
stock-size	guint	렌더링할 스톡 아이콘의 크기.

표 A-16. GtkCellRendererPixbuf 프로퍼티

프로퍼티	타입	설명
text	gchararray	진행 막대 위에 그려질 텍스트 문자열. NULL로 설정할 경우 기본 문자열이 표시될 것이다.
value	gint	진행 막대가 채워진 양으로, 0부터 100까지 숫자로 정의되는데 100은 완전히 채워짐을 의미한다.

표 A-17. GtkCellRendererProgress 프로퍼티



프로퍼티	타입	설명
adjustment	GtkAdjustment	스핀 버튼을 편집할 때 그에 관한 정보를 보유하는 조정. 이 프로퍼티는 편집 가능하게(editable) 설정되어야 한다.
climb-rate	gdouble	화살표 버튼을 누르고 있을 때 적용할 가속 속도.
digits	guint	셀을 편집하는 동안 스페인 버튼에 표시할 소수 자리수. 셀이 편집되지 않을 때 표시하는 소수 자리수에는 영향을 미치지 않음을 주목한다. 일반 상태 숫자(state digits)를 설정하기 위해서는 셀 데이터 함수를 이용해야 한다.

표 A-18. GtkCellRendererSpin 프로퍼티

프로퍼티	타입	설명
alignment	PangoAlignment	텍스트 행의 정렬. 이 프로퍼티가 효과를 발휘하기 위해서는 align-set을 TRUE로 설정해야 한다.
attributes	PangoAttrList	렌더러의 텍스트로 적용되는 속성 리스트.
background	gchararray	문자열로 된 셀의 배경색. 이 프로퍼티가 효과를 발휘하기 위해서는 background-set을 TRUE로 설정해야 한다.
background-gdk	GdkColor	셀의 배경색.
editable	gboolean	TRUE로 설정 시, 사용자는 텍스트를 편집할 수 있다. 이 프로퍼티가 효과를 발휘하기 위해서는 editable-set을 TRUE로 설정해야 한다.
ellipsize	PangoEllipsizeMode	전체 문자열을 표시하기에 충분한 공간이 없을 경우 문자열 내에서 텍스트를 타원(ellipse)으로 대체할 공간. 이 프로퍼티가 효과를 발휘하기 위해서는 ellipsize-set을 TRUE로 설정해야 한다.
family	gchararray	Arial 또는 Monospace와 같은 폰트 패밀리명. 이 프로퍼티가 효과를 발휘하기 위해서는 ellipsize-set을 TRUE로 설정해야 한다.
font	gchararray	"Monospace Bold 10"과 같은 폰트 패밀리 문자열. 이 프로퍼티가 효과를 발휘하기 위해서는 font-set을 TRUE로 설정해야 한다.
font-desc	PangoFontDescription	셀에 대한 글꼴을 정의하는 글꼴 설명.
foreground	gchararray	문자열로 된 셀의 전경색. 이 프로퍼티가 효과를 발휘하기 위해서는 foreground-set을 TRUE로 설정해야 한다.
foreground-gdk	GdkColor	셀의 전경색.
language	gchararray	ISO 코드로 된 셀의 텍스트 언어. 대부분의 경우 이 프로퍼티를 사용하지 않아도 될 것이다. 이 프로퍼티가 효과를 발휘하기 위해서는 language-set을 TRUE로 설정해야 한다.
markup	gchararray	Pango 마크업을 포함하는 셀이 렌더링하게 될 텍스트.
rise	gint	텍스트에 대한 양의 또는 음의 오프셋. 이 프로퍼티가 효과를 발휘하기 위해서는 rise-set을 TRUE로 설정해야 한다.
scale	gdouble	gdouble 값으로 된 폰트의 조정 인수(scaling factor). 이 프로퍼티가 효과를 발휘하기 위해서는 scale-set을 TRUE로 설정해야 한다.
single-paragraph-mode	gboolean	TRUE로 설정 시, 모든 텍스트는 강제로 하나의 단락으로 모아질 것이다.
size	gint	PANGO_UNITS의 인수로 스케일링된 텍스트의 글꼴 크기. 이 프로퍼티가 효과를 발휘하기 위해서는 size-set을 TRUE로 설정해야 한다.
size-points	gdouble	포인트로 된 텍스트의 글꼴 크기.
stretch	PangoStretch	텍스트 문자 간 공간을 추가하거나 제거하는 데에 사용되는 플래그. 이 프로퍼티가 효과를 발휘하기 위해서는 stretch-set을 TRUE로 설정해야 한다.

strikethrough	gboolean	TRUE로 설정 시, 텍스트를 거쳐 하나의 행이 위치할 것이다. 이 프로퍼티가 효과를 발휘하기 위해서는 strikethrough-set을 TRUE로 설정해야 한다.
style	PangoStyle	이탤릭체 또는 oblique와 같은 글꼴의 스타일. 이 프로퍼티가 효과를 발휘하기 위해서는 style-set을 TRUE로 설정해야 한다.
text	gchararray	셀에 표시할 텍스트.
underline	PangoUnderline	텍스트 아래 위치시킬 밑줄의 스타일. 이 프로퍼티가 효과를 발휘하기 위해서는 underline-set을 TRUE로 설정해야 한다.
variant	PangoVariant	소문자로 된 문자를 작은 대문자로 렌더링하려면 PANGO_VARIANT_SMALL_CAPS로 설정한다. 이 프로퍼티가 효과를 발휘하기 위해서는 variant-set을 TRUE로 설정해야 한다.
weight	gint	글꼴의 두께. 이 프로퍼티가 효과를 발휘하기 위해서는 weight-set을 TRUE로 설정해야 한다.
width-chars	gint	문자에서 셀의 너비. 이 프로퍼티를 -1로 설정하면 GTK+는 너비를 계산할 것이다.
wrap-mode	PangoWrapMode	텍스트에 사용할 래핑(wrap) 타입. 기본적으로 PANGO_WRAP_CHAR으로 설정된다.
wrap-width	gint	텍스트가 래핑될 너비. 이 프로퍼티를 -1로 설정하면 래핑이 비활성화될 것이다.

표 A-19. GtkCellRendererText 프로퍼티

프로퍼티	타입	설명
activatable	gboolean	TRUE로 설정 시, 사용자가 토글 버튼을 활성화할 수 있다. 그렇지 않으면 토글 버튼은 설정을 표시하는 기능에 그친다.
active	gboolean	TRUE로 설정 시, 토글 버튼이 활성화 상태로(activated) 설정될 것이다.
inconsistent	gboolean	TRUE로 설정 시, 토글 버튼의 상태는 활성화 상태(active), 비활성화(inactive) 상태도 아니다.
indicator-size	gint	체크 버튼 또는 라디오 버튼의 크기. 기본적으로 12 픽셀로 설정된다.
radio	gboolean	TRUE로 설정 시, 토글이 라디오 버튼으로 그려진다. 하지만 라디오 버튼의 기능은 개발자가 구현해야 할 것이다.

표 A-21. GtkCellView 프로퍼티

프로퍼티	타입	설명
active	gboolean	TRUE로 설정 시, 체크 메뉴 항목이 활성화로 설정된다.
draw-as-radio	gboolean	TRUE로 설정 시, 메뉴 항목이 라디오 버튼으로 그려진다. 하지만 라디오 버튼의 기능은 개발자가 구현해야 할 것이다.
inconsistent	gboolean	TRUE로 설정 시, 토글 버튼은 활성화 상태도, 비활성화 상태도 아닌 중간 상태로 표시될 것이다.

표 A-22. GtkCheckMenuItem 프로퍼티

프로퍼티	타입	설명
alpha	guint	선택된 색상의 투명도로, 0은 투명한 수준이고 65,535는 불투명한 수준이다.
color	GdkColor	현재 선택된 색상.
title	gchararray	사용자가 버튼을 클릭하면 표시되는 GtkColorSelectionDialog에 제공할 제목.
use-alpha	gboolean	TRUE로 설정 시, 사용자에게 투명도를 선택할 수 있는 선택권이 부여될 것이다.

표 A-23. GtkColorButton 프로퍼티

프로퍼티	타입	설명
current-alpha	guint	선택된 색상의 투명도로, 0은 투명한 수준이고 65,535는 불투명한 수준이다.
current-color	GdkColor	현재 선택된 색상.
has-opacity-control	gboolean	TRUE로 설정 시, 사용자에게 투명도를 선택할 수 있는 선택권이 부여될 것이다.
has-palette	gboolean	TRUE로 설정 시, 색상 팔레트가 사용자에게 표시될 것이다.

표 A-24. GtkColorSelection 프로퍼티

프로퍼티	타입	설명
active	gint	활성화된 현재 항목의 색인. 선택된 행이 루트 요소가 아닐 경우 이 항목은 <code>gtk_tree_path_get_indices()</code> 가 리턴한 값과 동일할 것이다.
add-tearoffs	gboolean	TRUE로 설정 시, 콤보 박스가 메뉴 스타일을 이용한다면 메뉴에는 테어 오프(tear-off) 메뉴 항목을 가질 것이다.
column-span-column	gint	리스트 내 여러 열에 걸쳐 값을 원한다면 이 프로퍼티를 <code>G_TYPE_INIT</code> 타입의 모델 열을 가리키는 음수가 아닌 (non-negative) 정수로 설정하라. 이 정수는 값이 얼마나 많은 열에 걸쳐 표시될 것인지 정의한다.
focus-on-click	gboolean	TRUE로 설정 시, 사용자가 콤보 박스를 클릭하면 콤보 박스가 포커스를 잡을 것이다.
has-frame	gboolean	TRUE로 설정 시, 선택된 항목 주위로 프레임이 그려질 것이다.
model	GtkTreeModel	콤보 박스에 대한 선택을 보유하는 트리 모델.
popup-shown	gboolean	TRUE로 설정 시, 콤보 박스는 현재 선택을 표시한다. <code>notify</code> 시그널과 이 프로퍼티를 연결하면 사용자에게 팝업 창이 표시될 때 알림을 받을 수 있다.
row-span-column	gint	이 프로퍼티는 수직 방향이라는 점만 제외하면 <code>column-span-column</code> 과 동일한 기능을 수행한다.
tearoff-title	gchararray	콤보 박스 선택을 표시하는 팝업 창을 본래 위치에서 분리할(tear) 때 표시할 제목.
wrap-width	gint	이 프로퍼티를 양의 정수로 설정하면 리스트를 다수의 열에 걸쳐 표시할 수 있다. 이 프로퍼티는 열의 개수를 정의한다.

표 A-25. GtkComboBox 프로퍼티

프로퍼티	타입	설명
text-column	gint	G_TYPE_STRING의 GType으로 된 데이터를 보유하는 GtkTreeModel 내의 열 번호.

표 A-25. GtkComboBox 프로퍼티

프로퍼티	타입	설명
border-width	guint	컨테이너의 자식의 외부를 따라 위치시킬 픽셀 수를 정의하는 정수.
child	GtkWidget	컨테이너의 자식 위젯. 이 프로퍼티를 이용해 컨테이너로 새로운 자식을 추가할 수 있다. 하지만 컨테이너에 다수의 자식이 있는 경우 이 프로퍼티를 이용해선 안 된다.
resize-mode	GtkResizeMode	컨테이너와 그 자식의 크기 조정 요청을 처리하는 방법을 정의한다.

표 A-25. GtkComboBox 프로퍼티

프로퍼티	타입	설명
curve-type	GtkCurveType	곡선의 타입. 예를 들어, 곡선의 타입으로는 선형, 스플라인 보간형(spline interpolated), 자유형(freeform)이 있다.
max-x (max-y)	gfloat	최대 x 또는 y 값을 정의하는 숫자.
min-x (min-y)	gfloat	최소 x 또는 y 값을 정의하는 숫자.

표 A-28. GtkCurve 프로퍼티

프로퍼티	타입	설명
has-separator	gboolean	TRUE로 설정 시, 대화상자의 GtkVBox 위젯과 그 액션 영역 사이에 구분자가 위치할 것이다.

표 A-29. GtkDialog 프로퍼티

프로퍼티	타입	설명
activates-default	gboolean	TRUE로 설정 시, 사용자가 Enter 키를 누르면 창의 기본 창이 활성화될 것이다.
cursor-position	gint	0과 65,535 사이의 정수값으로, GtkEntry 위젯 내에서 현재 커서 위치를 정의한다.
editable	gboolean	TRUE로 설정 시, 사용자는 GtkEntry 위젯의 내용을 편집할 수 있다.
has-frame	gboolean	TRUE로 설정 시, 위젯 주변에 테두리가 위치할 것이다.
inner-border	GtkBorder	텍스트 네 개의 면에 추가될 공간을 정의하는 객체.
invisible-char	guint	visibility가 FALSE로 설정되면 실제 텍스트 대신 이 문자가 표시될 것이다. 이 프로퍼티는 비밀번호 엔트리를 구현하는 데에 종종 사용된다.
max-length	gint	GtkEntry가 수락하게 될 텍스트의 최대 길이로, 제한이 없어야 한다면 0을 이용하라. GtkEntry는 최대 65,535개 문자로 된 문자열을 처리할 수 있다.
scroll-offset	gint	위젯의 좌측으로 스크롤 오프(scroll off)되는 GtkEntry 내용의 픽셀 수를 설명하는 정수.
selection-bound	gint	문자 개수에서 선택내용(selection) 중 cursor-position의 반대편에 해당하는 정수 색인.

text	gchararray	GtkEntry의 현재 내용.
truncate-multiline	gboolean	TRUE로 설정 시, 사용자가 여러 행에 걸쳐 표시된 텍스트를 GtkEntry 위젯으로 붙여 넣으면 첫 번째 행만 삽입될 것이다.
visibility	gboolean	FALSE로 설정할 경우, GtkEntry 위젯 내 모든 문자는 invisibility-char로 대체될 것이다.
width-chars	gint	사용자에게 표시할 문자 개수. GtkEntry는 보통 이 프로퍼티를 수용하도록 크기가 조정될 것이다.
xalign	gfloat	0.0과 1.0 사이 값으로 설명되는 GtkEntry 위젯 내 텍스트의 정렬을 정의하는 값으로, 0.0과 1.0 사이 값이어야 하며 0.5는 중앙 정렬을 의미한다.

표 A-30. GtkEntry 프로퍼티

프로퍼티	타입	설명
inline-completion	gboolean	TRUE로 설정 시, 모든 선택에 공통된 접두사가 텍스트가 추가될 것이다. 이 프로퍼티가 효과를 발휘하기 위해서는 text-column을 설정해야 한다.
minimum-key-length	gint	매치를 모두 표시하기 전에 GtkEntry 위젯에 입력해야 하는 최소 문자 수.
model	GtkTreeModel	모든 가능한 선택을 보유하는 트리 모델. 열들 중 하나는 G_TYPE_STRING의 GType을 가져야 한다.
popup-completion	gboolean	TRUE로 설정 시, 모든 가능한 매치가 팝업 창에 표시될 것이다.
popup-set-width	gboolean	TRUE로 설정 시, 팝업 창의 너비는 GtkEntry 위젯과 동일해질 것이다.
popup-single-match	gboolean	TRUE로 설정 시, 선택이 하나에 불과하더라도 팝업 창이 표시될 것이다. inline-completion이 TRUE로 설정될 경우 이 프로퍼티를 FALSE로 설정해야 한다.
text-column	gint	model 프로퍼티 내에서 G_TYPE_STRING의 GType을 가진 열의 색인. 이 열은 매치의 내용을 제공할 것이다.

표 A-31. GtkEntryCompletion 프로퍼티

프로퍼티	타입	설명
above-child	gboolean	TRUE로 설정 시, 이벤트 박스는 그 내부에서 발생하는 모든 이벤트를 수신할 것이다. 그 외의 경우 이벤트는 자식으로 먼저 간 다음 이벤트 박스로 갈 것이다.
visible-window	gboolean	TRUE로 설정 시, 이벤트 박스가 사용자에게 표시된다.

표 A-32. GtkEventBox 프로퍼티

프로퍼티	타입	설명
expanded	gboolean	TRUE로 설정 시, 익스팬더가 현재 그 자식 위젯을 표시하고 있다.
label	gchararray	익스팬더의 화살표 옆에 표시할 텍스트 문자열.
label-widget	GtkWidget	label이 정의한 텍스트 대신 익스팬더의 화살표 옆에 표시할 GtkWidget.
spacing	gint	익스팬더의 라벨과 그 자식 위젯 사이에 위치시킬 공간의 양을 나타내는 정수.
use-markup	gboolean	TRUE로 설정 시, label 내 어느 Pango 마크업이든 파싱 및 적용될 것이다.
use-underline	gboolean	TRUE로 설정 시, 니모닉 키보드 가속기가 label에서 지원될 것이다.

표 A-33. GtkExpander 프로퍼티

프로퍼티	타입	설명
action	GtkFileChooserAction	파일 선택자가 실행하는 기능.
do-overwrite-confirmation	gboolean	TRUE로 설정 시, GTK_FILE_CHOOSER_ACTION_SAVE의 action은 사용자에게 파일이 이미 존재하는지 확인을 요청할 것이다.
extra-widget	GtkWidget	사용자에게 추가 옵션을 제공하는 데에 사용할 수 있는 보완적 위젯.
file-system-backend	gchararray	파일 시스템 백엔드를 참조하는 이름.
filter	GtkFileFilter	현재 선택된 파일 필터로, 어떤 파일을 표시할 것인지 필터링 시 사용된다.
local-only	gboolean	TRUE로 설정 시, 로컬 파일만 선택으로 표시될 것이다.
preview-widget	GtkWidget	선택된 파일의 내용을 미리보기할 때 이용할 위젯.
preview-widget-active	gboolean	preview-widget을 사용하길 원할 경우 이를 표시하려면 해당 프로퍼티를 TRUE로 설정해야 한다.
select-multiple	gboolean	TRUE로 설정 시, 사용자는 다수의 파일을 선택할 수 있을 것이다.
show-hidden	gboolean	TRUE로 설정 시, 숨겨진 파일과 폴더가 파일 선택자에 표시될 것이다.
use-preview-label	gboolean	TRUE로 설정 시, 현재 미리보기를 실행한 파일명과 함께 라벨이 표시될 것이다.

표 A-34. GtkFileChooser 프로퍼티

프로퍼티	타입	설명
dialog	GtkFileChooserDialog	사용자가 버튼을 클릭하면 표시되는 파일 선택자 대화상자.
focus-on-click	gboolean	TRUE로 설정 시, 사용자가 GtkFileChooserButton 위젯을 클릭하면 해당 위젯이 포커스를 잡을 것이다.
title	gchararray	사용자가 버튼을 클릭하면 표시되는 GtkFileChooserDialog 위젯의 제목.
width-chars	gint	파일 선택자 버튼 내에 라벨의 너비를 문자로 표시한 것.

표 A-35. GtkFileChooserButton 프로퍼티

프로퍼티	타입	설명
font-name	gchararray	"Monospace Bold 10"와 같이 현재 선택된 글꼴명.
show-size	gboolean	TRUE로 설정 시, 글꼴 크기가 글꼴 버튼의 라벨에 표시될 것이다.
show-style	gboolean	TRUE로 설정 시, 글꼴 스타일이 글꼴 버튼의 라벨에 표시될 것이다.
title	gchararray	사용자가 버튼을 클릭하면 표시되는 GtkFontSelectionDialog 위젯의 제목.
use-font	gboolean	TRUE로 설정 시, 글꼴 버튼의 라벨은 그것을 그릴 때 선택된 글꼴을 이용할 것이다.
use-size	gboolean	TRUE로 설정 시, 글꼴 버튼의 라벨은 그것을 그릴 때 선택된 크기를 이용할 것이다.

표 A-36. GtkFontButton 프로퍼티

프로퍼티	타입	설명
font	GdkFont	GtkFontSelection에 현재 선택된 글꼴.
font-name	gchararray	현재 선택된 글꼴을 나타내는 문자열.
preview-text	gchararray	현재 선택된 글꼴의 미리보기로 표시할 텍스트.

표 A-37. GtkFontSelection 프로퍼티

프로퍼티	타입	설명
label	gchararray	GtkFrame의 라벨에 따라 표시되는 텍스트.
label-widget	GtkWidget	label 프로퍼티에서 텍스트 집합 대신 사용할 위젯.
label-xalign	gfloat	라벨 내부에서 라벨의 수평 정렬로, 0.0과 1.0 사이의 값으로 정의된다.
label-yalign	gfloat	라벨 내부에서 라벨의 수직 정렬로, 0.0과 1.0 사이의 값으로 정의된다.
shadow-type	GtkShadowType	GtkFrame이 사용하는 그림자 타입을 정의하는 플래그.

표 A-38. GtkFrame 프로퍼티

프로퍼티	타입	설명
handle-position	GtkPositionType	자식 위젯을 기준으로 한 핸들의 위치.
shadow-type	GtkShadowType	GtkHandleBox 위젯이 사용하는 그림자 타입을 정의하는 플래그.
snap-edge	GtkPositionType	GtkHandleBox 위젯의 도킹(docking)에 사용될 스냅 에지(snap edge)의 위치. 이 프로퍼티가 효과를 발휘하기 위해서는 snap-edge-set을 TRUE로 설정해야 한다.

표 A-39. GtkHandleBox 프로퍼티

프로퍼티	타입	설명
column-spacing	gint	아이콘의 열 사이에 넣을 공간의 양.
columns	gint	아이콘을 정렬시킬 열의 개수. -1로 설정 시 이 값을 선택해줄 것을 GTK+로 알릴 것이다.
item-width	gint	각 항목의 너비를 픽셀로 나타낸 값. -1로 설정 시 이 값을 선택해줄 것을 GTK+로 알릴 것이다.
margin	gint	GtkIconView의 변(edge)을 따라 위치시킬 패딩의 픽셀 수.
markup-column	gint	GtkTreeModel 위젯 내에 마크업에 관한 정보를 보유하는 열. 이 열은 G_TYPE_STRING의 GType을 가져야 한다.
model	GtkTreeModel	GtkIconView이 표시하는 데이터를 정의하는 트리 모델.
orientation	GtkOrientation	서로를 기준으로 한 아이콘과 텍스트의 가로 및 수직 방향.
pixbuf-column	gint	GtkTreeModel 위젯 내에 아이콘을 포함하는 열. 이 열은 GDK_TYPE_PIXBUF의 GType을 가져야 한다.
reorderable	gboolean	TRUE로 설정 시, GtkIconView 위젯 내 항목들은 드래그 앤 드롭을 이용해 정렬할 수 있다.
row-spacing	gint	아이콘의 행 사이에 넣을 공간의 양.
selection-mode	GtkSelectionMode	아이콘 뷰의 선택 모드.
spacing	gint	항목과 그 주위 항목들(neighbors) 사이에 넣을 공간의 픽셀 수.
text-column	gint	GtkTreeModel에서 각 항목의 텍스트를 포함하는 열. 이 열은 G_TYPE_STRING의 GType을 가져야 한다.

표 A-40. GtkIconView 프로퍼티

프로퍼티	타입	설명
file	gchararray	아이콘 이미지의 위치를 명시하는 파일명.
icon-name	gchararray	현재 아이콘 테마로부터의 아이콘. 아이콘 테마가 변경되면 이 또한 자동으로 업데이트된다. 이 아이콘의 크기는 icon-size에서 정의된다.
icon-set	GtkIconSet	아이콘으로 표시할 GtkIconSet. 이 아이콘의 크기는 icon-size에서 정의된다.
icon-size	gint	icon-name, icon-set, stock 중 하나를 이용할 경우 해당 프로퍼티를 이용해 GtkIconSize에서 정의된 아이콘 크기를 명시할 수 있다.
image	GdkImage	아이콘으로 표시할 이미지. GdkPixbuf로 아이콘을 가리길(mask) 원한다면 mask를 이용하라.
mask	GdkPixmap	image 또는 pixmap이 제공하는 아이콘을 가리는 데에 사용되는 pixmap.
pixbuf	GdkPixbuf	아이콘으로 표시할 pixbuf.
pixbuf-animation	GdkPixbufAnimation	아이콘으로 표시할 움직이는(animated) 이미지로, animated pixbuf 객체에 해당한다.
pixel-size	gint	크기는 픽셀로 되어 있어야 한다. 이미지가 icon-name을 이용해 명시된 경우 이 프로퍼티는 icon-size보다 우선한다.
pixmap	GdkPixmap	이미지로 표시할 pixmap. GdkPixbuf로 아이콘을 가리길 원한다면 mask를 이용하라.
stock	gchararray	아이콘으로 표시할 이미지에 대한 스톡 식별자. 이 아이콘의 크기는 icon-size에서 정의된다.
storage-type	GtkImageType	GtkImage가 사용 중인 이미지 보관 타입(image storage type).

표 A-41. GtkImage 프로퍼티

프로퍼티	타입	설명
image	GtkWidget	메뉴 항목의 라벨 옆에 표시할 위젯.

표 A-42. GtkImageMenuItem 프로퍼티

프로퍼티	타입	설명
screen	GdkScreen	GtkInvisible 창이 표시되는 화면.

표 A-43. GtkInvisible 프로퍼티

프로퍼티	타입	설명
angle	gdouble	x 축을 기준으로 0.0과 360.0 사이의 값에 해당하는 텍스트 각도로, 반시계 방향으로 회전한다. 예를 들어, 90.0 값의 경우 텍스트 하단이 화면의 우측면에 위치할 것이다. ellipsize, selectable, wrapped 중 하나를 설정했다면 이 프로퍼티는 무시될 것이다.
attributes	PangoAttrList	라벨의 텍스트에 적용되는 속성의 리스트.
cursor-position	gint	selectable을 TRUE로 설정 시, 이 프로퍼티는 라벨의 텍스트 내에 커서의 위치를 설정할 것이다.
ellipsize	PangoEllipsizeMode	전체 문자열을 표시하기에 공간이 충분치 않은 경우 문자열 내에서 텍스트를 타원으로 대체하기 위한 장소. 이 프로퍼티가 효과를 발휘하기 위해서는 ellipsize-set을 TRUE로 설정해야 한다.
justify	GtkJustification	라벨의 맞춤(justification). 자식 라벨 내에서 라벨을 정렬하는 데에 사용되는 것이 아니라 여러 행에 걸쳐 위치 한 라벨을 맞추는 데에 사용된다.
label	gchararray	라벨이 표시하게 될 텍스트 문자열.



max-width-chars	gint	단일 행에 표시할 최대 문자의 수. -1로 설정 시 자동으로 계산될 것이다. 이 프로퍼티는 max-chars로 오버라이드된다.
mnemonic-keyval	gint	라벨의 니모닉 키보드 가속기에 대한 키 값.
mnemonic-widget	GtkWidget	라벨의 니모닉 키보드 가속기가 활성화될 때 활성화되는 위젯.
pattern	gchararray	표시해야 할 텍스트 문자열로, 밑줄 문자는 밑줄을 그어야 할 문자를 지정한다.
selectable	gboolean	TRUE로 설정 시, 사용자는 마우스로 라벨을 선택할 수 있을 것이다.
selection-bound	gint	선택된 텍스트의 cursor-position 반대쪽 끝의 위치. 선택된 텍스트가 없다면 cursor-position과 동일한 값이 될 것이다.
single-line-mode	gboolean	TRUE로 설정 시, 라벨은 하나의 텍스트 행으로 강제로 이동할 것이다.
use-mnemonic	gboolean	TRUE로 설정 시, 라벨 텍스트 내 Pango 마크업이 파싱될 것이다.
use-underline	gboolean	TRUE로 설정 시, 니모닉 키보드 가속기에 사용할 키를 지정하기 위해 밑줄 문자가 사용될 것이다.
width-chars	gint	라벨의 너비를 문자로 나타낸 값. 이 프로퍼티를 -1로 설정할 경우 자동으로 GTK+가 계산해줄 것이다. 이 프로퍼티는 max-width-chars보다 우선한다.
wrap	gboolean	TRUE로 설정 시, 한 행에 들어가지 않으면 라벨은 래핑될 것이다.
wrap-mode	PangoWrapMode	wrap이 TRUE로 설정되면 실행되어야 하는 래핑 타입.

표 A-44. GtkLabel 프로퍼티

프로퍼티	타입	설명
hadjustment	GtkAdjustment	위젯을 스크롤 시 이용되는 수평적 조정.
height	guint	GtkLayout 위젯의 높이를 픽셀로 나타낸 값. 위젯은 내부적으로(natively) 스크롤링을 지원하므로 높이는 화면의 높이보다 클 수 있다.
vadjustment	GtkAdjustment	위젯을 스크롤 시 이용되는 수직적 조정.
width	guint	GtkLayout 위젯의 너비를 픽셀로 나타낸 값. 위젯은 내부적으로(natively) 스크롤링을 지원하므로 높이는 화면의 너비보다 클 수 있다.

표 A-45. GtkLayout 프로퍼티

프로퍼티	타입	설명
uri	GCharArray	링크 버튼이 방문하는 웹 사이트의 URI. http://www.gtkbook.com 과 같이 완전한 URI여야 한다.

표 A-46. GtkLinkButton 프로퍼티

프로퍼티	타입	설명
tearoff-state	GBoolean	TRUE로 설정 시, 메뉴는 부착된 위젯으로부터 분리할 수 있다.
tearoff-title	GCharArray	메뉴를 그에 부착된 위젯에서 분리할 때 표시할 제목.

표 A-47. GtkMenu 프로퍼티

프로퍼티	타입	설명
child-pack-direction	GtkPackDirection	자식의 메뉴 항목을 패킹하는 방향.
pack-direction	GtkPackDirection	자식 메뉴 항목을 패킹하는 방향.

표 A-48. GtkMenuBar 프로퍼티

프로퍼티	타입	설명
take-focus	GBoolean	TRUE로 설정 시, 메뉴와 하위메뉴는 키보드에서 포커스를 잡을 것이다.

표 A-49. GtkMenuShell 프로퍼티

프로퍼티	타입	설명
menu	GtkMenu	사용자가 툴 버튼 옆의 화살표 모양을 클릭할 때 표시할 메뉴.

표 A-50. GtkMenuToolButton 프로퍼티

프로퍼티	타입	설명
buttons	GtkButtonType	메시지 대화상자의 액션 영역에 표시되는 버튼(들).
image	GtkWidget	GtkMessageDialog에 표시할 위젯 이미지.
message-type	GtkMessageType	GtkMessageDialog가 보고하는 메시지 타입. 메시지 타입은 image가 설정되지 않을 시 대화상자에 표시되는 이미지를 정의한다.
secondary-text	GCharArray	text에 정의된 문자열 아래에 표시되는 2차(secondary) 텍스트.
secondary-use-markup	GBoolean	TRUE로 설정 시, secondary-text 에서 마크업이 파싱될 것이다.
text	GCharArray	대화상자가 표시하는 메인 텍스트로, 이차 텍스트 위에 표시될 것이다.
use-markup	GBoolean	TRUE로 설정 시, text 에서 마크업이 파싱될 것이다.

표 A-51. GtkMessageDialog 프로퍼티

프로퍼티	타입	설명
xalign(yalign)	gfloat	수평 또는 수직 정렬로, 0.0과 1.0 사이의 값으로 정의되는데 0.5는 중앙 정렬을 의미한다.
xpad(ypad)	gint	위젯의 어떤 변으로든 추가되는 패딩의 픽셀 값

표 A-52. GtkMisc 프로퍼티

프로퍼티	타입	설명
enable-popup	gboolean	TRUE로 설정 시, 사용자가 탭 위를 오른쪽 마우스로 클릭하면 다른 페이지를 살펴볼 수 있는 팝업 메뉴가 표시될 것이다.
group-id	gint	GtkNotebook 탭에서 드래그 앤 드롭 연산에 사용되는 정수 그룹 식별자.
homogeneous	gboolean	TRUE로 설정 시, 모든 GtkNotebook 탭의 너비가 동일해질 것이다.
page	gint	현재 선택된 페이지의 색인으로, 0부터 색인된다.
scrollable	gboolean	TRUE로 설정 시, 스크롤 탭의 공간이 충분치 않으면 탭을 스크롤하도록 화살표가 그려질 것이다.
show-border	gboolean	TRUE로 설정 시, 테두리가 표시될 것이다.
show-tabs	gboolean	TRUE로 설정 시, 탭이 사용자에게 표시될 것이다.
tab-border	gint	각 탭 라벨 주위에 위치하는 테두리 너비.
tab-hborder	gint	각 탭 라벨 주위에 위치하는 가로 테두리의 너비.
tab-pos	GtkPositionType	GtkNotebook 자식을 기준으로 한 탭의 위치.
tab-vborder	gint	각 탭 라벨 주위에 위치하는 세로 테두리의 너비.

표 A-53. GtkNotebook 프로퍼티

프로퍼티	타입	설명
user-data	gpointer	GtkObject와 연관된 데이터 조각으로 gpointer 타입이다.

표 A-54. GtkObject 프로퍼티

프로퍼티	타입	설명
max-position	gint	패인(pane)의 최대 위치로, 자식의 타입과 크기를 바탕으로 계산된다.
min-position	gint	패인의 최소 위치로, 자식의 타입과 크기를 바탕으로 계산된다.
position	gint	구분자의 위치를 명시적으로 설정하는 데에 사용되는 프로퍼티로, 0은 상단면 또는 좌측면을 의미한다. 이 프로퍼티가 효과를 발휘하기 위해서는 position-set을 TRUE로 설정해야 한다.

표 A-55. GtkPaned 프로퍼티

프로퍼티	타입	설명
accepts-pdf	gboolean	TRUE로 설정 시, 프린터가 PDF 파일을 수락할 수 있을 것이다.
accepts-ps	gboolean	TRUE로 설정 시, 프린터가 PostScript 파일을 수락할 수 있을 것이다.
backend	GtkPrintBackend	GtkPrinter가 사용하는 프린트 백엔드(print backend).
icon-name	Gchararray	GtkPrinter에 사용할 아이콘명.
is-virtual	gboolean	TRUE로 설정 시, GtkPrinter는 가상 프린터로, 실제 하드웨어를 나타내지 않을 수도 있다.
job-count	gint	현재 GtkPrint가 이용 가능해지길 기다리는 인쇄 작업의 개수.
location	Gchararray	프린터의 위치를 설명하는 문자열.
name	Gchararray	프린터를 식별하는 유일한 이름.
state-message	Gchararray	현재 프린터의 상태에 관한 정보를 추가로 제공하는 문자열.

표 A-56. GtkPrinter 프로퍼티

프로퍼티	타입	설명
page-setup	GtkPageSetup	인쇄 작업과 연관된 페이지 셋업. 이 프로퍼티는 페이지 방향이나 용지 크기와 같은 정보를 보유한다.
printer	GtkPrinter	인쇄 작업을 처리하는 데에 선택된 프린터.
settings	GtkPrintSettings	인쇄 작업에 연관된 인쇄 설정. 이 프로퍼티는 사본 수, 인쇄 품질, 해상도와 같은 정보를 보유한다.
title	Gchararray	실현 가능하도록 인쇄 작업에 주어진 제목. 주로 자신의 애플리케이션에서 설정한 인쇄 작업과 다른 곳에서 설정한 인쇄 작업을 구별한다.
tack-print-settings	gboolean	TRUE로 설정 시, 인쇄 작업이 프린터로 전달된 후에도 status-changed 시그널은 계속 발생할 것이다.

표 A-57. GtkPrintJob 프로퍼티

프로퍼티	타입	설명
current-page	gint	문서로부터 현재 페이지. 이 프로퍼티를 -1로 설정하면 Range 옵션이 선택 가능한 상태가 된다.
page-setup	GtkPageSetup	인쇄 대화상자와 연관된 페이지 셋업. 이 프로퍼티는 페이지 방향이나 용지 크기와 같은 정보를 보유한다.
print-settings	GtkPrintSettings	인쇄 대화상자와 연관된 인쇄 설정. 이 프로퍼티는 사본 수, 인쇄 품질, 해상도와 같은 정보를 보유한다.
selected-printer	GtkPrinter	인쇄 대화상자에서 선택된 프린터.

표 A-58. GtkPrintUnixDialog 프로퍼티

프로퍼티	타입	설명
ellipsize	PangoEllipsizeMode	전체 문자열을 표시하기에 공간이 충분치 않은 경우 문자열 내에서 텍스트를 타원으로 대체하기 위한 장소. 이 프로퍼티가 효과를 발휘하기 위해서는 ellipsize-set을 TRUE로 설정해야 한다.
fraction	gdouble	상태 표시줄이 채워진 양으로, 0.0부터 1.0까지 숫자로 정의되는데 1.0은 완전히 채워짐을 의미한다.
orientation	GtkProgressBarOrientation	진행 막대가 채워지는 방향.
pulse-step	gdouble	진행 막대가 펼칠 때 진행 막대를 따라 움직이는 거리를 설정한다. 예를 들어, pulse-step이 0.1로 설정되면 블록이 진행 막대의 한쪽 끝에서 반대편 끝으로 이동하려면 진행 막대가 10회 펼칠 것이다.
text	gchararray	진행 막대의 상단에 인쇄할 텍스트.

표 A-59. GtkProgressBar 프로퍼티

프로퍼티	타입	설명
current-value	gint	현재 활성화된 액션 그룹의 member에 대한 value.
group	GtkRadioAction	라디오 액션이 속한 라디오 그룹을 명시하는 라디오 액션.
value	gint	라디오 그룹에서 GtkRadioAction에 유일한 정수. 이 프로퍼티를 current-value와 함께 이용하면 현재 활성화된 라디오 액션을 찾을 수 있다.

표 A-60. GtkRadioAction 프로퍼티

프로퍼티	타입	설명
group	GtkWidget	라디오 위젯을 동일한 라디오 그룹 내 다른 위젯으로 연결하는 라디오 버튼, 라디오 메뉴 항목 또는 라디오 툴 버튼.

표 A-61. GtkRadioButton, GtkRadioMenuItem, GtkRadioToolButton 프로퍼티

프로퍼티	타입	설명
adjustment	GtkAdjustment	경계(bound)를 비롯해 GtkRange의 현재 값을 보유하는 조정.
inverted	gboolean	TRUE로 설정 시, 슬라이더는 더 큰 값과 작은 값의 위치를 전환할 것이다.
lower-stepper-sensitivity	GtkSensitivityType	활성화되면 GtkAdjustment 객체의 값을 감소시키는 버튼과 연관된 민감도.
update-policy	GtkUpdateType	GtkRange가 화면에서 어떻게 업데이트되어야 하는지를 정의한다.
upper-stepper-sensitivity	GtkSensitivityType	활성화되면 GtkAdjustment 객체의 값을 증가시키는 버튼과 연관된 민감도.

표 A-62. GtkRange 프로퍼티

프로퍼티	타입	설명
filter	GtkRecentFilter	현재 선택된 파일 필터로, 사용자에게 어떤 자원을 표시하는지 결정하는 데에 사용된다.
limit	gint	GtkRecentChooser에 표시되는 최대 항목의 개수. 제한을 없애려면 -1로 설정하라. 이 프로퍼티는 GtkRecentChooserMenu를 이용 시 설정되어야만 팝업 메뉴가 조작하기 힘들어지는 상황을 피할 수 있다.
local-only	gboolean	TRUE로 설정 시, file:// 접두사가 붙은 자원만 표시될 것이다.
recent-manager	GtkRecentManager	표시해야 할 현재 사용된 자원을 보유하는 관리자. 현재 화면에서 기본 GtkRecentManager를 검색하기 위해서는 gtk_recent_manager_get_default()를 이용할 수 있다.
select-multiple	gboolean	TRUE로 설정 시, 사용자는 리스트에서 다수의 자원을 선택할 수 있다.
show-icons	gboolean	TRUE로 설정 시, 각 자원 옆에 아이콘이 표시되어 MIME 타입 등 자원에 관한 정보를 제공한다.
show-not-found	gboolean	이 프로퍼티를 FALSE로 설정하면 더 이상 이용할 수 없는 파일을 숨길 수 있다. 로컬 자원에만 영향을 미치는 프로퍼티임을 주목해야 한다.
show-private	gboolean	TRUE로 설정 시, 특정 애플리케이션에 private하게 설정된 항목이 표시될 것이다.
show-tips	gboolean	TRUE로 설정 시, 각 항목에 툴팁이 이용 가능하다면 툴팁이 표시될 것이다.
sort-type	GtkRecentSortType	최근 자원 리스트를 정렬하는 데에 사용할 방법.

표 A-63. GtkRecentChooser 프로퍼티

프로퍼티	타입	설명
show-numbers	gboolean	TRUE로 설정 시, 메뉴에서 가장 최근의 자원 10개 앞에 번호가 붙을 것이다.

표 A-64. GtkRecentChooserMenu 프로퍼티

프로퍼티	타입	설명
filename	gchararray	최근 사용된 자원의 리스트를 저장하는 파일의 위치.
limit	gint	최근 사용된 자원의 최대수로, gtk_recent_manager_get_items()가 호출되면 GtkRecentManager가 이 값을 리턴하게 될 것이다.
size	gint	최근 사용된 자원의 리스트 내 총 항목의 개수.

표 A-65. GtkRecentManager 프로퍼티

프로퍼티	타입	설명
lower	gdouble	눈금자(ruler)가 표시하는 가장 작은 값.
max-size	gdouble	눈금자의 최대 크기. 이 값을 0.0으로 설정하면 크기가 제한되지 않는다.
metric	GtkMetricType	눈금자가 사용하는 단위의 타입으로, 픽셀, 인치, 센티미터가 있다.
position	gdouble	눈금자의 마커의 현재 위치.
upper	gdouble	눈금자가 표시하는 가장 큰 값.

표 A-66. GtkRuler 프로퍼티

프로퍼티	타입	설명
digits	gint	표시된 값의 최대 소수 자리수로, 64까지 가능하다. 이 프로퍼티를 -1로 설정 시 이 값을 선택해줄 것을 GTK+로 알릴 것이다.
draw-value	gboolean	TRUE로 설정 시, 슬라이더 옆에 값이 표시될 것이다.
value-pos	GtkPositionType	슬라이더를 기준으로 GtkScale 값의 위치.

표 A-67. GtkScale 프로퍼티

프로퍼티	타입	설명
hadjustment	GtkAdjustment	수평 스크롤바에 대한 조정.
hscrollbar-policy	GtkPolicyType	수평 스크롤바를 항상 표시할 것인지, 항상 숨길 것인지, 아니면 필요할 때만 표시할 것인지 정의한다.
shadow-type	GtkShadowType	자식 위젯 주위에 위치시킬 그림자 타입.
vadjustment	GtkAdjustment	수직 스크롤바에 대한 조정.
vscrollbar-policy	GtkPolicyType	수직 스크롤바를 항상 표시할 것인지, 항상 숨길 것인지, 아니면 필요할 때만 표시할 것인지 정의한다.
window-placement(set)	GtkCornerType	스크롤바를 기준으로 한 자식 위젯의 배치. 이 프로퍼티가 효과를 발휘하기 위해서는 window-placement-set을 TRUE로 설정해야 한다.

표 A-68. GtkScrolledWindow 프로퍼티

프로퍼티	타입	설명
draw	gboolean	TRUE로 설정 시, 화면에 구분자 툴팁이 그려질 것이다. 그 외의 경우 해당 자리에 빈 공간이 추가될 것이다.

표 A-69. GtkSeparatorToolItem 프로퍼티

프로퍼티	타입	설명
ignore-hidden	gboolean	TRUE로 설정 시, 화면에 표시되지 않은 위젯은 그룹의 크기를 계산할 때 무시될 것이다.
mode	GtkSizeGroupMode	크기 그룹이 자식의 크기를(들) 어떻게 결정하는지를 정의하는 플래그.

표 A-70. GtkSizeGroup 프로퍼티

프로퍼티	타입	설명
adjustment	GtkAdjustment	스핀 버튼의 값과 경계에 관한 정보를 보유하는 조정.
climb-rate	gdouble	화살표 버튼을 누를 때 가속 속도.
digits	guint	0과 20 사이의 숫자로, 표시할 값의 소수 자리수에 해당한다.
numeric	gboolean	TRUE로 설정 시, 수치 문자만 스핀 버튼에 의해 실현될 것이다.
snap-to-ticks	gboolean	TRUE로 설정 시, 값이 자동으로 업데이트되고 가장 가까운 단계 증가값(step increment)에 맞추어 조정될 것이다.
update-policy	GtkSpinButtonUpdatePolicy	스핀 버튼이 얼마나 자주 그리고 언제 업데이트될 것인지 결정하는 플래그.
value	gdouble	스핀 버튼이 저장한 현재 값. 스핀 버튼의 조정과 상호작용하는 대신 이 값을 읽고 쓸 수 있다.
wrap	gboolean	스핀 버튼이 그것의 상단 또는 하단 경계에 도달하고 이 프로퍼티를 TRUE로 설정하면 스핀 버튼 값이 반대편 끝으로 래핑할 것이다.

표 A-71. GtkSpinButton 프로퍼티

프로퍼티	타입	설명
has-resize-grip	gboolean	이 프로퍼티를 TRUE로 설정 시 GtkStatusbar 위젯이 그래픽을 표시하면서 사용자가 마우스로 창을 이동시켜 크기를 변경하도록 해준다.

표 A-72. GtkStatusbar 프로퍼티

프로퍼티	타입	설명
blinking	gboolean	TRUE로 설정 시, 이 행위가 지원되는 플랫폼에서는 상태 아이콘이 깜빡거릴 것이다.
file	gchararray	상태 아이콘으로 표시할 아이콘의 위치.
icon-name	gchararray	아이콘 테마 중에서 상태 아이콘으로 표시할 아이콘.
pixbuf	GdkPixbuf	상태 아이콘으로 표시할 이미지.
size	gint	표시할 아이콘의 크기.
stock	gchararray	상태 아이콘으로 표시할 아이콘을 정의하는 스톡 식별자.
storage-type	GtkImageType	표시할 이미지 타입. file, icon-name, pixbuf, stock 중 무엇을 사용할 것인지 식별 시 사용된다.
visible	gboolean	TRUE로 설정 시, 상태 아이콘이 시스템 트레이를 통해 사용자에게 표시될 것이다.

표 A-73. GtkStatusIcon 프로퍼티



프로퍼티	타입	설명
column-spacing	guint	하나의 열과 그 주위 대상들 사이에 추가할 공간의 픽셀 수.
homogeneous	gboolean	TRUE로 설정 시, 모든 셀에 동일한 높이와 너비가 주어질 것이다.
n-columns	guint	GtkTable 내에 총 열의 개수.
n-rows	guint	GtkTable 내에 총 행의 개수.
row-spacing	guint	하나의 행과 그 주위 대상들 사이에 추가할 공간의 픽셀 수.

표 A-74. GtkTable 프로퍼티

프로퍼티	타입	설명
copy-target-list	GtkTargetList	클립보드에서 복사하기와 드래그 앤 드롭 소스에 관한 정보를 저장하는 데에 사용되는 버퍼에 대한 목표 리스트(target list).
cursor-position	gint	버퍼 내에서 커서의 현재 위치. 커서가 이동할 때를 알기 위해서는 notify 시그널을 이용해 이 프로퍼티를 감시할 수 있다.
has-selection	gboolean	TRUE로 설정 시, 텍스트 버퍼가 현재 선택된 텍스트를 갖는다.
paste-target-list	GtkTargetList	클립보드에서 붙여넣기와 드래그 앤 드롭 목적지에 관한 정보를 저장하는 데에 사용되는 버퍼에 대한 목표 리스트(target list).
tag-table	GtkTextTagTable	텍스트 버퍼가 사용하는 모든 텍스트 태그를 보유하는 텍스트 태그 테이블.
text	gchararray	포함된 이미지나 자식 위젯을 제외하고 텍스트 버퍼에 현재 포함된 텍스트.

표 A-75. GtkTextBuffer 프로퍼티

프로퍼티	타입	설명
accepts-tab	gboolean	TRUE로 설정 시, 탭 순서에서 다음 위젯으로 포커스를 주는 대신 Tab 키를 눌러야 텍스트 뷰가 탭 문자를 삽입할 것이다.
buffer	GtkTextBuffer	현재 텍스트 뷰가 표시하는 텍스트 버퍼.
cursor-visible	gboolean	TRUE로 설정 시, 커서가 사용자에게 표시될 것이다.
editable	gboolean	TRUE로 설정 시, 사용자는 텍스트 뷰의 내용을 편집할 수 있을 것이다.
indent	gint	각 문단에 들여쓰기할 픽셀 수로, 기본값은 0으로 설정된다.
justification	GtkJustification	텍스트의 왼쪽, 오른쪽, 또는 중앙 맞춤.
left-margin	gint	텍스트 뷰의 왼쪽면과 내용 사이에 추가할 공간의 픽셀 수.
overwrite	gboolean	TRUE로 설정 시, 새 문자가 이미 존재하는 문자를 덮어쓸 것이다. 그 외의 경우 새 문자가 삽입된다.
pixels-above-lines	gint	각 문단 위에 위치시킬 패딩의 픽셀 수.
pixels-below-lines	gint	각 문단의 아래에 위치시킬 패딩의 픽셀 수.
pixels-inside-wrap	gint	문단 내에서 래핑된 행들 사이에 위치시킬 패딩의 픽셀 수.
right-margin	gint	텍스트 뷰의 우측면과 내용 사이에 추가할 공간의 픽셀 수.
tabs	PangoTabArray	사용자가 Tab 키를 누르면 추가될 내용을 정의하는 탭 배열.

wrap-mode	GtkWrapMode	실행할 래핑 타입.
표 A-76. GtkTextView 프로퍼티		

프로퍼티	타입	설명
active	gboolean	TRUE로 설정 시, GtkToggleAction이 체크되어 그려질 것이다.
draw-as-radio	gboolean	TRUE로 설정 시, GtkToggleAction이 라디오 버튼으로 그려질 것이다.
표 A-77. GtkToggleAction 프로퍼티		

프로퍼티	타입	설명
active	gboolean	TRUE로 설정 시, GtkToggleButton이 체크되어 그려질 것이다.
draw-indicator	gboolean	TRUE로 설정 시, GtkToggleButton의 토글면이 표시될 것이다.
inconsistent	gboolean	TRUE로 설정 시, 토글 버튼의 상태는 활성화 상태도, 비활성화 상태도 아닌 중간 상태가 될 것이다.
표 A-78. GtkToggleButton 프로퍼티		

프로퍼티	타입	설명
active	gboolean	TRUE로 설정 시, GtkToggleToolButton이 체크되어 그려질 것이다.
표 A-79. GtkToggleToolButton 프로퍼티		

프로퍼티	타입	설명
icon-size	GtkIconSize	툴바 아이콘의 크기. 이 프로퍼티는 특수 투바에만 이용해야 한다. 대부분의 경우 사용자가 선택한 테마를 따라야 한다. 이 프로퍼티가 효과를 발휘하기 위해서는 icon-size-set을 TRUE로 설정해야 한다.
orientation	GtkOrientation	툴바의 방향으로, 수평 또는 수직 중 하나다.
show-arrow	gboolean	TRUE로 설정 시, 투바 항목이 알맞게 들어맞지 않으면 화살표가 표시될 것이다. 화살표는 오버플로 투바 항목을 표시하는 팝업 메뉴로 접근하도록 해줄 것이다.
toolbar-style	GtkToolbarStyle	텍스트 또는 아이콘이 표시되는지를 나타내는 투바 스타일.
tooltips	gboolean	TRUE로 설정 시, 투바 항목에 대해 투팁이 표시될 것이다.
표 A-80. GtkToolbar 프로퍼티		

프로퍼티	타입	설명
icon-name	gchararray	아이콘 테마 중에서 표시할 아이콘명. label, icon-widget, stock-id 프로퍼티들은 icon-name보다 우선한다.
icon-widget	GtkWidget	툴 버튼의 아이콘으로 표시할 위젯.
label	gchararray	툴 버튼에 대한 라벨로 표시할 텍스트 문자열.
label-widget	GtkWidget	label 대신 투 버튼의 라벨로 사용할 위젯.
stock-id	gchararray	액션을 이용하는 위젯에 표시할 스톡 아이콘. 이 프로퍼티는 icon-name보다 우선한다.
use-underline	gboolean	TRUE로 설정 시, 밑줄 문자는 밑줄 다음의 문자를 니모닉 키보드 가속기로 지정할 것이다.
표 A-81. GtkToolButton 프로퍼티		

프로퍼티 타입 설명		
is-important	gboolean	툴바가 GTK_TOOLBAR_BOTH_HORIZ의 toolbar-style을 이용할 때 이 프로퍼티를 TRUE로 설정하면 GTK+로 툴 버튼의 라벨을 표시할 것을 알린다. 그 외의 경우 아이콘만 표시될 것이다.
visible-horizontal	gboolean	TRUE로 설정 시, 툴바의 orientation이 GTK_ORIENTATION_HORIZONTAL로 설정되면 툴 항목이 표시될 것이다.
visible-vertical	gboolean	TRUE로 설정 시, 툴바의 orientation이 GTK_ORIENTATION_VERTICAL로 설정되면 툴 항목이 표시될 것이다.

표 A-82. GtkToolItem 프로퍼티

프로퍼티 타입 설명		
child-model	GtkTreeModel	GtkTreeModelFilter에 의해 필터링되는 내용을 보유하는 트리 모델.
virtual-root	GtkTreePath	child-model에서 사용할 루트 행을 가리키는 트리 경로. 트리 모델의 절대적 루트 경로일 필요는 없다.

표 A-83. GtkTreeModelFilter 프로퍼티

프로퍼티 타입 설명		
model	GtkTreeModel	GtkModelSort에 의해 정렬되는 내용을 보유하는 트리 모델.

표 A-84. GtkTreeModelSort 프로퍼티

프로퍼티 타입 설명		
enable-grid-lines	GtkTreeViewGridLines	수평 또는 수직 격자선을 위치시키는 플래그.
enable-search	gboolean	TRUE로 설정 시, 사용자는 키보드를 이용해 GtkTreeView의 내용을 검색할 수 있을 것이다.
enable-tree-lines	gboolean	TRUE로 설정 시, 트리 뷰 내용의 계층구조(hierarchy)를 정의하는 직선(line)이 그려질 것이다.
expand-colum	GtkTreeViewColumn	GtkTreeStore를 이용해 트리 뷰에 대한 익스팬더가 표시되는 트리 뷰 열.
fixed-height-mode	gboolean	TRUE로 설정 시, GTK+는 모든 행의 높이가 동일할 것으로 추측하여 렌더링 속도를 높인다. 이 프로퍼티는 모든 행의 높이가 동일할 것이라고 확신할 때에만 이용해야 한다.
hadjustment	GtkAdjustment	위젯의 스크롤에 사용되는 수평적 조정.
headers-clickable	gboolean	TRUE로 설정 시, 사용자는 열 헤더를 클릭할 수 있을 것이다.
headers-visible	gboolean	TRUE로 설정 시, 열 헤더가 사용자에게 표시될 것이다.
hover-expand	gboolean	TRUE로 설정 시, 마우스 포인터를 행 위에 갖다대면 행이 펼쳐지거나(expand) 접힐(collapse) 것이다.
hover-selection	gboolean	TRUE로 설정 시, GTK_SELECTION_SINGLE 또는 GTK_SELECTION_BROWSE 선택 모드 중 하나를 이용해 마우스 포인터를 행 위에 갖다대면 행이 선택될 것이다.

level-indentation	gint	자식 행에 추가로 들여쓰기할 공간의 픽셀 수. 이 프로퍼티를 0으로 설정하더라도 자식 행은 여전히 기본 패딩으로 들여쓰기가 적용될 것이다.
model	GtkTreeModel	현재 트리 뷰가 표시하는 트리 모델.
reorderable	gboolean	TRUE로 설정 시, 트리 뷰는 사용자 상호작용에 의해 재정렬 가능하다. 예를 들어, 개발자는 드래그 앤 드롭 지원을 구현할 수 있을 것이다.
rubberbanding	gboolean	TRUE로 설정 시, 사용자는 마우스 포인터를 드래그하여 다수의 항목을 선택할 수 있을 것이다.
rules-hint	gboolean	TRUE로 설정 시, 테마 엔진은 교호하는(alternating) 행을 다른 색상으로 그리도록 지시받는다. 이것은 힌트에 불과하며, 테마에서 유효하게 간주되지(honored) 않을지도 모른다는 사실을 명심한다. 또 일부 테마는 기본적으로 교호하는 행을 다른 색상으로 칠한다.
search-column	gint	enable-search가 TRUE로 설정될 때 검색할 행 번호.
show-expanders	gboolean	TRUE로 설정 시, 하나 또는 그 이상의 자식을 가진 행 옆에 익스팬더가 표시될 것이다.
vadjustment	GtkAdjustment	위젯의 스크롤에 사용되는 수직적 조정.

표 A-85. GtkTreeView 프로퍼티

프로퍼티	타입	설명
alignment	gfloat	헤더 내에서 열 제목의 수평적 정렬을 정의하는 값으로, 0.0과 1.0 사이 값이어야 하며 0.5는 중앙 정렬을 의미한다.
clickable	gboolean	TRUE로 설정 시, 사용자는 열 헤더를 클릭할 수 있을 것이다.
expand	gboolean	TRUE로 설정 시, 열이 확장되어 그것이 속한 GtkTreeView로 할당된 추가 공간을 채울 것이다.
fixed-width	gint	열의 고정된 너비를 정의하는 픽셀 수.
max-width	gint	열을 확장할 수 있는 최대 너비의 픽셀 수.
min-width	gint	열을 축소할 수 있는 최소 너비의 픽셀 수.
reorderable	gboolean	TRUE로 설정 시, 열은 드래그 앤 드롭과 같은 방식을 이용해 재정렬할 수 있다.
resizable	gboolean	TRUE로 설정 시, 사용자는 열의 크기를 조정할 수 있을 것이다.
sizing	GtkTreeViewColumnSizing	열에 대한 크기 조정 방식을 설정하는 플래그.
sort-indicator	gboolean	TRUE로 설정 시, 내용에 따라 트리 뷰가 정렬되었음을 나타내는 화살표가 열 헤더에 표시될 것이다.
sort-order	GtkSortType	정렬 표시기(sort indicator)가 표시될 방향을 정의하는 플래그.
spacing	gint	행과 그 주위 행들(neighbors) 사이에 추가되는 공간의 픽셀 수.
title	gchararray	헤더에 표시되는 열의 제목.
visible	gboolean	TRUE로 설정 시, 열이 사용자에게 표시될 것이다.
widget	GtkWidget	이 프로퍼티를 이용하면 열 헤더에 title 문자열 대신 위젯을 위치시킬 수 있다.
width	gint	트리 뷰 열의 너비를 픽셀로 나타낸 값.

표 A-86. GtkTreeViewColumn 프로퍼티

프로퍼티	타입	설명
add-tearoffs	gboolean	TRUE로 설정 시, GtkUIManager가 생성한 메뉴는 팝업 메뉴가 아니라면 테어오프 메뉴 항목을 가질 것이다.
ui	gchararray	메뉴 또는 툴바 사용자 인터페이스를 생성하는 데에 사용되는 XML 문자열.

표 A-87. GtkUIManager 프로퍼티

프로퍼티	타입	설명
hadjustment	GtkAdjustment	네이티브 스크롤링 지원에 사용되는 뷰포트의 수평적 조정.
shadow-type	GtkShadowType	뷰포트의 자식 위젯 주위에 그려지는 그림자 타입.
vadjustment	GtkAdjustment	네이티브 스크롤링 지원에 사용되는 뷰포트의 수직적 조정.

표 A-88. GtkViewport 프로퍼티

프로퍼티	타입	설명
app-paintable	gboolean	TRUE로 설정 시, GTK+는 GtkWidget 위에 직접 그릴 것이다.
can-default	gboolean	TRUE로 설정 시, GtkWidget은 창의 기본 위젯이 될 것이다.
can-focus	gboolean	TRUE로 설정 시, GtkWidget은 창의 포커스를 수락할 수 있을 것이다.
composite-child	gboolean	TRUE로 설정 시, 위젯은 GtkWidget에서 직접적으로 파생되지 않는다.
events	GdkEventManager	GdkEventManager으로부터 위젯이 수신하게 될 모든 이벤트의 비트마스킹(bitmask).
extension-events	GdkExtensionMode	GdkExtensionMode로부터 위젯이 수신하게 될 모든 확장 이벤트(extension event)의 비트마스킹.
has-default	gboolean	TRUE로 설정 시, GtkWidget는 현재 그 부모 창의 기본 위젯에 해당한다.
has-focus	gboolean	TRUE로 설정 시, GtkWidget은 현재 포커스가 있는 창에 해당한다.
height-request	gint	위젯에 요청된 높이. -1로 설정 시 GTK+가 위젯의 높이를 설정하도록 허용한다. 이는 요청에 불과하며 일부 경우 유효하게 간주되지(honored) 않을지도 모른다.
is-focus	gboolean	TRUE로 설정 시, 위젯은 최상위 수준의 창 내에서 포커스를 갖는다.
name	gchararray	동일한 타입으로 된 위젯을 구별하는 데에 사용 가능한 유일한 이름. 종종 자원 파일에서 위젯 스타일을 설정 시 사용된다.
no-show-all	gboolean	TRUE로 설정 시, 위젯은 gtk_widget_show_all()의 호출의 영향을 받지 않는다.
parent	GtkContainer	위젯의 부모 컨테이너.
receives-default	gboolean	TRUE로 설정 시, 위젯에 포커스가 있으면 기본 액션을 수신할 것이다.
sensitive	gboolean	TRUE로 설정 시, 사용자는 위젯과 상호작용이 가능할 것이다.
style	GtkStyle	위젯이 그려지는 방식을 맞춤설정하는 데에 사용되는 위젯과 연관된 스타일.
visible	gboolean	TRUE로 설정 시, 위젯이 화면에 표시될 것이다.
width-request	gint	위젯에 요청된 너비. -1로 설정 시 GTK+가 위젯의 너비를 설정하도록 허용한다. 이는 요청에 불과하며 일부 경우 유효하게 간주되지(honored) 않을지도 모른다.

표 A-89. GtkWidget 프로퍼티

프로퍼티	타입	설명
accept-focus	gboolean	TRUE로 설정 시, 창이 입력을 위해 포커스를 수신할 수 있을 것이다.
allow-grow	gboolean	TRUE로 설정 시, 사용자는 창을 처음 크기보다 크게 조정할 수 있을 것이다.
allow-shrink	gboolean	TRUE로 설정 시, 창의 최소 크기가 정해지지 않을 것이다.
decorated	gboolean	TRUE로 설정 시, 창 관리자가 제목 표시줄이 있는 창을 그릴 것이다.
default-height	gint	창의 기본 높이. 이 높이는 창이 화면에 처음으로 매핑될 때 사용된다.
default-width	gint	창의 기본 너비. 이 너비는 창이 화면에 처음으로 매핑될 때 사용된다.
deletable	gboolean	TRUE로 설정 시, 창의 제목 표시줄은 닫기 버튼을 표시할 것이다.
destroy-with-parent	gboolean	TRUE로 설정 시, 창은 그 부모 창과 함께 소멸될 것이다.
focus-on-map	gboolean	TRUE로 설정 시, 창은 매핑 시 포커스를 수신할 것이다.
gravity	GdkGravity	gtk_window_move()를 이용할 때 창의 참조점.
has-toplevel-focus	gboolean	창의 자식이 포커스를 가질 때 이 프로퍼티는 TRUE로 설정될 것이다.
icon	GdkPixbuf	이 프로퍼티를 이용하는 창 관리자에 창 아이콘으로 표시되는 이미지.
icon-name	gchararray	이 프로퍼티를 이용하는 창 관리자에 창 아이콘으로 표시되는 아이콘 테마로부터 명명된 아이콘.
is-active	gboolean	TRUE로 설정 시, 창은 포커스가 있는 현재 창에 해당한다.
modal	gboolean	TRUE로 설정 시, 사용자는 이 프로퍼티가 리턴할 때까지 부모 창과 상호작용을 할 수 없을 것이다. 부모 창은 transient-for를 이용해 설정된다.
resizable	gboolean	TRUE로 설정 시, 사용자는 다른 GTK+ 설정에서 금지하지 않는 한 창의 크기를 조정할 수 있을 것이다.
role	gchararray	창 관리자가 지난 세션을 복구할 때 사용되는 창을 구별하는 유일한 문자열.
screen	GdkScreen	창이 그려지는 화면.
skip-pager-hint	gboolean	TRUE로 설정 시, 창 관리자는 자체의 pager에서 창을 실현할 것이다.
skip-taskbar-hint	gboolean	TRUE로 설정 시, 창 관리자는 작업 표시줄에 창을 표시할 것이다.
title	gchararray	창 관리자의 작업 표시줄과 제목 표시줄에 표시할 창의 제목.
transient-for	GtkWindow	현재 창의 부모 창. 모달 창이 되도록 해준다.
type	GdkWindowType	최상위 수준의 창 타입 또는 팝업 창 타입.
type-hint	GdkWindowTypeHint	창 관리자에게 창의 목적과 관련해 주어지는 힌트. 다양한 창 관리자에서 창이 그려지는 방식에 영향을 미칠 수도 있다.
urgency-hint	gboolean	TRUE로 설정 시, 사용자는 창에 주의가 필요하다는 통지를 받을 것이다.
window-position	GtkWindowPosition	처음 매핑될 때 창의 위치. 이것은 힌트에 불과하며, 창 관리자는 이를 유효하게 간주하지(honored) 않을 지도 모른다.

표 A-90. GtkWindow 프로퍼티

자식 위젯 프로퍼티

GTK+에서 컨테이너의 모든 자식으로 할당되는 프로퍼티를 가진 컨테이너는 몇 개에 불과하다. 표 A-91부터 A-100까지 그러한 프로퍼티를 열거하겠다.

표 A-91. GtkAssistant 자식 프로퍼티

프로퍼티	타입	설명
complete	gboolean	TRUE로 설정 시, 페이지가 완전한 것으로(complete) 설정되고, 탐색(navigation) 버튼이 사용자와 상호작용하도록 설정될 것이다.
header-image	GdkPixbuf	페이지 헤더 옆에 표시되는 이미지.
page-type	GtkAssistantPageType	사용할 버튼의 타입.
sidebar-image	GdkPixbuf	페이지 옆에 사이드바(sidebar)로 표시되는 이미지.
title	gchararray	헤더에 페이지 제목으로 표시되는 문자열.

표 A-91. GtkAssistant 자식 프로퍼티

프로퍼티	타입	설명
expand	gboolean	TRUE로 설정 시, 박스가 커지면 자식 위젯에 추가 공간이 생길 것이다. 자식 위젯은 스스로 또는 패딩을 이용해 추가 공간을 차지할 수 있다.
fill	gboolean	TRUE로 설정 시, 자식으로 할당된 추가 공간을 위젯이 차지할 것이다. 그 외의 경우 패딩이 차지할 것이다.
pack-type	GtkPackType	자식이 이용하는 패킹의 타입.
padding	guint	자식 위젯과 그 주위 위젯들(neighbors) 사이에 패딩의 픽셀 수.
position	gint	박스 내에서 자식의 위치로, 0부터 색인된다.

표 A-92. GtkBox 자식 프로퍼티

프로퍼티	타입	설명
secondary	gboolean	TRUE로 설정 시, 자식 버튼은 2차 버튼 그룹에 위치할 것이다.

표 A-93. GtkButtonBox 자식 프로퍼티

프로퍼티	타입	설명
x (y)	gint	GtkFixed 위젯 내에서 자식 위젯의 가로 및 세로 위치.

표 A-94. GtkFixed 자식 프로퍼티

프로퍼티 타입 설명		
x (y)	gint	GtkLayout 위젯 내에서 자식 위젯의 가로 및 세로 위치.
표 A-95. GtkLayout 자식 프로퍼티		

프로퍼티	타입	설명
bottom-attach	gint	자식 위젯의 하단면이 붙은 행.
left-attach	gint	자식 위젯의 좌측면이 붙은 열.
right-attach	gint	자식 위젯의 우측면이 붙은 열.
top-attach	gint	자식 위젯의 상단면이 붙은 행.
표 A-96. GtkMenu 자식 프로퍼티		

프로퍼티	타입	설명
detachable	gboolean	TRUE로 설정 시, 사용자는 부모 노트북에서 탭을 분리할 수 있을 것이다.
menu-label	gchararray	노트북의 탭 선택 메뉴에서 탭에 표시할 문자열.
position	gint	노트북 내에서 자식의 현재 위치로, 0부터 색인된다.
reorderable	gboolean	TRUE로 설정 시, 사용자는 현재 탭의 위치를 변경할 수 있을 것이다.
tab-expanded	gboolean	TRUE로 설정 시, 노트북에 할당된 추가 공간을 차지하도록 자식의 탭이 확장될 것이다.
tab-fill	gboolean	TRUE로 설정 시, 자식의 탭이 그에 할당된 추가 공간을 차지할 것이다.
tab-label	gchararray	자식의 탭 라벨로 표시할 문자열. 자신만의 라벨 위젯을 생성하고자 한다면 이 프로퍼티를 설정하지 않은 채 남겨두어도 좋다.
tab-pack	GtkPackType	자식을 추가하는 데에 사용된 패킹 타입.
표 A-97. GtkNotebook 자식 프로퍼티		

프로퍼티	타입	설명
resize	gboolean	TRUE로 설정 시, 자식 위젯이 전체 패인을 차지하도록 크기가 조정될 것이다.
shrink	gboolean	TRUE로 설정 시, 요청된 자식 위젯의 크기보다 작은 크기로 패인을 조정할 수 있다.
표 A-98. GtkPaned 자식 프로퍼티		



프로퍼티	타입	설명
bottom-attach	guint	자식 위젯의 하단면이 붙은 행.
left-attach	guint	자식 위젯의 좌측면이 붙은 열.
right-attach	guint	자식 위젯의 우측면이 붙은 열.
top-attach	guint	자식 위젯의 상단면이 붙은 행.
x-options (y-options)	GtkAttachOptions	위젯에 제공된 수평 및 수직 부착(attach) 옵션.
x-padding (y-padding)	guint	자식 위젯의 면들 중 한 면에 추가할 수평 또는 수직 패딩.

표 A-99. GtkTable 자식 프로퍼티

프로퍼티	타입	설명
expand	gboolean	TRUE로 설정 시, 툴바가 커지면 툴 항목은 공간이 추가될 것이다.
homogeneous	gboolean	TRUE로 설정 시, 툴 항목은 이 프로퍼티 집합을 가진 다른 모든 항목들과 크기가 무조건적으로(force) 같아질 것이다.

표 A-100. GtkToolbar 자식 프로퍼티

## Notes

# FoundationsofGTKDevelopment:Appendix B

## 부록 B GTK+ 시그널

### GTK+ 시그널

GTK+는 시그널과 콜백 함수에 의존하는 시스템이다. 시그널은 사용자가 특정 액션을 실행했음을 애플리케이션에게 알리는 것이다. 시그널이 발생하면 개발자는 GTK+로 콜백 함수라 불리는 함수를 실행할 것을 알릴 수 있다.

시그널을 연결하기 위해서는 `g_signal_connect()`를 이용할 수 있다. 이 함수는 네 개의 매개변수를 수락한다. 첫 번째는 시그널을 감시하는 GObject다. 두 번째 매개변수 `signal_name`은 시그널을 나타내는 문자열로, 시그널명의 리스트는 이 부록을 통해 찾아볼 수 있다.

```
gulong g_signal_connect (gpointer object,
                        const gchar *signal_name,
                        GCallback handler,
                        gpointer data);
```

세 번째 매개변수는 시그널이 발생할 때 호출되는 콜백 함수명이다. 각 콜백 함수에 대한 형태는 GTK+ API 문서에서 찾을 수 있다. 하지만 함수 프로토타입 중 다수는 문서가 완전하지 않으므로 비표준 매개변수에 대한 정보는 이 부록의 시그널 참조 표에서 찾을 수 있다.

`g_signal_connect()`의 마지막 매개변수는 콜백 함수로 임의의 포인터 타입의 데이터를 전송할 수 있도록 해준다. 이는 `gpointer`는 C의 void 포인터 타입과 동일하기 때문에 가능한 일이다.

객체와 데이터 매개변수의 순서가 바뀐다는 점만 제외하면 `g_signal_connect()`와 동일하게 작용하는 `g_signal_connect_swapped()`를 이용할 수도 있다. 이는 데이터 매개변수 포인터에서 함수를 호출하도록 해준다.

이 부록에서는 GTK+ 객체와 위젯에서 이용 가능한 이벤트와 시그널의 전체 리스트를 제공한다. 첫 번째 절은 GtkWidget과 파생 클래스에서 이용할 수 있는 GDK 이벤트 타입에 관한 정보를 제공하고자 한다. 그 다음으로 GTK+에서 시그널을 가진 객체에 대한 설명과 모든 시그널명의 리스트를 제공한다.

### 이벤트

이벤트는 X Windows System이 발생시키는 특별한 타입의 시그널이다. 우선 시그널이 발생하면 GLib가 제공하는 시그널 시스템에 의해 해석되도록 창 관리자로부터 자신의 애플리케이션으로 전송된다.

이를 통해 우리는 일반 시그널과 동일한 시그널 연결 및 콜백 함수 메서드를 사용할 수 있다. 한 가지 차이가 있다면 이벤트 콜백 함수는 gboolean 값을 리턴한다는 점이다. TRUE를 리턴할 경우 어떤 액션도 발생하지 않을 것이다. 반대로 FALSE를 리턴하면 GTK+는 이벤트를 계속해서 처리할 것이다.

시그널 명	GdkEvent Type 값	설명
delete-event	GDK_DELETE	창 관리자가 최상위 수준의 창을 소멸할 것을 요청하였다. 창이 삭제되었는지 확인할 때 사용할 수 있다.
destroy-event	GDK_DESTROY	위젯의 GdkWindow가 소멸되었다. 위젯은 주로 시그널이 발생되기 전에 연결 해제되므로 이 시그널을 사용해서는 안 된다.
expose-event	GDK_EXPOSE	위젯의 새로운 일부가 표시되었으며 그릴 필요가 있다. 창이 이전에 다른 객체에 의해 가려진 경우 발생한다.
motion-notify-event	GDK_MOTION_NOTIFY	위젯 주위영역(proximity)에서 마우스 커서가 이동하였다.
button-press-event	GDK_BUTTON_PRESS	마우스 버튼이 한 번 클릭되었다. GDK_2BUTTON_PRESS와 GDK_3BUTTON_PRESS 이벤트를 따라 발생하였다.
button-press-event	GDK_2BUTTON_PRESS	마우스 버튼이 두 번 클릭되었다. 이 또한 GDK_BUTTON_PRESS를 발생시키므로 개발자는 콜백 함수에서 이벤트 타입을 확인해야 한다. button-press-event GDK_3BUTTON_PRESS 마우스 버튼이 세 번 클릭되었다. 이 또한 GDK_BUTTON_PRESS를 발생시키므로 개발자는 콜백 함수에서 이벤트 타입을 확인해야 한다.
button-release-event	GDK_BUTTON_RELEASE	앞서 클릭한 마우스 버튼이 해제되었다.
key-press-event	GDK_KEY_PRESS	키보드 키가 눌러졌다. 키 누름으로 인해 텍스트가 입력되거나 액션이 취해지는 것을 막기 위해서는 TRUE를 리턴한다.
key-release-event	GDK_KEY_RELEASE	앞서 누른 키보드 키가 해제되었다. 보통 key-press-event처럼 유용하진 않다.
enter-notify-event	GDK_ENTER_NOTIFY	마우스 커서가 위젯의 주위영역으로 들어갔다.
leave-notify-event	GDK_LEAVE_NOTIFY	마우스 커서가 위젯의 주위영역에서 벗어났다.
focus-in-event	GDK_FOCUS_CHANGE	키보드 포커스가 창의 다른 위젯에서 해당 위젯으로 들어갔다.
focus-out-event	GDK_FOCUS_CHANGE	키보드 포커스가 창 내의 다른 위젯으로 떠났다.

configure-event	GDK_CONFIGURE	위젯의 크기, 위치, 쌓는(stacking) 순서가 변경되었다. 보통 위젯에 새로운 크기가 할당될 때 발생한다.
map-event	GDK_MAP	위젯이 디스플레이 위에 매핑되었다. 위젯이 실현되었음을 의미하기도 한다.
unmap-event	GDK_UNMAP	위젯이 디스플레이에서 매핑 해제되었다.
property-notify-event	GDK_PROPERTY_NOTIFY	위젯의 프로퍼티가 변경되거나 삭제되었다. GObject가 저장한 구체적인 위젯 프로퍼티의 변경내용을 추적 시 이 시그널을 이용할 수 있다.
selection-clear-event	GDK_SELECTION_CLEAR	애플리케이션이 더 이상 선택내용에 대한 소유권을 갖고 있지 않으므로 삭제되어야 한다.
selection-request-event	GDK_SELECTION_REQUEST	위젯의 선택내용이 다른 애플리케이션에서 요청되었다.
selection-notify-event	GDK_SELECTION_NOTIFY	선택내용의 소유주가 선택내용의 변환 요청에 응답하였다.
proximity-in-event	GDK_PROXIMITY_IN	터치화면의 펜과 같은 입력 장치가 감지 표면(sensing surface)과 접촉되었다.
proximity-out-event	GDK_PROXIMITY_OUT	터치화면의 펜과 같은 입력 장치와 감지 표면의 접촉이 끊겼다.
event	GDK_DRAG_ENTER	드래그 액션이 진행 중일 때 마우스 포인터가 위젯으로 들어왔다.
event	GDK_DRAG_LEAVE	드래그 액션이 진행 중일 때 마우스 포인터가 위젯에서 벗어났다.
event	GDK_DRAG_MOTION	드래그 액션이 진행 중일 때 마우스 포인터가 위젯 내에서 이동하였다.
event	GDK_DRAG_STATUS	드래그 액션의 현재 상태가 변경되었다.
event	GDK_DROP_START	위젯에서 드롭 액션이 시작되었다.
event	GDK_DROP_FINISHED	위젯에서 드롭 액션이 완료되었다.
client-event	GDK_CLIENT_EVENT	다른 애플리케이션으로부터 위젯에 대한 이벤트가 수신되었다.
visibility-notify-event	GDK_VISIBILITY_NOTIFY	위젯의 가시성(visibility)이 변경되었다. 예를 들어, 위젯의 일부분이 가려지거나 개방되었다.
no-expose-event	GDK_NO_EXPOSE	그리기 가능한 영역의 일부가 복사되었을 때 소스 부분(source region)을 모두 이용할 수 있었다.

scroll-event	GDK_SCROLL	위젯이 한 방향으로 스크롤되었다. 이는 위젯의 시각적 영역을 업데이트하도록 해준다.
window-state-event	GDK_WINDOW_STATE	위젯의 상태가 변경되었다. 위젯이 최상위 수준의 창인 경우, 위젯이 최소화 혹은 최대화되거나, sticky로 만들어지거나, 아이콘으로 만들어질 때 이런 일이 발생할 수 있다.
event	GDK_SETTING	위젯에 대한 설정이 추가, 제거, 또는 수정되었다.
event	GDK_OWNER_CHANGE	위젯의 소유자가 변경되었다. 이 이벤트는 GTK+ 2.6 버전에서 소개되었다.
grab-bracket-event	GDK_GRAB_BROKEN	위젯이 포인터 또는 키보드에 의해 잡혔으나(grab) 깨졌다. 이는 창이 안보이거나(invisible) 사용자가 반복하여 잡기를 시도할 때 발생할 수 있다. 이 이벤트는 GTK+ 2.8 버전에서 소개되었다.

표 B-1. GtkWidget 이벤트 타입

### 위젯 시그널

표 B-2부터 B-69까지는 GTK+에서 시그널을 가진 각 클래스에 관련된 시그널을 모두 제공한다. 시그널명이 표준 시그널 프로토타입을 따르지 않으면 추가 매개변수를 열거하였는데, 이러한 추가적 매개변수는 객체 자체와 사용자 데이터 포인터를 포함하지 않는다.

시그널명 설명	
activate	연관된 메뉴 또는 툴바 항목이 트리거되었다.

표 B-2. GtkAction 시그널

시그널명	추가 매개변수	설명
connect-proxy	GtkAction *action GtkWidget *proxy	GtkAction 객체와 연관된 위젯 간에 프로퍼티를 동기화하는 데에 사용되는 프록시가 추가되었다.
disconnect-proxy	GtkAction *action GtkWidget *proxy	GtkAction 객체와 연관된 위젯 간에 프로퍼티를 동기화하는 데에 사용되는 프록시가 제거되었다.
post-activate	GtkAction *action	시그널이 발생한 후에 액션 그룹에 포함된 액션이 활성화되었다.
pre-activate	GtkAction *action	이 시그널이 발생한 후에 액션 그룹에 포함된 액션이 활성화될 것이다.

표 B-3. GtkActionGroup 시그널

시그널명	설명
changed	value 프로퍼티를 제외하고 하나 또는 그 이상의 조정의 프로퍼티가 변경되었다.
value-changed	조정의 value 프로퍼티가 변경되었다.

표 B-4. GtkAdjustment 시그널

시그널명	추가 매개 변수	설명
apply	None	GtkAssistant 페이지에서 Apply 버튼 또는 Forward 버튼이 클릭되었다.
cancel	None	GtkAssistant 페이지에서 Cancel 버튼이 클릭되었다.
close	None	GtkAssistant의 마지막 페이지에서 Close 버튼 또는 Apply 버튼이 클릭되었다.
prepare	GtkWidget *page	새로운 페이지가 표시되기 직전이다. 이 시그널은 페이지가 사용자에게 표시되기 전에 개발자가 원하는 준비 작업을 실행할 수 있도록 확보하기 위해 발생되었다.

표 B-5. GtkAssistant 시그널

시그널명	설명
activate	이 시그널은 버튼을 애니메이션으로 만드는 데에 사용된다. 이 시그널로 절대 연결해선 안 된다! 대신 clicked 시그널을 사용하라.
clicked	버튼이 클릭 또는 클릭 해제되었다.

표 B-6. GtkButton 시그널

시그널명	설명
day-selected	사용자가 캘린더에서 일자를 선택하였다.
day-selected-double-click	사용자가 더블 클릭하여 일자를 선택하였다. 보완용(supplementary) 위젯이 존재할 경우 그 위젯들을 강제로 업데이트하는 데에 사용한다.
month-changed	사용자가 화살표 버튼 중 하나를 이용해 캘린더에서 새로운 월을 선택하였다.
next-month	표시된 월이 증가하였다. 사용자가 월을 수동으로 변경할 때에만 호출될 것이다.
next-year	표시된 연도가 증가하였다. 사용자가 연도를 수동으로 변경할 때에만 호출될 것이다.
prev-month	표시된 월이 감소하였다. 사용자가 월을 수동으로 변경할 때에만 호출될 것이다.
prev-year	표시된 연도가 감소하였다. 사용자가 연도를 수동으로 변경할 때에만 호출될 것이다.

표 B-7. GtkCalendar 시그널

시그널명	설명
editing-done	사용자가 셀의 텍스트 내용 편집을 완료하였다. 이 시그널은 셀 렌더러에게 그 값을 업데이트할 것을 알린다.
remove-widget	셀이 편집을 완료하였으므로 이제 텍스트 편집 위젯을 소멸시킬 수 있다.

표 B-8. GtkCellEditable 시그널

시그널명	추가 매개 변수	설명
editing-cancelled	None	사용자가 셀의 편집을 취소하기로 선택했다. Escape 키를 누르는 등의 상황에서 발생하도록 설정할 수 있다.
editing-started	GtkCellEditable *editable gchar *path	셀이 편집 가능하게 되었다. 이 시그널을 이용해 셀 내용에 연관된 기본 타입이 아닌 다른 타입으로 된 편집 위젯을 추가할 수 있다.

표 B-9. GtkCellRenderer 시그널

시그널명	추가 매개 변수	설명
accel-cleared	<code>gchar *path_string</code>	가속기가 셀에서 제거되었다. 셀은 기본 텍스트 타입으로 리셋되어 셀이 비었다거나 기본 값으로 리턴해야 함을 사용자에게 알려야 한다.
accel-edited	<code>gchar *path_string</code> <code>guint accel_key</code> <code>GdkModifierType accel_mods</code>	사용자가 선택한 가속기가 변경되었다. 콜백 함수는 새로운 선택내용을 즉시 적용하기에 충분한 정보를 제공한다.

표 B-10. GtkCellRendererAccel 시그널

시그널명	추가 매개 변수	설명
edited	<code>gchar *path_string</code> <code>gchar *new_text</code>	렌더러의 텍스트 내용이 변경되었다. 콜백 함수는 셀의 경로와 셀의 새로운 내용을 수신한다.

표 B-11. GtkCellRendererText 시그널

시그널명	추가 매개 변수	설명
toggled	<code>gchar *path_string</code>	셀이 활성화 또는 비활성화되었다. 렌더러를 라디오 버튼으로 표시하도록 설정할 경우 처음에 활성화되었던 렌더러를 업데이트해야 할 것이다.

표 B-12. GtkCellRendererToggle 시그널

시그널명	설명
toggled	체크 박스의 상태가 변경되었다. 현재 상태를 알아내기 위해서는 <code>GtkCheckMenuItem</code> 클래스의 <code>active</code> 프로퍼티를 확인해야 할 것이다.

표 B-13. GtkCheckMenuItem 시그널

시그널명	설명
color-set	사용자가 새로운 색상을 선택하였다. 색상을 프로그램적으로 변경하면 이 시그널이 발생하지 않을 것이다! 모든 변경내용을 확인하기 위해서는 <code>GtkColorButton</code> 의 <code>color</code> 와 <code>alpha</code> 프로퍼티를 추적해야 할 것이다.

표 B-14. GtkColorButton 시그널

시그널명	설명
color-changed	선택된 색상이 변경되었다. 사용자가 또는 코드에서 이러한 변경을 실행된 경우 해당 시그널이 발생한다.

표 B-15. GtkColorSelection 시그널

시그널명		설명
changed		사용자가 리스트에서 다른 항목을 선택하였거나, 개발자의 코드가 <code>gtk_combo_box_set_active_iter()</code> 를 호출하였다. 이 시그널은 사용자가 <code>GtkComboBoxEntry</code> 로 타이핑 시에도 발생한다.

표 B-16. GtkComboBox 시그널

시그널명	추가 매개 변수	설명
add	GtkWidget *child	자식 위젯이 컨테이너로 추가 또는 패킹되었다. <code>gtk_container_add()</code> 를 명시적으로 호출하지 않았지만 위젯의 내장된 패킹 함수를 사용하더라도 이 시그널이 발생한다.
check-resize	None	자식 위젯이 추가되기 전에 컨테이너의 크기를 조정할 필요가 있는지 검사하였다.
remove	GtkWidget *child	자식 위젯이 컨테이너로부터 제거되었다.
set-focus-child	GtkWidget *child	컨테이너의 자식 위젯이 창 관리자로부터 포커스를 받았다.

표 B-17. GtkContainer 시그널

시그널명	설명
curve-type-changed	<code>gtk_curve_set_gamma()</code> , <code>gtk_curve_reset()</code> , <code>gtk_curve_set_curve_type()</code> 중 하나가 호출되었다.

표 B-18. GtkCurve 시그널

시그널명	추가 매개 변수	설명
close	None	GtkDialog 객체가 닫혔다.
response	gint response	대화상자가 삭제(delete) 이벤트를 수신하였거나 개발자가 <code>gtk_dialog_response()</code> 를 호출하여 GtkDialog의 액션 영역 내 버튼이 활성화되었다. 삭제 이벤트들은 <code>GTK_RESPONSE_NONE</code> 의 응답 식별자가 발생하도록 야기한다. 그 외의 경우, 응답 식별자는 이미 정의되어 있을 것이다.

표 B-19. GtkDialog 시그널

시그널명	추가 매개 변수	설명
changed	None	사용자가 편집 가능한 위젯의 내용을 변경하였다.
delete-text	gint start_pos gint end_pos	위젯 내 두 개의 위치 사이의 텍스트를 사용자가 삭제하였다.
insert-text	gchar *new_text gint text_length gint *position	위젯 내 주어진 위치에 사용자가 텍스트를 삽입하였다.

표 B-20. GtkEditable 시그널

시그널명	추가 매개 변수	설명
activate	None	GtkEntry 위젯에 포커스가 있을 때 Enter 키가 눌러졌다. 활성화되면 개발자는 GtkEntry 와 연관된 대화상자의 기본 버튼을 실행해야 한다.
backspace	None	Backspace 키가 눌러졌다. 커서 좌측에 문자가 있을 경우 모두 삭제된다.
copy-clipboard	None	선택된 텍스트가 클립보드로 복사되었다.
cut-clipboard	None	선택된 텍스트가 클립보드로 복사된 후 GtkEntry 위젯으로부터 제거되었다.
delete-from-cursor	GtkDeleteType type gint num_deletions	텍스트가 커서 주위에서 삭제되었다.
insert-at-cursor	gchar *new_text	텍스트가 커서의 위치에 삽입되었다.
move-cursor	GtkMovementStep step gint num_steps gboolean extended	커서가 특정 거리만큼 이동하였다. 콜백 함수는 선택내용이 확장되었는지 여부를 수신한다.
paste-clipboard	None	클립보드의 텍스트가 GtkEntry로 삽입되었다.
populate-popup	GtkMenu *popup_menu	사용자가 오른쪽 마우스 버튼을 클릭해 팝업 메뉴가 표시되었다.
toggle-overwrite	None	덮어쓰기(overwrite) 프로퍼티를 토글하기 위해 Insert 키가 눌러졌거나, 프로퍼티가 명시적으로 변경되었다.

표 B-21. GtkEntry 시그널

시그널명	추가 매개 변수	설명
action-activated	gint index	주어진 index의 액션 항목이 팝업 리스트에서 선택되었다.
insert-prefix	gchar *prefix	GtkEntryCompletion이 제공한 자동 완성 기능이 활성화되었다. 이는 위젯이 표시하는 기본 접두사를 변경하도록 해준다.
match-selected	GtkTreeModel *model GtkTreeIter *match	사용자가 항목의 리스트에서 매치를 선택하였고, 이는 주어진 GtkTreeModel에 의해 정의된다.

표 B-22. GtkEntryCompletion 시그널

시그널명	설명
activate	익스팬더가 토글되었다. 위젯이 확장되거나 축소될 때 모두 이 시그널이 발생한다.

표 B-23. GtkExpander 시그널



시그널명	설명
confirm-overwrite	사용자가 이미 존재하는 파일명으로 파일을 저장하길 원한다. 개발자는 사용자의 선택을 수락하려면 GTK_FILE_CHOOSER_CONFIRMATION_ACCEPT_FILENAME을, 덮어쓸 것인지 확인하는 기본 대화상자를 표시하려면 GTK_FILE_CHOOSER_CONFIRMATION_CONFIRM을, 사용자가 다른 이름을 선택하도록 하려면 GTK_FILE_CHOOSER_CONFIRMATION_SELECT_AGAIN을 리턴해야 한다.
current-folder-changed	GtkFileChooser가 표시하는 현재 폴더가 변경되었다. 사용자가 폴더를 변경하거나, 북마크가 클릭되거나, 함수 호출이 명시적으로 폴더를 변경하는 경우를 예로 들 수 있겠다.
file-activated	사용자가 리스트에서 파일을 더블 클릭하거나 Enter 키를 눌렀다. 주로 GtkFileChooserDialog에 의해 내부적으로만 사용된다.
selection-changed	선택된 파일이 GtkFileChooser 내에서 변경되었다. 마우스 또는 키보드가 선택내용을 변경하거나 코드가 선택내용을 명시적으로 변경하는 경우를 예로 들 수 있겠다.
update-preview	사용자가 어떤 액션을 실행하였으므로 파일 선택자에서 미리보기 위젯이 재생성되어야 한다. 파일 선택자에 미리보기 위젯이 있을 경우 이 시그널을 사용할 필요가 있다.

표 B-24. GtkFileChooser 시그널

시그널명	설명
font-set	사용자가 새로운 글꼴을 선택하였다. 글꼴을 명시적으로 변경할 때는 이 시그널이 발생하지 않는다. 글꼴의 변경내용을 모두 감시하기 위해서는 GtnFontButton의 font-name 프로퍼티에서 notify 시그널을 사용할 필요가 있다.

표 B-25. GtkFontButton 시그널

시그널명	추가 매개변수	설명
child-attached	GtkWidget *child	핸들 박스가 메인 창에 다시 붙었다.
child-detached	GtkWidget *child	핸들 박스가 메인 창에서 다시 분리되었다.

표 B-26. GtkHandleBox 시그널

시그널명	추가 매개변수	설명
activate-cursor-item	None	아이콘이 추가될 때 사용자가 Enter 키를 눌렀다.
item-activated	GtkTreePath *path	사용자가 아이콘 항목을 더블 클릭하였거나 Enter 키를 눌렀다. gtk_icon_view_item_activated()를 호출하면 이 시그널을 강제로 발생시킬 수 있다.
move-cursor	GtkMovementStep step gint num_steps	사용자가 마우스 커서와 다른 아이콘을 선택하였다. Up, Down, Ctrl+P, Ctrl+N, Home, End, Page Up, Page Down, Right, Left 키 또는 몇 개의 Shift 및 Ctrl 키의 조합으로 이루어진다.
select-all	None	키보드에 Ctrl+A를 눌러서 모든 항목이 선택되었다.
select-cursor-item	None	사용자가 키보드에 스페이스 바를 눌러 아이콘 항목을 선택하였다.
selection-changed	None	선택된 아이콘이 사용자 액션 또는 애플리케이션에 의한 호출로 인해 변경되었다.
set-scroll-adjustments	GtkAdjustment *hadj GtkAdjustment *vadj	GtkIconView의 스크롤 조정이 변경되었다.
toggle-cursor-item	None	사용자가 키보드에 Ctrl+스페이스 바를 눌렀다.

unselect-all	None	사용자가 키보드에 Ctrl+Shift+A를 눌러 모든 아이콘 항목이 선택 해제되었다.
표 B-27. GtkIconView 시그널		

시그널명	추가 매개변수	설명
commit	gchar *str	애플리케이션이 문자열을 표시할 준비가 되었다.
delete-surrounding	gint offset gint delete_chars	입력 방식이 컨텍스트 텍스트를 삭제할 필요가 있다. 시그널이 처리되었다면 TRUE를 리턴되어야 한다.
preedit-changed	None	사전에 편집된(preeditd) 텍스트가 변경되었다.
preedit-end	None	사전에 편집된 텍스트 변경이 완료되었다.
preedit-start	None	사전에 편집된 텍스트 변경이 시작되었다.
retrieve-surrounding	None	입력 방식이 커서를 둘러싼 컨텍스트에 대해 알 필요가 있다. gtk_im_context_set_surrounding()을 이용해 주변 컨텍스트를 설정 시 이 시그널을 이용하고, 성공적으로 처리되면 TRUE를 리턴한다.
표 B-28. GtkIMContext 시그널		

시그널명	추가 매개변수	설명
disable-device	GdkDevice *deviceid	사용자가 입력 장치 모드를 GDK_MODE_SCREEN 또는 GDK_MODE_WINDOW에서 GDK_MODE_DISABLED로 변경하였다.
enable-device	GdkDevice *deviceid	사용자가 입력 장치 모드를 GDK_MODE_DISABLED에서 GDK_MODE_SCREEN 또는 GDK_MODE_WINDOW로 변경하였다.
표 B-29. GtkInputDialog 시그널		

시그널명	설명
deselect	사용자가 GtkItem 위젯을 선택해제하였거나, gtk_item_deselect()가 호출되었다.
select	사용자가 GtkItem 위젯을 선택하였거나, gtk_item_select()가 호출되었다.
toggle	사용자가 GtkItem 위젯을 토글하였거나, or gtk_item_toggle()가 호출되었다.
표 B-30. GtkItem 시그널	

시그널명	추가 매개변수	설명
copy-clipboard	None	텍스트가 GtkLabel 위젯으로부터 GtkClipboard로 복사되었다. GtkLabel 위젯을 선택 가능하게 만들면 라벨의 전체를 비롯해 일부의 복사할 수 있다.
move-cursor	GtkMovementStep step gint num_steps gboolean extended	GtkLabel을 마우스로 선택 가능하도록 허용했다면 커서가 표시될 것이다. 이후 라벨 주위에 커서를 이동시키면 이 시그널이 발생할 것이다. 콜백 함수는 선택 범위가 확장되었는지 여부를 수신한다.
populate-popup	GtkMenu *popup_menu	사용자가 GtkLabel 위젯을 오른쪽 마우스로 클릭하였으며, 개발자는 새로운 메뉴를 채울(populate) 필요가 있다.
표 B-31. GtkLabel 시그널		

시그널명	추가 매개변수	설명
set-scroll-adjustments	GtkAdjustment *hadj GtkAdjustment *vadj	레이아웃의 스크롤 조정이 변경되었다.
표 B-32. GtkLayout 시그널		

시그널명	추가 매개변수	설명
move-scroll	GtkScrollType type	사용자가 GtkScrollType 값들 중 하나를 이용해 메뉴를 스크롤하였다.
표 B-33. GtkMenu 시그널		

시그널명	추가 매개변수	설명
activate	None	메뉴 항목이 활성화되었다. 하위메뉴의 활성화를 포착해야 한다면 activate-item 시그널을 사용해야 한다.
activate-item	None	메뉴 항목이 활성화되었거나, 메뉴 항목에 활성화된 하위메뉴가 있다.
toggle-size-allocate	gint new_size	메뉴 항목에 새로운 크기가 할당되었다.
toggle-size-request	gpointer size	메뉴 항목이 새로운 크기를 요청하였다.
표 B-34. GtkMenuItem 시그널		

시그널명	추가 매개변수	설명
activate-current	gboolean force_hide	GtkMenuShell이 포함하는 현재 메뉴 항목을 활성화하라.
cancel	None	선택된 메뉴 항목의 선택부분을 취소하라. 이는 selection-done 시그널을 발생시킬 것이다.
cycle-focus	GtkDirectionType type	포커스가 주어진 방향에 있는 다른 메뉴 바로 이동하였다.
deactivate	None	GtkMenuShell 이 활성화되었고, 이는 주로 화면에서 제거되었음을 의미한다.
move-current	GtkMenuDirectionType type	메뉴 셀에서 현재 메뉴 항목이 주어진 방향으로 이동하였다.
selection-done	None	메뉴 셀 내에서 선택내용의 사용이 끝났다.
표 B-35. GtkMenuShell 시그널		

시그널명	설명
show-menu	이 시그널은 메뉴가 표시되기 직전에 발생하여, 사용자가 메뉴를 확인하기 전에 개발자가 업데이트할 수 있는 기회를 제공한다.
표 B-36. GtkMenuToolButton 시그널	

시그널명	추가 매개변수	설명
change-current-page	gint pages_moved	현재 GtkNotebook이 표시하는 페이지가 변경되었다.
focus-tab	GtkNotebookTab type	현재 탭을 변경하여 포커스가 이동하였다. 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
move-focus-out	GtkDirectionType type	포커스가 주어진 방향으로 GtkNotebook 위젯 밖으로 이동하였다.
page-added	GtkWidget *child guint page_num	GtkNotebook 위젯으로 페이지가 추가되었다. 이 시그널은 GTK+ 2.10 버전에 추가되었다.
page-removed	GtkWidget *child guint page_num	GtkNotebook 위젯에서 페이지가 제거되었다. 이 시그널은 GTK+ 2.10 버전에 추가되었다.
page-reordered	GtkWidget *child guint page_num	GtkNotebook 위젯 페이지가 재정렬되었다. 이 시그널은 GTK+ 2.10 버전에 추가되었다.
select-page	gboolean focus_moved	자식 위젯에 새 페이지가 선택되었다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
switch-page	GtkNotebookPage *page guint page_num	노트북 페이지가 주어진 페이지로 변경되었다.

표 B-37. GtkNotebook 시그널

**시그널 설명**

destroy	GtkWidget 위젯이 그 참조를 모두 해제하면 위젯은 소멸될 것이다. 따라서 개발자가 모든 참조를 해제하면 객체가 최종화될 것이다.
---------	---

표 B-38. GtkWidget 시그널

**시그널명 추가 매개 변수 설명**

accept-position	None	페이지의 크기 조정이 완료되었고, 사용자가 Return 키, Enter 키, 스페이스 바 중 하나를 눌렀다. 이 시그널은 자식 위젯에게 포커스를 주어 활성화할 것이다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
cancel-position	None	사용자가 내용 변경을 취소하기 위해 Escape 키를 눌러 페이지의 크기 조정이 중단되었다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
cycle-child-focus	gboolean reversed	GtkPaned 위젯에 포커스가 있을 때 사용자가 F6 또는 Shift+F6을 눌러 자식의 포커스를 변경하였다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
cycle-handle-focus	gboolean reversed	GtkPaned 위젯에 포커스가 있을 때 사용자가 Tab, Ctrl+Tab, Shift+Tab, Ctrl+Shift+Tab 중 하나를 누르면 시그널이 발생한다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
move-handle	GtkScrollType type	핸들이 제거되고 Left, Right, Up, Down, Page Up, Page Down, Home, End 키 중 하나에 포커스가 있을 때 해당 키가 눌러졌다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
toggle-handle-focus	None	GtkPaned 위젯이 포커스 범위 내에 있었고, F8을 눌러 핸들로 포커스를 주거나 포커스를 없앴다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.

표 B-39. GtkPaned 시그널

시그널명 설명	
embedded	GtkPlug 창이 부모로 소켓 창에 할당되었다. GtkPlug는 최상위 수준의 위젯이 다른 프로세스로 포함되는 것을 허용한다.
표 B-40. GtkPlug 시그널	

시그널명	추가 매개변수	설명
details-acquired	gboolean success	프린트 백엔드로부터 프린터에 대한 상세한 정보가 요청되었다.
표 B-41. GtkPrinter 시그널		

시그널명	설명
status-changed	인쇄 작업의 현재 상태가 변경되었다. 인쇄 작업의 새로운 상태를 확인하기 위해서는 gtk_print_job_get_status()를 사용해야 한다.
표 B-42. GtkPrintJob 시그널	

시그널명	추가 매개변수	설명
begin-print	GtkPrintContext *context	사용자가 프린터 설정 변경을 마쳤으나 렌더링이 아직 시작되지 않았다.
create-custom-widget	None	대화상자가 방금 표시되었다. 콜백 함수로부터 다수의 위젯을 포함하는 위젯 또는 컨테이너 위젯을 리턴하여 대화상자의 GtkNotebook에 커스텀 페이지로 추가할 수 있다.
custom-widget-apply	GtkWidget *widget	커스텀 위젯이 create-custom-widget 시그널 핸들러에 추가된 경우 begin-print가 발생하기 직전에 이 시그널이 발생된다.
done	GtkPrintOperationResult result	인쇄가 완료되었고, 개발자는 결과를 볼 수 있다. 결과가 GTK_PRINT_OPERATION_RESULT_ERROR일 경우 gtk_print_operation_get_error()를 이용해 오류를 확인할 수 있다.
draw-page	GtkPrintContext *context gint page_num	각 페이지가 Cairo 컨텍스트로 변환되어야 한다. 페이지를 수동으로 렌더링 시 이 콜백 함수를 이용해야 한다.
end-print	GtkPrintContext *context	모든 페이지가 렌더링되었다.
paginate	GtkPrintContext *context	페이지 렌더링이 시작되기 전에 begin-print 다음으로 이 시그널이 발생한다. 이 시그널은 FALSE가 리턴될 때까지 혹은 시그널이 처리되지 않는 한 계속 발생한다. 이는 문서를 단계별로 페이지로 나누도록 허용하여 사용자 인터페이스가 현저히 블로킹(blocked)되지 않도록 한다.
preview	GtkPrintOperationPreview *preview GtkPrintContext *context GtkWindow *parent	사용자가 메인 인쇄 대화상자로부터 문서의 미리보기를 요청하였다. 이 시그널은 자신만의 미리보기 대화상자를 생성하도록 해준다. 이 시그널이 처리되지 않으면 기본 핸들러가 사용될 것이다. 인쇄 미리보기를 처리 중이라면 콜백 함수는 TRUE를 리턴한다.
request-page-setup	GtkPrintContext *context gint page_num GtkPageSetup *setup	이 시그널은 페이지마다 발생하여 페이지가 인쇄되기 전에 셋업을 마지막으로 편집할 수 있는 기회를 제공한다. 어떤 변경사항이든 현재 페이지에만 적용될 것이다!
status-changed	None	인쇄 연산의 상태가 변경되었다. 가능한 값은 GtkPrintStatus 열거에서 정의되며, 현재 값은 gtk_print_operation_get_status()를 이용해 검색할 수 있다.
표 B-43. GtkPrintOperation 시그널		

시그널명	추가 매개변수	설명
changed	GtkRadioAction *current	그룹 내 두 개의 라디오 버튼의 상태가 변경되었다. 이 시그널은 선택내용이 변경될 때 라디오 버튼의 개수만큼 발생한다.

표 B-44. GtkRadioAction 시그널

시그널명	설명
group-changed	라디오 버튼이 새로운 그룹으로 전환되었거나, 라디오 버튼으로부터 제거되었다.

표 B-45. GtkRadioButton and GtkRadioMenuItem 시그널

시그널명	추가 매개변수	설명
adjust-bounds	gdouble value	GtkRange의 경계(bounds)가 특정 타입의 사용자 액션에 의해 수정되었다.
change-value	GtkScrollType type gdouble value	범위(range)의 현재 값이 변경되었다. TRUE를 리턴하면 범위가 업데이트되는 것을 막을 수 있지만 표시된 값을 바람직한 소수 자리수로 직접 올림/내림해야 한다.
move-slider	GtkScrollType type	사용자가 키보드 키, 즉 Page Up, Page Down, Home, End, 또는 슬라이더를 이동시키는 화살표 키 중에서 하나를 눌렀다.
value-changed	None	범위 값이 변경되었다. 이는 사용자 액션이나 코드로부터의 호출로 인해 야기된다.

표 B-46. GtkRange 시그널

시그널명	추가 매개변수	설명
format-value	gdouble value	스케일이 표시될 예정이지만 GTK+는 먼저 스케일을 표시하는 방법을 맞춤설정할 수 있는 기회를 제공한다. 콜백 함수는 개발자가 생성한 값을 표시하는 맞춤설정된 문자열을 리턴한다.

표 B-47. GtkScale 시그널

시그널명	추가 매개변수	설명
move-focus-out	GtkDirectionType type	사용자가 Ctrl+Tab 또는 Ctrl+Shift+Tab을 눌러 스크롤이 있는 창에서 포커스를 이동하였다. 주어진 방향은 언제나 Ctrl+Tab 와 Ctrl+Shift+Tab 중 하나에 해당한다.
scroll-child	GtkScrollType type gboolean horizontal	자식 위젯이 한 방향으로 스크롤되었다. 이는 마우스 또는 다음의 기본 키 바인딩으로 야기된다. Ctrl+Left, Ctrl+Right, Ctrl+Up, Ctrl+Down, Ctrl+Page Up, Ctrl+Page Down, Page Up, Page Down, Ctrl +Home, Ctrl+End, Home, End.

표 B-48. GtkScrolledWindow 시그널

시그널명 설명	
plug-added	클라이언트가 성공적으로 소켓에 추가되었다.
plug-removed	클라이언트가 소켓에서 제거되었다. 보통은 기본 위젯인 GtkSocket 위젯을 소멸하길 원할 것이다. 이를 막기 위해서는 콜백 함수로부터 TRUE를 리턴하면 된다.
표 B-49. GtkSocket 시그널	

시그널명	추가 매개 변수	설명
change-value	GtkScrollType type	표시된 스펀 버튼의 값이 변경되었다. 이는 다음 키보드 바인딩 중 하나를 누르면 실행된다. Up, Down, Page Up, Page Down, Ctrl+Page Up, Ctrl+Page Down.
input	gpointer value	표시된 값이 변경되었다.
output	None	실현된 위젯의 digits 프로퍼티를 변경하거나 새로운 값을 설정하여 표시된 스펀 버튼의 값이 변경되었다. 시그널을 성공적으로 처리했다면 추가 액션을 취하지 않도록 TRUE를 리턴해야 한다.
value-changed	None	스핀 버튼 값의 변경을 요하는 스펀 버튼의 프로퍼티들 중 하나가 (value 또는 digits) 변경되었다.
wrapped	None	스핀 버튼이 최대값에서 최소값으로, 또는 그 반대로 래핑되었다. 이 시그널은 GTK+ 2.10 버전에서 소개되었다.
표 B-50. GtkSpinButton 시그널		

시그널명	추가 매개 변수	설명
text-popped	guint context_id gchar *message	상태 표시줄의 스택에서 맨 위의 메시지가 제거되었다. 그 다음 메시지가 표시될 것이다.
text-pushed	guint context_id gchar *message	상태 표시줄의 스택에서 맨 위에 메시지가 추가되었다.
표 B-51. GtkStatusbar 시그널		

시그널명	추가 매개 변수	설명
activate	None	상태 아이콘이 활성화되었다. 상태 아이콘이 어떻게 활성화되는지는 사용자의 플랫폼에 따라 결정된다. 어떤 경우든 적절하게 조치를 취해야 한다.
popup-menu	guint button guint activate_time	상태 아이콘의 팝업 메뉴가 표시되었다. 함수 매개 변수를 gtk_menu_popup()으로 전달할 수 있다. 팝업 메뉴 기능이 모든 플랫폼에서 이용할 수 있는 것은 아니므로 대안적인 기능을 항상 제공해야 한다!
size-changed	gint size	상태 아이콘에 이용 가능한 영역이 변경되었다. 아이콘 크기를 스케일링한 경우 콜백 함수는 TRUE를 리턴한다. 그렇지 않으면 GTK+가 알아서 스케일링을 처리할 것이다.
표 B-52. GtkStatusIcon 시그널		

시그널명	추가 매개변수	설명
apply-tag	GtkTextTag *tag GtkTextIter *start GtkTextIter *end	GtkTextTag 위젯이 텍스트 버퍼의 섹션으로 적용되었다.
begin-user-action	None	사용자가 텍스트 버퍼에서 특정한 타입의 액션을 시작하였다.
changed	None	텍스트 버퍼가 어떤 방식으로 변경되어 시각적 또는 비시각적 텍스트, 이미지 혹은 위젯이 변경되는 결과를 야기하였다.
delete-range	GtkTextIter *start GtkTextIter *end	텍스트 버퍼로부터 텍스트가 삭제되었다.
end-user-action	None	텍스트 버퍼에서 일부 타입의 사용자 액션이 종료되었다.
insert-child-anchor	GtkTextIter *location GtkTextChildAnchor *anchor	앵커(anchor)가 삽입되어 텍스트 버퍼가 다른 위젯을 포함하도록 허용한다.
insert-pixbuf	GtkTextIter *location GdkPixbuf *pixbuf	GdkPixbuf 위젯이 텍스트 버퍼로 삽입되었다.
insert-text	GtkTextIter *location gchar *new_text gint text_length	텍스트가 텍스트 버퍼로 삽입되었다.
mark-deleted	GtkTextMark *mark	텍스트 버퍼로부터 GtkTextMark 객체가 삭제되었다.
mark-set	GtkTextIter *location GtkTextMark *mark	GtkTextMark 객체가 텍스트 버퍼로 추가되었다.
modified-changed	None	텍스트 버퍼가 modified 또는 unmodified 상태로 설정되었다.
remove-tag	GtkTextTag *tag GtkTextIter *start GtkTextIter *end	텍스트 버퍼로부터 주어진 반복자(iterator)들 사이에서 태그가 제거되었다.

표 B-53. GtkTextBuffer 시그널

시그널명	추가 매개변수	설명
event	GObject *object GdkEvent *event GtkTextIter *location	GtkTextTag가 포함하는 텍스트의 범위 내에서 이벤트가 발생하였다.

표 B-54. GtkTextTag 시그널

시그널명	추가 매개변수	설명
tag-added	GtkTextTag *tag	GtkTextTag 객체가 태그 테이블로 추가되었다.
tag-changed	GtkTextTag *tag gboolean size_changed	태그 테이블이 포함하는 태그의 프로퍼티가 변경되었다. 표시된 텍스트의 크기는 크기 외에 굵기나 폰트 패밀리와 같은 프로퍼티를 변경하여 변경이 가능하다.
tag-removed	GtkTextTag *tag	태그 테이블로부터 GtkTextTag 객체가 제거되었다.

표 B-55. GtkTextTagTable 시그널



시그널명	추가 매개변수	설명
backspace	None	커서 뒤의 문자 하나가 문서에서 삭제되었다.
copy-clipboard	None	선택된 텍스트가 클립보드로 복사되었다.
cut-clipboard	None	선택된 텍스트가 클립보드로 복사되고 문서에서 제거되었다.
delete-from-cursor	GtkDeleteType type gint length	커서 주위에서 텍스트가 삭제되었다.
insert-at-cursor	gchar *text	현재 커서 위치에 텍스트가 삽입되었다.
move-cursor	GtkMovementStep step gint num_steps gboolean extended	커서가 새로운 위치로 이동하였으며, 현재 선택내용을 확장할 수도 있다.
move-focus	GtkDirectionType type	포커스가 주어진 방향으로 이동하였다.
move-viewport	GtkScrollStep step gint num_steps	특정 타입의 스크롤링이 발생하였으며, 주어진 단계에서 설명된다.
paste-clipboard	None	클립보드의 텍스트가 문서로 삽입되었다.
populate-popup	GtkMenu *menu	팝업 메뉴가 표시되어 편집에 이용할 수 있다.
select-all	gboolean selected	문서 내 모든 텍스트가 선택 또는 선택 해제되었다.
set-anchor	None	앵커가 텍스트 뷰로 추가되었다.
set-scroll-adjustments	GtkAdjustment *hadj GtkAdjustment *vadj	텍스트 뷰의 조정이 설정되었다.
toggle-overwrite	None	덮어쓰는 키가 토글되어 켜지거나 꺼졌다.

표 B-56. GtkTextView 시그널

시그널명 설명	
toggled	토글의 상태가 변경되었다. 토글이 활성화되거나 비활성화되었을 때 어떤 액션을 취하고 싶다면 이 시그널을 연결해야 한다.

표 B-57. GtkToggleAction, GtkToggleButton, and GtkToggleToolButton 시그널

시그널명	추가 매개변수	설명
focus-home-or-end	gboolean focus_home	이 시그널은 툴바에서 첫 번째 요소 또는 마지막 요소로 이동하기 위해 GTK+가 내부적으로 사용하는 시그널로, 애플리케이션 코드에선 사용할 수 없다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
move-focus	GtkDirectionType type	이 시그널은 포커스가 있는 항목을 이동하기 위해 GTK+가 내부적으로 사용하는 시그널로, 애플리케이션 코드에선 사용할 수 없다.
orientation-changed	GtkOrientation dir	툴바의 방향이 가로 또는 세로로 변경되었다.
popup-context-menu	gint x_position gint y_position gint button	사용자가 툴바를 오른쪽 마우스로 클릭하였거나 키 바인딩을 눌러 팝업 메뉴를 표시하였다. 이 시그널은 툴바에 대한 커스텀 컨텍스트 메뉴를 표시할 때 사용할 수 있다. 시그널이 처리되면 TRUE를 리턴해야 한다.
style-changed	GtkToolbarStyle style	툴바의 스타일이 변경되었다.

표 B-58. GtkToolbar 시그널

시그널명 설명	
changed	마우스로 툴 버튼이 클릭되었다. 툴 버튼이 키보드 바인딩을 이용해 활성화되었을 때도 이 시그널을 발생시킬 수 있다.
표 B-59. GtkToolButton 시그널	

시그널명	추가 매개변수	설명
create-menu-proxy	None	툴바는 항목이 오버플로 메뉴에 표시되어야 하는지 여부를 알 필요가 있다. 이 시그널을 처리하기 위해서는 <code>gtk_tool_item_set_proxy_menu_item()</code> 을 호출하거나 FALSE를 리턴해야만 오버플로 메뉴에 나타나는 것을 방지할 수 있다. 시그널이 처리되면 TRUE를 리턴해야 한다.
set-tool-tip	GtkTooltips *tooltips gchar *tip_text gchar *tip_private	항목의 툴팁이 주어진 설정으로 변경되었다.
toolbar-reconfigured	None	툴 항목의 부모에서 일부 프로퍼티가 변경되어 자식이 변경될 필요가 있다. 이는 툴바의 방향, 스타일, 아이콘 크기, 또는 양감 스타일이 변경하였을 때 발생한다.
표 B-60. GtkToolItem 시그널		

시그널명	추가 매개변수	설명
row-changed	GtkTreePath *path GtkTreeIter *iter	트리 모델 내 주어진 위치의 행이 변경되었다.
row-deleted	GtkTreePath *path	트리 모델 내 주어진 위치에서 행이 제거되었다.
row-has-child-toggled	GtkTreePath *path GtkTreeIter *iter	주어진 위치의 행에 첫 번째 자식이 주어졌거나, 마지막으로 남은 자식이 제거되었다.
row-inserted	GtkTreePath *path GtkTreeIter *iter	트리 모델에 행이 추가되었다. 이 시그널은 행이 추가된 직후에 호출되므로 아직 데이터를 포함하지 않을 수도 있다.
rows-reordered	GtkTreePath *path GtkTreeIter *iter gpointer row_nums	트리 모델 내의 행들이 드래그 앤 드롭 외의 특정 방법에 따라 재정렬되었다. 콜백 함수는 재정렬된 행 번호의 배열을 수신한다.
표 B-61. GtkTreeModel 시그널		

시그널명	설명
changed	선택내용이 변경되었을 가능성이 있다. 이 시그널은 항상 신뢰성이 있는 것은 아닌데, Shift 키를 이용해 다중 행을 선택 시 한 번만 발생하기도 하고, 아무 일도 일어나지 않았을 때에도 시그널이 발생할 수 있기 때문이다. 이 시그널에 대해서는 콜백 함수에 오류 보호(error protection)를 만들어야 한다.
표 B-62. GtkTreeSelection 시그널	

시그널명	설명
sort-column-changed	정렬 열(sort column)은 GtkTreeSortable 모델 내에서 모든 행을 정렬하는 데에 사용할 열이다. 이 시그널은 선택된 정렬 열이 변경되면 발생한다.

표 B-63. GtkTreeSortable 시그널

시그널명	추가 매개변수	설명
columns-changed	None	열이 트리 뷰에 추가되거나 그로부터 제거되어 열의 개수가 변경되었다.
cursor-changed	None	셀 내에서 포커스가 있는 커서의 위치가 변경되었다.
expand-collapse-cursor-row	gboolean logical gboolean expanded expand_children	커서 위치에 있는 행이 확장 또는 축소되어야 한다. 이 시그널이 처리되면 TRUE를 리턴해야 한다.
move-cursor	GtkMovementStep step gint num_steps	다음 키 바인딩 중 하나를 이용해 커서가 이동되었다. Right, Left, Up, Down, Page Up, Page Down, Home, End. 시그널이 처리되면 TRUE를 리턴해야 한다.
row-activated	GtkTreePath *path GtkTreeViewColumn *column	사용자가 행을 더블 클릭하였거나 gtk_tree_view_row_activated()가 호출되었다. 다음 키 바인딩을 이용해도 시그널이 발생할 수 있다. Space, Shift+space bar, Return, Enter.
row-collapsed	GtkTreeIter *iter GtkTreePath *path	주어진 행의 자식 노드들이 숨겨졌다.
row-expanded	GtkTreeIter *iter GtkTreePath *path	주어진 행의 자식 노드들이 표시되었다.
select-all	None	트리 뷰 내의 모든 행이 선택되었다. Ctrl+A 또는 Ctrl+/를 누르면 실행된다.
select-cursor-parent	None	행에 포커스가 있을 때 사용자가 Backspace 키를 눌렀다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
select-cursor-row	gboolean editing	다음 키 바인딩 중 하나를 눌러 편집 불가능(noneditable) 행이 선택되었다. space bar, Shift+space bar, Return, Enter. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
set-scroll-adjustments	GtkAdjustment *hadj GtkAdjustment *vadj	트리 뷰에 대해 수평 및 수직 스크롤 조정이 설정되었다. 이 시그널이 처리되면 콜백 함수는 TRUE를 리턴한다.
start-interactive-search	None	트리 뷰에 포커스가 있을 때 사용자가 Ctrl+F를 눌렀다. 시그널이 처리되면 TRUE를 리턴해야 한다.
test-collapse-row	GtkTreeIter *iter GtkTreePath *path	행이 축소되기 직전이다. 축소를 하기 위해서는 콜백 함수가 TRUE를 리턴해야 한다.
test-expand-row	GtkTreeIter *iter GtkTreePath *path	행이 확장되기 직전이다. 확장을 하기 위해서는 콜백 함수가 TRUE를 리턴해야 한다.
toggle-cursor-row	None	The user pressed Ctrl+space bar while a row had focus. 시그널이 처리되면 TRUE를 리턴해야 한다.
unselect-all	None	Shift+Ctrl+A 또는 Shift+Ctrl+/를 눌러 트리 뷰의 모든 행이 선택 해제되었다. 시그널이 처리되면 TRUE를 리턴해야 한다.

표 B-64. GtkTreeView 시그널

시그널명	설명
clicked	사용자가 트리 뷰 열의 헤더 버튼을 눌렀다. 이는 주로 정렬을 지원하는 뷰에서 해당 열에 따라 트리 뷰의 행을 정렬시키는 결과를 야기한다.

표 B-65. GtkTreeViewColumn 시그널

시그널명	추가 매개변수	설명
actions-changed	None	UI 관리자 내의 액션 집합이 변경되었다.
add-widget	GtkWidget *widget	메뉴 표시줄 또는 툴바가 생성되었다. 팝업 메뉴에 대해서는 발생하지 않는 시그널이므로 팝업 메뉴를 검색하려면 gtk_ui_manager_get_widget()을 이용해야 할 것이다.
connect-proxy	GtkAction *action GtkWidget *proxy	그룹 내 액션으로 프록시가 연결되었다. 다수의 액션에서 사용하는 맞춤설정에 이 시그널을 이용할 수 있다.
disconnect-proxy	GtkAction *action GtkWidget *proxy	그룹 내 액션으로 프록시가 연결되었다.
post-activate	GtkAction *action	액션이 방금 활성화되었다. 모든 액션의 활성화에 대한 설명(notice)을 검색 시 이 시그널을 이용할 수 있다.
pre-activate	GtkAction *action	액션이 활성화되기 직전이다. 모든 액션의 활성화에 대한 설명을 검색 시 이 시그널을 이용할 수 있다.

표 B-66. GtkUIManager 시그널

시그널명	추가 매개변수	설명
set-scroll-adjustments	GtkAdjustment *hadj GtkAdjustment *vadj	뷰포트에 대한 조정이 변경되었다.

표 B-67. GtkViewport 시그널

시그널명	추가 매개변수	설명
accel-closures-changed	None	가속기가 위젯의 가속기 그룹으로 추가되었거나 그로부터 제거되었다. 이 시그널은 가속기 경로가 설정되었을 때도 발생한다.
can-activate-accel	guint signal_id	이 시그널을 이용해 가속기의 활성화 가능 여부에 따라 기본 핸들러를 오버라이드 할 수 있다. 시그널의 활성화가 가능하다면 TRUE를 리턴해야 한다.
child-notify	GParamSpec *pspec	위젯에 대한 자식 프로퍼티가 변경되었다. 시그널 프로퍼티를 감시할 때 이 시그널을 이용할 수 있다.
composited-changed	None	위젯의 composited 상태가 변경되었는데, composited란 위젯의 알파 채널이 유효하게 간주될 것인지(honored) 여부를 결정하는 프로퍼티에 해당한다.
direction-changed	GtkTextDirection dir	위젯 내에서 텍스트의 방향이 변경되었다. 주로 gtk_widget_set_direction()의 호출로 시작된다.
drag-begin	GdkDragContext *context	드래그 액션이 시작되었다. 이 시그널은 드래그 소스에서 발생한다. 이 시그널을 이용해 드래그 시 표시할 커스텀 아이콘을 설정할 수 있다.
drag-data-delete	GdkDragContext *context	드래그 액션이 성공적으로 완료되었다. 이 시그널은 액션이 완료된 후 드래그하던 데이터를 삭제할 때 사용된다.
drag-data-get	GdkDragContext *context GtkSelectionData *data guint info guint timestamp	드롭 사이트(drop site)가 드래그된 데이터를 요청하였다.

drag-data-received	GdkDragContext *context gint x_position gint y_position GtkSelectionData *data guint int guint timestamp	드롭 사이트가 드래그 데이터를 수신하였다.
drag-drop	GdkDragContext *context gint x_position gint y_position guint timestamp	사용자가 데이터를 위젯에 드롭하였다. 개발자는 커서의 위치가 허용되는 드롭 영역 내에 있는지 결정해야 한다. 드롭이 허용되면 TRUE를 리턴해야 한다.
drag-end	GdkDragContext *context	드래그 액션이 완료되었으며, 이를 이용해 drag-begin 콜백에서 실행된 액션을 실행 취소할 수 있다.
drag-leave	GdkDragContext *context guint timestamp	커서가 드롭 사이트 주위 영역을 벗어났다. 이 시그널을 이용해 drag-motion 콜백에서 실행된 액션을 실행 취소할 수 있다.
drag-motion	GdkDragContext *context gint x_position gint y_position guint timestamp	드래그 도중에 커서가 드롭 사이트 위로 이동하였다. 커서가 허용되는 드롭 영역 내에 있을 경우 TRUE를 리턴해야 한다.
focus	GtkDirectionType type	위젯이 포커스를 받았다. 시그널이 처리되면 TRUE를 리턴해야 한다.
grab-focus	None	gtk_widget_grab_focus()를 호출함으로써 위젯이 강제로 포커스를 받았다. 니모닉 가속기를 이용해 시그널을 시작할 수도 있다.
grab-notify	gboolean was_grabbed	다른 위젯에서 gtk_grab_add()를 명시적으로 호출하여 위젯에 그림자가 생겼거나 (shadowed), 제거된 그림으로 인해 그림자가 해제되었다.
hide	None	위젯이 사용자의 뷰에서 숨겨졌다. 누락된 위젯을 포함하도록 사용자 인터페이스가 다시 그려질 것이다.
hierarchy-changed	GtkWidget *oplevel	최상위 수준의 조상이 GtkWidget 위젯일 때 위젯은 앵커(anchored)된 것으로 간주된다. 이 시그널은 자식 위젯이 앵커 또는 앵커해제 될 때 발생할 수 있다.
map	None	위젯이 매핑을 요청하였다. 이 시그널은 gtk_widget_show() 또는 gtk_widget_map()을 호출하여 시작할 수 있다.
mnemonic-activate	gboolean shift_focus	위젯을 활성화하기 위해 니모닉 가속기가 사용되었다.
parent-set	GtkObject *old_parent	부모 위젯이 변경되었다.
popup-menu	None	사용자가 팝업 메뉴를 표시할 것을 요청하였다. 처리되고 나면 이 콜백 함수는 TRUE를 리턴한다.
realize	None	gtk_widget_realize()의 호출로 인해 위젯이 실현되도록 요청하였다. 자신만의 커스텀 위젯을 생성하지 않는 한 보통 코드에서 명시적으로 호출하지는 않는다.
screen-changed	GdkScreen *screen	위젯이 새로운 화면으로 이동하였다.
selection-get	GtkSelectionData *data guint info guint timestamp	위젯으로부터 선택내용 데이터가 요청되었다.
selection-received	GtkSelectionData *data guint timestamp	선택내용의 소유주가 위젯의 선택내용 데이터의 요청에 응답하였다.
show	None	위젯이 표시되도록(visible) 설정되었다. 사용자 인터페이스는 새로 표시되는 위젯을 포함하도록 다시 그려질 것이다.
show-help	GtkWidgetHelpType type	사용자가 Ctrl+F1을 눌러 위젯과 관련해 도움을 요청하였다. 도움의 유형은 GTK_WIDGET_HELP_TOOLTIP과 GTK_WIDGET_HELP_WHATS_THIS로 구성된 GtkWidgetHelpType에서 정의된다.
size-allocate	GtkAllocation *alloc	위젯에 새로운 크기 할당이 부여되었다.
size-request	GtkRequisition *req	위젯이 gtk_widget_set_size_request()를 이용해 새로운 크기를 요청하였다.
state-changed	GtkStateType state	위젯의 현재 크기가 주어진 상태로 변경되었다.

style-set	GtkStyle *prev_style	위젯의 스타일이 수정되었다. 이는 전체 스타일을 변경하거나 스타일의 특정 요소를 변경하여 야기된다.
unmap	None	위젯이 매핑해제를 요청하였다. 이는 gtk_widget_unmap()을 호출하여 시작할 수 있다.
unrealize	None	위젯이 실현 취소(unrealized)를 요청하였다. 이는 모든 자식 위젯의 자원과 연관된 자원을 모두 해제하는 결과를 야기할 것이다.

표 B-68. 이벤트가 제거된 GtkWidget 시그널

시그널명	추가 매개변수	설명
activate-default	None	창의 기본 위젯이 활성화되었다. 보통 사용자가 Return 키 또는 Enter 키를 눌러서 발생한다.
activate-focus	None	포커스가 있는 창 의 자식 위젯이 활성화되었다. 보통 사용자가 스페이스 바를 눌러서 발생한다.
frame-event	GdkEvent *event	창의 프레임에서 포커스의 변경 또는 key-press-event, key-release-event 이외의 이벤트가 수신되었다.
keys-changed	None	니모닉 가속기가 창 내에서 추가, 제거 또는 변경되었다. 니모닉 수정자를 설정 시 발생하기도 한다.
move-focus	GtkDirectionType type	창의 자식 위젯 내에서 포커스가 변경되었다. 주로 사용자가 다음 키 바인딩 중 하나를 누르면 발생한다. Tab, Shift+Tab, Up, Down, Left, Right.
set-focus	GtkWidget *widget	포커스가 창의 다른 자식으로 변경되었다.

표 B-69. GtkWindow 시그널

## Notes

# FoundationsofGTKDevelopment:Appendix C

## 부록 C GTK+ 스타일

### GTK+ 스타일

GTK+는 위젯의 스타일을 맞춤설정할 수 있도록 많은 방도를 제공한다. 위젯 스타일의 맞춤설정 대부분은 스타일 프로퍼티와 자원(RC) 파일을 통해 이루어지는데, 이는 본 서적의 제 4장 "위젯 스타일" 절에서 다룬 바 있다.

제 4장에 실린 정보를 비롯해 이번 부록은 모든 위젯, Pango 텍스트 마크업 언어, GtkTextTag 스타일에 적용할 수 있는 기본 RC 파일 요소에 대한 참고자료를 제공하고자 한다.

### 기본 RC 파일 스타일

자원 파일은 제 4장에서 소개했지만 이번 절에 실린 내용을 모든 위젯이 지원하는 기본 스타일에 대한 참고자료로 사용한다면 좋겠다.

개발자는 배경색, 전경색, 기본색(base color), 텍스트 색상 스타일을 비롯해 다수의 스타일을 적용할 위젯 상태를 명시할 필요가 있다. 상태는 일부 함수에 대한 스톱 아이콘을 명시할 때에도 필요하다. 아래에는 다섯 가지의 위젯 상태를 소개하겠다.

- **NORMAL**: 정상적인 작동 시 위젯의 상태.
- **ACTIVE**: 토크를 누름 해제할 때와 같이 활성화된 위젯의 상태.
- **PRELIGHT**: 마우스가 위젯 위에 있으며, 버튼의 클릭에 응답할 것이다.
- **SELECTED**: 위젯 또는 위젯 텍스트가 선택되었다.
- **INSENSITIVE**: 위젯이 비활성화되어 사용자에게 응답하지 않을 것이다.

색상은 다수의 포맷으로 명시가 가능하다. 이러한 포맷으로 #RGB, #RRGGBB, #RRRGGBBBB, #RRRRGGGGBBBB와 같은 16진 포맷들이 있는데, 여기서 R, G, B는 각각 적색, 녹색, 청색을 나타내는 16진 숫자다. 색상을 {R, G, B}와 같이 명시할 수도 있는데, 여기서 값은 0과 65,535 사이의 정수이거나 0.0과 1.0 사이의 부동 소수점 값으로 주어진다.

표 C-1은 GTK+ 2.10 버전부터 지원되는 기본 RC 파일 스타일을 전부 열거한다. 스타일 설명 중 일부는 구현 방법의 예제도 포함한다.

스타일	설명
base[state]	다섯 가지 상태 중 하나로 텍스트의 편집을 허용하는 위젯의 (예: GtkEntry) 배경색을 설정하라. 예제: base[ACTIVE] = { 0.5, 0.3, 1.0 }
bg[state]	대부분 위젯의 배경색을 다섯 가지 상태 중 하나로 설정하라. 예제: bg[NORMAL] = "#036"
bg_pixmap[state]	위젯의 배경으로 사용할 이미지를 다섯 가지 상태 중 하나로 설정하라. 이미지 파일이 상대적일 경우 pixmap_path가 명시한 경로 중 하나가 검색될 것이다. 예제: bg_pixmap[SELECTED] = "image.xpm"
class::property	구체적인 위젯 클래스에 대한 스타일 프로퍼티를 설정하라. 예를 들어, GtkWidget 프로퍼티는 cursor-aspect-ratio, cursor-color, draw-border를 포함한다. 예제: class::cursor-aspect-ratio = 0.1
color["color_name"]	GTK+ 2.10 버전부터 개발자는 자신만의 색상을 정의할 수 있다. 색상은 @color_name으로 참조된다. 아래 소개된 표에서 더 상세한 정보를 찾을 수 있다.
engine	테마 엔진은 RC 파일로부터 자신만의 위젯 스타일을 정의하도록 해준다. 엔진의 사용에 관한 추가 정보는 GTK+ 문서에서 찾을 수 있다.

fg	대부분 위젯의 전경색을 다섯 가지 상태 중 하나로 설정하라. 예제: fg[PRELIGHT] = "#123456"
font_name	이 스타일을 위해 GTK+ 2.10 버전부터 font와 fontset 스타일이 무시된다. Pango Font Description 문자열과 마찬가지로 이 글꼴의 이름을 명시해야 한다. 예제: font_name = "Sans Bold 12"
stock["stockid"]	애플리케이션이 사용할 수 있는 새로운 스톡 항목을 정의하라. 스톡 항목은 이미지 파일명, 텍스트 방향(왼쪽으로 오른쪽, 혹은 오른쪽에서 왼쪽으로), 위젯 상태, 크기를 수락한다. 크기로는 gtk-menu, gtk-small-toolbar, gtk-large-toolbar, gtk-button, gtk-dialog가 있다. 마지막 세 개의 매개변수에서는 별표(*) 문자를 와일드카드로 사용할 수 있다. 예제: stock["myitem"] = { "myitem.png", LTR, NORMAL, "gtk-menu" }
text[state]	GtkEntry와 같은 위젯에 텍스트 색상을 설정하라. 예제: fg[PRELIGHT] = { 0, 65535, 0 }
xthickness	GTK+의 다양한 값에 대한 수평적 패딩을 설정하라. 이 값은 정수로 명시된다.
ythickness	GTK+의 다양한 값에 대한 수직적 패딩을 설정하라. 이 값은 정수로 명시된다.
표 C-1. RC 파일 스타일	

GTK+ 2.10 버전부터는 자신만의 색상을 정의할 수가 있다. 기존 색상을 변경하도록 4개의 함수가 제공되기도 한다. 아래 네 가지 방법들은 각각 지원되는 색상 표현식을 모두 수락한다.

- shade (factor, color): 명시된 색상을 연하거나 진하게 만든다. factor는 부동 소수점 수가 될 수 있으며, 1.0은 색상을 그대로 둔다. factor가 작을수록 색상은 어두워지고 클수록 연해진다.
- dark (color): 이 표현식은 shade (0.7, color)와 동일하다.
- lighter (color): 이 표현식은 shade (1.3, color)와 동일하다.
- mix (factor, color1, color2): 두 가지 색상을 섞어 새로운 색상을 생성하는데, factor가 0.0이면 color2가 나오고 1.0이면 color1이 된다.

이러한 방법들을 모두 이용해 색상을 생성할 수도 있다. 이해를 돕기 위해 색상 생성 표현식의 예를 몇 가지 소개하겠다.

```
color["blackwhite"] = mix (0.5, "#000000", "#FFFFFF")
color["darker"] = shade (0.5, @blackwhite)
color["multiple"] = shade (1.4, mix (0.1, "#369", { 0, 1.0, 0 })))
```

### Pango 텍스트 마크업 언어

Pango 텍스트 마크업 언어는 GtkLabel을 비롯해 몇 가지 위젯에서 XML 태그로 된 텍스트의 스타일을 변경할 수 있도록 해준다.

여러 속성과 함께 <span> 태그를 이용해 텍스트 스타일을 정의할 수 있다. 예를 들어 <span font\_desc='Sans Bold 12'>Text</span>은 태그들 사이의 텍스트를 명시된 글꼴로 설정한다. 표 C-2에서는 <span> 태그에 지원되는 속성의 리스트를 제공하겠다.



속성	설명
background	배경색을 설명하는 값. #RRGGBB와 같은 형태로 된 16진 RGB 값 또는 blue와 같이 지원하는 색상명이 가능하다.
face	Sans 또는 Monospace와 같은 폰트 패밀리명. 이 태그는 font_family와 동일하다.
fallback	기본적으로 활성화되는데, 활성화 시 시스템은 명시된 글꼴에 가장 가깝게 일치하는 글꼴의 검색을 시도할 것이다. 이 속성을 꺼서는 안 되지만, 불가피한 경우 false의 값을 사용해야 한다.
font_desc	"Sans Bold 12"와 같이 PangoFontDescription이 지원하는 글꼴 설명 문자열.
font_family	Sans 또는 Monospace와 같은 폰트 패밀리명. 이 태그는 face와 동일하다.
foreground	전경색을 설명하는 값. #RRGGBB와 같은 형태로 된 16진 RGB 값 또는 blue와 같이 지원하는 색상명이 가능하다.
lang	텍스트 문자열이 어떤 언어로 되어 있는지를 나타내는 언어 코드.
rise	이 값은 em 단위를 10,000로 나누어(10,000ths of an em unit) 수직 변위(vertical displacement)를 명시함으로써 슈퍼스크립트와 서브스크립트의 생성을 허용한다. 음의 값은 서브스크립트를, 양의 값은 슈퍼스크립트를 생성한다.
size	포인트를 1,024로 나누어(1,024ths of a point) 표시된 글꼴 크기. xx-small, x-small, small, medium, large, x-large, xx-large, larger, smaller 또한 사용 가능하다. 절대 크기는 font_desc를 이용 시 명시하기가 수월하다.
stretch	텍스트가 얼마나 늘어날 것인지 정의한다. ultracondensed, extracondensed, condensed, semicondensed, normal, semiexpanded, expanded, extraexpanded, ultraexpanded와 같은 값이 가능하다.
strikethrough	텍스트를 통해 단일 행을 위치시키려면 true를, 속성을 끄려면 false를 명시해야 한다.
strikethrough_color	취소선(strikethrough line) 색상을 설명하는 값. #RRGGBB와 같은 형태로 된 16진 RGB 값 또는 blue와 같이 지원하는 색상명이 가능하다.
style	이탤릭 스타일의 텍스트. normal, oblique, 또는 italic과 같은 값이 가능하다.
underline	텍스트에 밑줄 긋는 방식을 설명하는 값. single, double, low, 또는 none과 같은 값이 가능하다.
underline_color	밑줄 색상을 설명하는 값. #RRGGBB와 같은 형태로 된 16진 RGB 값 또는 blue와 같이 지원하는 색상명이 가능하다.
variant	normal 또는 smallcaps의 값으로, 텍스트를 모두 대문자로 렌더링하도록 해준다.
weight	텍스트의 굵기. ultralight, light, normal, bold, ultrabold, heavy, 또는 수치로 된 굵기 값이 가능하다.

표 C-2. Span 태그 속성

Pango 텍스트 마크업 언어는 수많은 편의(convenience) 태그를 제공하기도 한다. 이러한 태그는 다양한 <span> 속성을 대신해 사용이 가능하다. <span> 태그와 마찬가지로 항상 닫는 태그를 제공해야 한다(예: </span>).

- <b>: 글꼴을 볼드체로 만들며, <span weight="bold">와 동일하다.
- <big>: 글꼴을 현재 글꼴보다 크게 만들며, <span size="larger">와 동일하다.
- <i>: 글꼴을 이탤릭체로 만들며, <span style="italic">와 동일하다.
- <s>: 텍스트에 취소선을 그리며, <span strikethrough="true">와 동일하다.
- <sub>: 텍스트 문자열 서브스크립트를 만든다. 서브스크립트 텍스트에 기본 값을 사용한다.
- <sup>: 텍스트 문자열 슈퍼스크립트를 만든다. 슈퍼스크립트 텍스트에 기본 값을 사용한다.
- <small>: 글꼴을 현재 글꼴보다 작게 만들며, <span size="smaller">와 동일하다.
- <tt>: 글꼴을 monospace 글꼴로 만든다. 고정폭 간격을 요하는 코드 세그먼트나 문자열에 사용할 수 있다.
- <u>: 텍스트에 밑줄을 그으며, <span underline="single">과 동일하다.

## GtkTextTag 스타일

텍스트 태그는 GtkTextBuffer의 구체적인 섹션에 스타일을 정의하도록 해준다. 표 C-3은 GtkTextTag가 지원하는 스타일과 각 스타일이 지원하는 값의 타입에 대한 설명을 제공한다.

프로퍼티	타입	설명
background	gchar array	16진수 문자열로 된 배경색. 16진수 문자열로 된 배경색. 문자열은 #RRGGBB 포맷으로 명시되어야 한다.
background-full-height	gboolean	배경색이 행의 전체 높이를 채우는지 아니면 각 문자의 높이만큼만 채우는지 나타낸다.
background-gdk	GdkColor	배경색.
background-stipple	GdkPixmap	위젯의 배경으로 그릴 비트맵.
direction	GtkTextDirection	기본 텍스트 방향으로, GTK_TEXT_DIR_NONE, GTK_TEXT_DIR_LTR, GTK_TEXT_DIR_RTL 중 하나로 설정된다.
editable	gboolean	텍스트가 수정 가능한지 나타낸다.
family	gchar array	Sans 또는 Monospace와 같은 폰트 패밀리의 공식 명칭.
font	gchar array	PangoFontDescription이 허용하는 형태로 전체 글꼴을 설명하는 문자열.
font-desc	PangoFontDescription	위젯에 적용할 글꼴. 실제 글꼴 문자열을 명시하기 위해 font를 사용할 수도 있다.
foreground	gchar array	16진수 문자열로 된 전경색. 문자열은 #RRGGBB 포맷으로 명시되어야 한다.
foreground-gdk	GdkColor	전경색.
foreground-stipple	GdkPixmap	전경 마스크(foreground mask)로 사용할 비트맵.
indent	gint	문단을 들여쓰기할 픽셀 수를 설정하는 정수.
invisible	gboolean	텍스트의 숨김 여부를 나타낸다.
justification	GtkJustification	맞춤(justification 타입)으로, GTK_JUSTIFY_LEFT, GTK_JUSTIFY_RIGHT, GTK_JUSTIFY_CENTER 중 하나로 설정된다.
language	gchar array	기본 언어의 ISO 코드. 이전 설정을 제거하려면 NULL을 사용하라.

left-margin	gint	왼쪽 여백의 너비를 픽셀로 나타낸 값.
name	gchar array	텍스트 태그의 이름으로 사용할 수 있는 문자열. 이전 설정을 제거하려면 NULL을 사용하라.
paragraph-background	gchar array	16진수 문자열로 된 문단 배경색. 문자열은 #RRGGBB 포맷으로 명시되어야 한다.
paragraph-background-gdk	GdkColor	문단 배경색.
pixels-above-lines	gint	문단 위에 추가할 공간의 픽셀 수.
pixels-below-lines	gint	문단 아래에 추가할 공간의 픽셀 수.
pixels-inside-wrap	gint	래핑된 행 사이에 추가할 공간의 픽셀 수.
right-margin	gint	오른쪽 여백의 너비를 픽셀로 나타낸 값.
rise	gint	행의 하단면 위에 텍스트의 오프셋.
scale	gdouble	Pango 스케일 값으로 된 글꼴 크기로, PANGO_SCALE_XX_SMALL, PANGO_SCALE_X_SMALL, PANGO_SCALE_SMALL, PANGO_SCALE_MEDIUM, PANGO_SCALE_LARGE, PANGO_SCALE_X_LARGE, PANGO_SCALE_XX_LARGE 중 하나로 설정된다.
size	gint	Pango 단위로 된 글꼴 크기.
size-points	gdouble	포인트로 된 글꼴 크기.
stretch	Pango Stretch	텍스트가 얼마나 늘어날 것인지 정의하는 값으로, PANGO_STRETCH_ULTRA_CONDENSED, PANGO_STRETCH_EXTRA_CONDENSED, PANGO_STRETCH_CONDENSED, PANGO_STRETCH_SEMI_CONDENSED, PANGO_STRETCH_NORMAL, PANGO_STRETCH_SEMI_EXPANDED, PANGO_STRETCH_EXPANDED, PANGO_STRETCH_EXTRA_EXPANDED, PANGO_STRETCH_ULTRA_EXPANDED 중 하나로 설정된다.
strikethrough	gboolean	텍스트에 취소선을 그을 것인지 나타낸다.
style	Pango Style	글꼴 스타일 값으로, PANGO_STYLE_NORMAL, PANGO_STYLE_OBLIQUE, PANGO_STYLE_ITALIC 중 하나로 설정된다.
tabs	Pango TabArray	태그의 범위 내 모든 탭 문자에 사용할 커스텀 탭 배열.
underline	Pango Underline	밑줄 스타일로, PANGO_UNDERLINE_NONE, PANGO_UNDERLINE_SINGLE, PANGO_UNDERLINE_DOUBLE, PANGO_UNDERLINE_LOW, PANGO_UNDERLINE_ERROR 중 하나로 설정된다.
variant	Pango Variant	모든 텍스트가 모두 대문자 (PANGO_VARIANT_SMALL_CAPS) 또는 정상적으로 (PANGO_VARIANT_NORMAL) 렌더링되어야 한다.

weight	gint	글꼴 굵기로, PANGO_WEIGHT_ULTRALIGHT, PANGO_WEIGHT_LIGHT, PANGO_WEIGHT_NORMAL, PANGO_WEIGHT_SEMIBOLD, PANGO_WEIGHT_BOLD, PANGO_WEIGHT_ULTRABOLD, PANGO_WEIGHT_HEAVY 중 하나로 설정된다.
wrap-mode	GtkWrapMode	래핑 방식으로, GTK_WRAP_NONE, GTK_WRAP_CHAR, GTK_WRAP_WORD, GTK_WRAP_WORD_CHAR 중 하나로 설정된다.

표 C-3. GtkTextTag 스타일 프로퍼티

### 위젯 스타일 프로퍼티

많은 위젯들이 RC 파일을 이용해 변경될 수 있는 스타일 프로퍼티들을 갖는다. 표 C-4부터 C-32까지는 이 방법을 이용해 맞춤설정이 가능한 위젯들이 제공하는 스타일을 모두 제공하겠다.

프로퍼티	타입	설명
arrow-scaling	gfloat	화살표 크기의 스케일링에 사용된 0.0과 1.0 사이의 숫자로, 기본값은 0.7이다.

표 C-4. GtkArrow 스타일 프로퍼티

프로퍼티	타입	설명
content-padding	gint	각 페이지 내용 주위에 추가되는 패딩의 픽셀 수.
header-padding	gint	각 페이지의 헤더 주위에 추가되는 패딩의 픽셀 수.

표 C-5. GtkAssistant 스타일 프로퍼티

프로퍼티	타입	설명
child-displacement-x	gint	버튼의 자식 위젯의 수평 변위(horizontal displacement)로, 버튼을 누르면 발생할 것이다.
child-displacement-y	gint	버튼의 자식 위젯의 수직 변위(vertical displacement)로, 버튼을 누르면 발생할 것이다.
default-border	GtkBorder	버튼이 기본 위젯이 될 수 있을 때 버튼을 따라 추가할 추가 테두리.
default-outside-border	GtkBorder	버튼이 기본 위젯이 될 수 있을 때 버튼 외부를 따라 추가할 추가 테두리.
displace-focus	gboolean	TRUE로 설정 시 자식 변위 스타일 프로퍼티가 사용될 것이다.
image-spacing	gint	버튼에 포함된 텍스트와 이미지 사이에 추가할 공간의 픽셀 수.
inner-border	GtkBorder	버튼의 면과 그 자식 위젯의 면을 따라 위치시킬 테두리.

표 C-6. GtkButton 스타일 프로퍼티

프로퍼티	타입	설명
child-internal-pad-x	gint	패딩이 각 자식 위젯의 한 면에 위치된다.
child-min-height	gint	컨테이너 내에서 각 버튼의 최소 높이.

표 C-7. GtkButtonBox 스타일 프로퍼티

프로퍼티	타입	설명
indicator-size	gint	체크 버튼이나 라디오 버튼의 크기를 픽셀로 나타낸 값.
indicator-spacing	gint	체크 버튼 표시기(indicator) 주위에 추가할 패딩.

표 C-8. GtkCheckButton 스타일 프로퍼티

프로퍼티	타입	설명
indicator-size	gint	체크 버튼 표시기의 크기를 픽셀로 나타낸 값.

표 C-9. GtkCheckMenuItem 스타일 프로퍼티

프로퍼티	타입	설명
appears-as-list	gboolean	TRUE로 설정 시 위젯이 활성화되면 표시되는 드롭 다운 창이 메뉴 대신 리스트로 나타날 것이다.
arrow-size	gint	콤보 박스가 표시하는 화살표의 크기를 픽셀로 나타낸 값. 이는 최소값으로, 글꼴 크기가 커지면 값도 증가할 것이다.

표 C-10. GtkComboBox 스타일 프로퍼티

프로퍼티	타입	설명
action-area-border	gint	액션 영역 주위에 위치시킬 패딩의 픽셀 수로, 대화상자 하단면을 따라 발견된다.
button-spacing	gint	대화상자 액션 영역 내에서 버튼들 사이에 추가할 공간.
content-area-border	gint	대화상자의 주요 내용 주위에 위치시킬 패딩의 픽셀 수.

표 C-11. GtkDialog 스타일 프로퍼티

프로퍼티	타입	설명
inner-border	GtkBorder	GtkEntry 위젯의 텍스트와 그 면 사이에 위치시킬 패딩의 픽셀 수.

표 C-12. GtkEntry 스타일 프로퍼티

프로퍼티	타입	설명
expander-size	gint	익스팬더 화살표의 크기를 픽셀로 나타낸 값.
expander-spacing	gint	익스팬더 화살표 주위에 위치시킬 패딩을 픽셀로 나타낸 값.

표 C-13. GtkExpander 스타일 프로퍼티

프로퍼티	타입	설명
selection-box-alpha	guchar	선택(selection box) 박스의 알파값으로, 기본값으로 64가 설정된다.
selection-box-color	GdkColor	선택 박스가 표시하는 색상.

표 C-14. GtkIconView 스타일 프로퍼티

프로퍼티	타입	설명
double-arrow	gboolean	TRUE로 설정 시 메뉴를 통해 스크롤링하면 두 개의 화살표가 모두 표시될 것이다.
horizontal-offset	gint	하위메뉴의 본래 위치로부터 수평적 오프셋을 픽셀로 나타낸 값. 양수 또는 음수 값이 가능하다! 사실 기본값은 -1에 해당한다.
horizontal-padding	gint	메뉴의 좌측면과 우측면을 따라 추가할 패딩의 픽셀 수.
vertical-offset	gint	하위메뉴의 본래 위치로부터 수직적 오프셋을 픽셀로 나타낸 값. 양수 또는 음수 값이 가능하다!
vertical-padding	gint	메뉴의 상단면과 하단면을 따라 추가할 패딩의 픽셀 수.

표 C-15. GtkMenu 스타일 프로퍼티

프로퍼티	타입	설명
internal-padding	gint	메뉴 표시줄의 면과 메뉴 항목 사이에 위치시킬 패딩.
shadow-type	GtkShadowType	메뉴 표시줄의 면 주위에 위치시킬 그림자 타입.

표 C-16. GtkMenuBar 스타일 프로퍼티

프로퍼티	타입	설명
arrow-spacing	gint	항목이 하위메뉴를 포함할 때 메뉴 항목의 라벨과 화살표 사이에 추가할 패딩.
horizontal-padding	gint	메뉴 항목의 한쪽 면에 위치한 패딩의 픽셀 수.
selected-shadow-type	GtkShadowType	메뉴 항목의 면 주위에 위치시킬 그림자 타입.
toggle-spacing	gint	메뉴 항목의 텍스트와 아이콘 사이에 위치시킬 패딩의 픽셀 수.

표 C-17. GtkMenuItem 스타일 프로퍼티

프로퍼티	타입	설명
message-border	gint	메시지 대화상자에서 라벨과 이미지 주변에 추가할 패딩.
use-separator	gboolean	TRUE로 설정 시 메시지 대화상자의 내용과 그 버튼 사이에 구분자가 그려질 것이다.

표 C-18. GtkMessageDialog 스타일 프로퍼티

프로퍼티	타입	설명
arrow-spacing	gint	스크롤 화살표와 GtkNotebook 위젯의 탭 사이에 위치시킬 패딩.
has-backward-stepper	gboolean	TRUE로 설정 시 후방향(backward) 스크롤 화살표가 표시될 것이다.
has-forward-stepper	gboolean	TRUE로 설정 시 전방향(forward) 스크롤 화살표가 표시될 것이다.
has-secondary-backward-stepper	gboolean	TRUE로 설정 시 두 번째 후방향 스크롤 화살표가 탭의 다른 면에 위치될 것이다.
has-secondary-forward-stepper	gboolean	TRUE로 설정 시 두 번째 전방향 스크롤 화살표가 탭의 다른 면에 위치될 것이다.
tab-curvature	gint	선택된 탭과 선택 해제된 탭의 크기 차이.
tab-overlap	gint	주변에 탭이 겹치게 될 픽셀 수.

표 C-19. GtkNotebook 스타일 프로퍼티

프로퍼티 타입 설명		
handle-size	gint	두 개의 패인 사이에 위치한 구분자의 너비 또는 높이.
표 C-20. GtkPaned 스타일 프로퍼티		

프로퍼티 타입 설명		
xspacing	gint	위젯의 너비에 추가된 수평적 공간.
yspacing	gint	위젯의 높이에 추가된 수직적 공간.
표 C-21. GtkProgressBar 스타일 프로퍼티		

프로퍼티	타입	설명
activate-slider	gboolean	TRUE로 설정 시 슬라이더를 드래그하면 활성화(active) 상태로 그려지고, 그에 따라 그림자가 안쪽으로 그려질 것이다.
arrow-displacement-x	gint	수평 화살표를 누르면 화살표가 가리키는 방향으로 그만큼 범위가 이동할 것이다.
arrow-displacement-y	gint	수직 화살표를 누르면 화살표가 가리키는 방향으로 그만큼 범위가 이동할 것이다.
slider-width	gint	위젯의 방향에 따라 실제 스크롤바 또는 스케일 영역의 너비나 높이.
stepper-size	gint	스텝퍼(stepper) 버튼의 크기로, 범위 위젯의 타입에 따라 좌우된다.
stepper-spacing	gint	스텝퍼 버튼과 thumb 사이에 추가할 패딩의 양. 양수로 설정 시 through-under-steppers가 설정될 것이다.
trough-border	gint	스텝퍼와 외부 trough 사이에 추가할 패딩.
trough-side-details	gboolean	TRUE로 설정 시 스텝퍼의 면에 세부 내용이 표시될 것이다.
trough-upper-steppers	gboolean	TRUE로 설정 시 전체 범위를 따라 trough가 그려질 것이다.
표 C-22. GtkRange 스타일 프로퍼티		

프로퍼티	타입	설명
slider-length	gint	픽셀로 된 GtkScale의 슬라이더 길이.
value-spacing	gint	스케일의 값과 trough 간에 패딩이 표시될 경우 패딩의 양.
표 C-23. GtkScale 스타일 프로퍼티		

프로퍼티	타입	설명
fixed-slider-length	gboolean	TRUE로 설정 시 범위의 크기와 무관하게 슬라이더는 강제로 최소 길이로 유지될 것이다.
has-backward-stepper	gboolean	TRUE로 설정 시 후방향 화살표가 표시될 것이다.
has-forward-stepper	gboolean	TRUE로 설정 시 전방향 화살표가 표시될 것이다.
has-secondary-backward-stepper	gboolean	TRUE로 설정 시 2차 후방향 화살표가 스크롤바의 반대편에 위치할 것이다.
has-secondary-forward-stepper	gboolean	TRUE로 설정 시 2차 전방향 화살표가 스크롤바의 반대편에 위치할 것이다.

min-slider-length	gint	슬라이더의 최소 길이. fixed-slider-length가 TRUE로 설정된 경우 이것은 일정한 스크롤러 크기가 될 것이다.
표 C-24. GtkScrollbar 스타일 프로퍼티		

프로퍼티	타입	설명
scrollbar-spacing	gint	스크롤이 있는 창의 내용과 스크롤바 사이에 위치시킬 패딩의 픽셀 수.
표 C-25. GtkScrolledWindow 스타일 프로퍼티		

프로퍼티	타입	설명
shadow-type	GtkShadowType	스핀 버튼 주위에 그릴 그림자 타입. 그림자 기본 타입은 GTK_SHADOW_IN이다.
표 C-26. GtkSpinButton 스타일 프로퍼티		

프로퍼티	타입	설명
shadow-type	GtkShadowType	상태 표시줄의 내용 주위에 그릴 그림자 타입. 그림자 기본 타입은 GTK_SHADOW_IN이다.
표 C-27. GtkStatusbar 스타일 프로퍼티		

프로퍼티	타입	설명
error-underline-color	GdkColor	오류가 표시된 텍스트 아래 밑줄을 그릴 때 사용할 색상.
표 C-28. GtkTextView 스타일 프로퍼티		

프로퍼티	타입	설명
button-relief	GtkReliefStyle	툴바 버튼 주위에 위치시킬 테두리 타입.
internal-padding	gint	툴바의 테두리와 툴바 버튼 사이에 위치시킬 패딩의 픽셀 수.
max-child-expand	gint	각 툴 항목의 크기를 조정할 수 있는 최대 너비 또는 높이.
shadow-type	GtkShadowType	툴바 주위에 그릴 그림자 타입. 기본 그림자 타입은 GTK_SHADOW_OUT이다.
space-size	gint	툴바에서 발견되는 간격(spacer)의 너비 또는 높이.
space-style	GtkToolbarSpaceStyle	툴바가 표시하게 될 간격의 타입. GTK_TOOLBAR_SPACE_EMPTY 또는 GTK_TOOLBAR_SPACE_LINE으로 설정 가능한데, 각각 빈 패딩과 하나의 행을 표시할 것이다.
표 C-29. GtkToolbar 스타일 프로퍼티		



프로퍼티 타입 설명		
icon-spacing	gint	툴 버튼의 라벨과 아이콘 사이에 위치시킬 패딩의 픽셀 수.
표 C-30. GtkToolButton 스타일 프로퍼티		

프로퍼티	타입	설명
allow-rules	gboolean	TRUE로 설정 시 두 개의 색을 교대로 이용해(alternating) 행을 그릴 수 있다. 당시에 작업을 실행할 뿐, 이 기능을 활성화하지는 않음을 주의하라!
even-row-color	GdkColor	두 개의 행마다 다른 색으로 그릴 때 짝수 행의 배경색.
expander-size	gint	행 익스팬더의 크기를 픽셀로 나타낸 값으로, 기본값은 12다.
grid-line-pattern	gchararray	트리 뷰에서 격자선에 사용할 패턴.
grid-line-width	gint	트리 뷰에 그린 격자선의 너비.
horizontal-separator	gint	셀들 간 위치시킬 수평 공간으로, 양으로 된 짝수여야 한다.
indent-expanders	gboolean	TRUE로 설정 시 행의 내용이 증가하면 익스팬더가 들어 썬질 것이다.
odd-row-color	GdkColor	두 개의 행마다 다른 색으로 그릴 때 홀수 행의 배경색.
row-ending-details	gboolean	TRUE로 설정 시 행의 배경 테마(background theming)가 활성화될 것이다.
tree-line-pattern	gchararray	트리 뷰 직선을 그리는 데 사용된 패턴을 설명하는 문자열.
tree-line-width	gint	트리 뷰 직선의 너비를 픽셀로 나타낸 값.
vertical-separator	gint	셀들 간 위치시킬 수직 공간으로, 양으로 된 짝수여야 한다.
표 C-31. GtkTreeView 스타일 프로퍼티		

프로퍼티	타입	설명
cursor-aspect-ratio	gfloat	삼입 커서를 그릴 변의 가로세로비(aspect ratio)로, 0.0과 1.0 사이의 값이어야 하며 기본값은 0.04다.
cursor-color	GdkColor	삼입 커서를 그리는 데 사용할 색상.
draw-border	GtkBorder	위젯의 초기 할당 외에 위치시킬 테두리의 양.
focus-line-pattern	gchararray	위젯에 포커스가 있을 때 위젯 주위에 그려지는 패턴을 설명하는 문자열.
focus-line-width	gint	위젯에 포커스가 있을 때 그려지는 직선의 너비.
focus-padding	gint	포커스 선과 위젯의 면 사이에 위치시킬 패딩의 픽셀 수.
interior-focus	gboolean	TRUE로 설정 시 위젯에 대해 포커스 선이 그려질 것이다.
link-color	GdkColor	방문하지 않은 링크를 그리는 데 사용할 색상.
scroll-arrow-length	gint	수평 스크롤 화살표가 있는 위젯에서 화살표의 길이.

scroll-arrow-length	gint	수직 스크롤 화살표가 있는 위젯에서 화살표의 길이.
secondary-cursor-color	GdkColor	2차 삽입 커서를 그리는 데 사용할 색상. 이 커서는 좌측에서 우측 방향의 텍스트와 우측에서 좌측 방향의 텍스트를 동시에 편집할 때 표시된다.
separator-height	gint	여러 위젯에 표시되는 다양한 타입의 구분자의 높이. 이 프로퍼티는 wide-separators가 설정되었을 때에만 작동할 것이다.
separator-width	gint	여러 위젯에 표시되는 다양한 타입의 구분자의 너비. 이 프로퍼티는 wide-separators가 설정되었을 때에만 작동할 것이다.
visited-link-color	GdkColor	방문한 링크를 그릴 때 사용할 색상.
wide-separators	gboolean	TRUE로 설정 시 separator-width 와 separator-height을 이용해 구분자의 너비와 높이 프로퍼티를 설정할 수 있다. 이런 경우 직선 대신 박스로 그려질 것이다.

표 C-32. GtkWidget 스타일 프로퍼티

## Notes

# Foundations of GTK Development: Appendix D













## 부록 D GTK+ 스톡 항목

### GTK+ 스톡 항목













스톡 항목은 이미지와 약간의 텍스트를 제공하는 데에 자주 사용되는 항목이다. 이는 메뉴, 툴바, 버튼을 비롯해 여러 장소에서 사용된다. 스톡 문자열은 각 스톡 항목을 식별하기에 충분하지만 편의상 전처리기 매크로가 제공된다.


스톡 항목은 오른쪽에서 왼쪽으로 변형체(right-to-left variant)를 가질 수 있으므로 이를 선호하는 사람들에게 이용할 수 있다. 예를 들자면 **GTK\_STOCK\_GOTO\_FIRST**, **GTK\_STOCK\_GOTO\_LAST**, **GTK\_STOCK\_GO\_BACK**, **GTK\_STOCK\_GO\_FORWARD**, **GTK\_STOCK\_INDENT**, **GTK\_STOCK\_JUMP\_TO**, **GTK\_STOCK\_MEDIA\_FORWARD**, **GTK\_STOCK\_MEDIA\_NEXT**, **GTK\_STOCK\_MEDIA\_PLAY**, **GTK\_STOCK\_MEDIA\_PREVIOUS**, **GTK\_STOCK\_REWIND**, **GTK\_STOCK\_REDO**, **GTK\_STOCK\_REVERT\_TO\_SAVED**, **GTK\_STOCK\_UNDELETE**, **GTK\_STOCK\_UNDO**, **GTK\_STOCK\_UNINDENT** 가 있다.













애플리케이션에서 자신만의 스톡 항목을 등록하는 것도 가능하다. 표 D-1은 GTK+ 2.10 버전부터 이용할 수 있는 98개 항목을 열거하고 있다. 그 중 일부는 GTK+ 2.0 배포판부터 소개되며, 각 항목의 소개 일자도 명시하겠다.













Stock ID	Display	Introduced
GTK_STOCK_ABOUT	 About	GTK+ 2.6
GTK_STOCK_ADD	 Add	GTK+ 2.0
GTK_STOCK_APPLY	 Apply	GTK+ 2.0
GTK_STOCK_BOLD	 Bold	GTK+ 2.0
GTK_STOCK_CANCEL	 Cancel	GTK+ 2.0
GTK_STOCK_CDROM	 CD-Rom	GTK+ 2.0
GTK_STOCK_CLEAR	 Clear	GTK+ 2.0
GTK_STOCK_CLOSE	 Close	GTK+ 2.0
GTK_STOCK_COLOR_PICKER	 Color picker	GTK+ 2.2
GTK_STOCK_CONNECT	 Connect	GTK+ 2.6
GTK_STOCK_CONVERT	 Convert	GTK+ 2.0
GTK_STOCK_COPY	 Copy	GTK+ 2.0

GTK_STOCK_CUT	 Cut	GTK+ 2.0
GTK_STOCK_DELETE	 Delete	GTK+ 2.0
GTK_STOCK_DIALOG_AUTHENTICATION	 Authentication	GTK+ 2.4
GTK_STOCK_DIALOG_ERROR	 Error	GTK+ 2.0
GTK_STOCK_DIALOG_INFO	 Information	GTK+ 2.0
GTK_STOCK_DIALOG_QUESTION	 Question	GTK+ 2.0
GTK_STOCK_DIALOG_WARNING	 Warning	GTK+ 2.0
GTK_STOCK_DIRECTORY	 Directory	GTK+ 2.6
GTK_STOCK_DISCONNECT	 Disconnect	GTK+ 2.6
GTK_STOCK_DND	 Drag-And-Drop	GTK+ 2.0
GTK_STOCK_DND_MULTIPLE	 Drag-And-Drop multiple	GTK+ 2.0
GTK_STOCK_EDIT	 Edit	GTK+ 2.6













GTK_STOCK_EXECUTE	 Execute	GTK+ 2.0
GTK_STOCK_FILE	 File	GTK+ 2.6
GTK_STOCK_FIND	 Find	GTK+ 2.0
GTK_STOCK_FIND_AND_REPLACE	 Find and Replace	GTK+ 2.0
GTK_STOCK_FLOPPY	 Floppy	GTK+ 2.0
GTK_STOCK_FULLSCREEN	 Fullscreen	GTK+ 2.8
GTK_STOCK_GO_BACK	 Back	GTK+ 2.0
GTK_STOCK_GO_DOWN	 Down	GTK+ 2.0
GTK_STOCK_GO_FORWARD	 Forward	GTK+ 2.0
GTK_STOCK_GO_UP	 Up	GTK+ 2.0
GTK_STOCK_GOTO_BOTTOM	 Bottom	GTK+ 2.0
GTK_STOCK_GOTO_FIRST	 First	GTK+ 2.0


GTK_STOCK_GOTO_LAST	 Last	GTK+ 2.0
GTK_STOCK_GOTO_TOP	 Top	GTK+ 2.0
GTK_STOCK_HARDDISK	 Harddisk	GTK+ 2.4
GTK_STOCK_HELP	 Help	GTK+ 2.0
GTK_STOCK_HOME	 Home	GTK+ 2.0
GTK_STOCK_INDENT	 Increase Indent	GTK+ 2.4
GTK_STOCK_INDEX	 Index	GTK+ 2.0
GTK_STOCK_INFO	 Information	GTK+ 2.8
GTK_STOCK_ITALIC	 Italic	GTK+ 2.0
GTK_STOCK_JUMP_TO	 Jump to	GTK+ 2.0
GTK_STOCK_JUSTIFY_CENTER	 Center	GTK+ 2.0
GTK_STOCK_JUSTIFY_FILL	 Fill	GTK+ 2.0



GTK_STOCK_JUSTIFY_LEFT	 Left	GTK+ 2.0
GTK_STOCK_JUSTIFY_RIGHT	 Right	GTK+ 2.0
GTK_STOCK_LEAVE_FULLSCREEN	 Leave Fullscreen	GTK+ 2.8
GTK_STOCK_MEDIA_FORWARD	 Forward	GTK+ 2.6
GTK_STOCK_MEDIA_NEXT	 Next	GTK+ 2.6
GTK_STOCK_MEDIA_PAUSE	 Pause	GTK+ 2.6
GTK_STOCK_MEDIA_PLAY	 Play	GTK+ 2.6
GTK_STOCK_MEDIA_PREVIOUS	 Previous	GTK+ 2.6
GTK_STOCK_MEDIA_RECORD	 Record	GTK+ 2.6
GTK_STOCK_MEDIA_REWIND	 Rewind	GTK+ 2.6
GTK_STOCK_MEDIA_STOP	 Stop	GTK+ 2.6
GTK_STOCK_MISSING_IMAGE	 Missing Image	GTK+ 2.0

GTK_STOCK_NETWORK	 Network	GTK+ 2.4
GTK_STOCK_NEW	 New	GTK+ 2.0
GTK_STOCK_NO	 No	GTK+ 2.0
GTK_STOCK_OK	 OK	GTK+ 2.0
GTK_STOCK_OPEN	 Open	GTK+ 2.0
GTK_STOCK_ORIENTATION_LANDSCAPE	 Landscape	GTK+ 2.10
GTK_STOCK_ORIENTATION_PORTRAIT	 Portrait	GTK+ 2.10
GTK_STOCK_ORIENTATION_REVERSE_LANDSCAPE	 Reverse Landscape	GTK+ 2.10
GTK_STOCK_ORIENTATION_REVERSE_PORTRAIT	 Reverse Portrait	GTK+ 2.10
GTK_STOCK_PASTE	 Paste	GTK+ 2.0
GTK_STOCK_PREFERENCES	 Preferences	GTK+ 2.0
GTK_STOCK_PRINT	 Print	GTK+ 2.0



GTK_STOCK_PRINT_PREVIEW	 Print Preview	GTK+ 2.0
GTK_STOCK_PROPERTIES	 Properties	GTK+ 2.0
GTK_STOCK_QUIT	 Quit	GTK+ 2.0
GTK_STOCK_REDO	 Redo	GTK+ 2.0
GTK_STOCK_REFRESH	 Refresh	GTK+ 2.0
GTK_STOCK_REMOVE	 Remove	GTK+ 2.0
GTK_STOCK_REVERT_TO_SAVED	 Revert	GTK+ 2.0
GTK_STOCK_SAVE	 Save	GTK+ 2.0
GTK_STOCK_SAVE_AS	 Save As	GTK+ 2.0
GTK_STOCK_SELECT_ALL	 Select All	GTK+ 2.10
GTK_STOCK_SELECT_COLOR	 Color	GTK+ 2.0
GTK_STOCK_SELECT_FONT	 Font	GTK+ 2.0

GTK_STOCK_SORT_ASCENDING	 Ascending	GTK+ 2.0
GTK_STOCK_SORT_DESCENDING	 Descending	GTK+ 2.0
GTK_STOCK_SPELL_CHECK	 Spell Check	GTK+ 2.0
GTK_STOCK_STOP	 Stop	GTK+ 2.0
GTK_STOCK_STRIKETHROUGH	 Strikethrough	GTK+ 2.0
GTK_STOCK_UNDELETE	 Undelete	GTK+ 2.0
GTK_STOCK_UNDERLINE	 Underline	GTK+ 2.0
GTK_STOCK_UNDO	 Undo	GTK+ 2.0
GTK_STOCK_UNINDENT	 Decrease Indent	GTK+ 2.4
GTK_STOCK_YES	 Yes	GTK+ 2.0
GTK_STOCK_ZOOM_100	 Normal Size	GTK+ 2.0
GTK_STOCK_ZOOM_FIT	 Best Fit	GTK+ 2.0

GTK_STOCK_ZOOM_IN	 Zoom In	GTK+ 2.0
GTK_STOCK_ZOOM_OUT	 Zoom Out	GTK+ 2.0
표 D-1. GTK+ 스톡 항목		

## Notes

# FoundationsofGTKDevelopment:Appendix E

부록 E GError 타입

## GError 타입

GLib은 오류 전파에 대해 GError라고 불리는 표준 방법을 제공한다. 이번 부록에서는 GTK+ 2.10 버전부터 제공되는 GError 도메인의 리스트와 함께 각 도메인에 해당하는 오류 타입을 소개하겠다.

GError 구조체는 세 가지 요소, 즉 오류 도메인, 메시지 문자열, 오류 코드를 제공한다.

```
struct GError
{
    GQuark domain;
    gchar *message;
    gint code;
};
```

각 오류 도메인은 유사한 타입의 오류 집합을 나타낸다. 그 예제로 G\_BOOKMARK\_FILE\_ERROR, GDK\_PIXBUF\_ERROR, G\_FILE\_ERROR를 들 수 있겠다. 이들은 항상 <NAMESPACE>\_<MODULE>\_ERROR 식으로 명명되는데, 네임스페이스는 함수를 포함하는 라이브러이며 모듈은 위젯 또는 객체 타입을 나타낸다.

message는 사람이 읽을 수 있는 문자열로, 오류를 설명한다. 만일 발생한 오류 타입에 대해 사용자가 시각적 피드백을 기대할만한 상황이라면 개발자는 message를 출력해야 한다. 코드를 디버깅할 때도 매우 유용하게 사용된다.

오류 code는 도메인에서 발생한 오류에 특적이다. 각 오류 코드는 도메인명 뒤에 오류 타입이 따라오는 형태로 구성된다. 가령, 오류 타입이 G\_BOOKMARK\_FILE\_ERROR\_INVALID\_URI 라면 G\_BOOKMARK\_FILE\_ERROR 도메인이 속하는 것이다.

대부분의 오류 코드 도메인에는 일반적인 실패 코드인 <NAMESPACE>\_<MODULE>\_ERROR\_FAILED 도 포함되어 있다. 특정 오류를 이용할 수 없는 경우 이 코드가 리턴될 것이다.

표 E-1부터 E-14까지는 GTK+와 그것이 지원하는 라이브러리에서 발견되는 GError 열거의 참고자료를 모두 제공한다. 각 오류와 함께 무엇이 발생했는지에 대한 설명을 제공하겠다.

오류 값	설명
G_BOOKMARK_FILE_ERROR_INVALID_URI	함수로 제공된 URI가 올바르게 포맷팅되지 않았다.
G_BOOKMARK_FILE_ERROR_INVALID_VALUE	요청한 필드를 찾을 수 없다.
G_BOOKMARK_FILE_ERROR_APP_NOT_REGISTERED	요청한 애플리케이션이 북마크를 등록하지 않았다.
G_BOOKMARK_FILE_ERROR_URI_NOT_FOUND	함수로 제공된 요청 URI를 찾을 수 없다.
G_BOOKMARK_FILE_ERROR_READ	문서가 올바르게 포맷팅되지 않았다.
G_BOOKMARK_FILE_ERROR_UNKNOWN_ENCODING	파싱 중인 문서로 인해 알려지지 않은 인코딩이 실행되었다.
G_BOOKMARK_FILE_ERROR_WRITE	북마크를 쓰는 데 실패하였다. 특정한 유형의 쓰기 오류가 발생하였다.
G_BOOKMARK_FILE_ERROR_FILE_NOT_FOUND	요청한 북마크 파일을 찾을 수 없다.
표 E-1. GBookmarkFileError 열거 값	

오류 값	설명
GDK_PIXBUF_ERROR_CORRUPT_IMAGE	이미지 파일이 깨졌다.
GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY	이미지를 저장하기에 메모리가 충분하지 않다.
GDK_PIXBUF_ERROR_BAD_OPTION	올바르지 않은 옵션이 전달되었다. 이 오류는 이미지를 저장 시 발생할 수 있다.
GDK_PIXBUF_ERROR_UNKNOWN_TYPE	함수가 이미지 유형을 감지할 수 없다.
GDK_PIXBUF_ERROR_UNSUPPORTED_OPERATION	이 함수는 명시된 이미지에서 연산을 실행할 수 없다.
GDK_PIXBUF_ERROR_FAILED	그 외 모든 오류에 대한 일반적 실패 코드다.
표 E-2. GdkPixbufError 열거 값	

오류 값	설명
G_FILE_ERROR_EXIST	애플리케이션이 연산을 실행할 권한을 갖고 있지 않다.
G_FILE_ERROR_ISDIR	파일이 디렉터리여서 쓰기용으로 열 수 없다.
G_FILE_ERROR_ACCESS	파일 권한이 현재 연산을 허용하지 않는다.
G_FILE_ERROR_NAME_TOO_LONG	명시된 파일명이 너무 길다.
G_FILE_ERROR_NOENT	파일 또는 디렉터리가 존재하지 않는다.
G_FILE_ERROR_NOTDIR	명시된 위치가 디렉터리가 아닌데 옵션은 디렉터리를 필요로 한다.
G_FILE_ERROR_NXIO	파일이 위치한 장치를 찾을 수 없다.
G_FILE_ERROR_NODEV	파일 유형이 매핑을 지원하지 않는다.
G_FILE_ERROR_ROFS	파일 시스템이 읽기 전용으로 되어 있다.
G_FILE_ERROR_TXTBSY	텍스트 파일이 현재 사용 중이다(busy).
G_FILE_ERROR_FAULT	올바르지 않은 메모리 위치를 향한 포인터가 전달되었다.
G_FILE_ERROR_LOOP	원형의 심볼릭 링크가 감지되었다.
G_FILE_ERROR_NOSPC	디스크가 가득 차서 이용할 수 있는 공간이 없다.
G_FILE_ERROR_NOMEM	이용할 수 있는 메모리가 없으며, 가상 메모리가 가득 찼다.
G_FILE_ERROR_MFILE	현재 프로세스에 너무 많은 오픈 파일이 있다.
G_FILE_ERROR_NFILE	전체 시스템에 너무 많은 파일이 열려 있다.
G_FILE_ERROR_BADF	쓰기에 읽기 파일 디스크립터가 명시되거나, 읽기에 쓰기 파일 디스크립터가 명시되었다.
G_FILE_ERROR_INVAL	올바르지 않은 인자(argument)가 전달되었다.

G_FILE_ERROR_PIPE	파이프가 깨졌거나 블로킹되었다.
G_FILE_ERROR_AGAIN	자원이 깨졌지만 조금 있다 다시 시도하면 작동할 수도 있다.
G_FILE_ERROR_INTR	함수 호출이 방해받았다.
G_FILE_ERROR_IO	디스크 상에서 읽기 또는 쓰기 오류가 발생하였다.
G_FILE_ERROR_PERM	연산이 허용되지 않는다.
G_FILE_ERROR_NOSYS	함수가 자신의 운영체제에 맞게 구현되지 않았다.
G_FILE_ERROR_FAILED	확인할 수 없는 이유로 연산이 실패하였다.
표 E-3. GFileError 열거 값	

오류 값	설명
G_KEY_FILE_ERROR_UNKNOWN_ENCODING	파싱 중인 문서로 인해 알려지지 않은 인코딩이 실행되었다.
G_KEY_FILE_ERROR_PARSE	파싱 중인 문서가 올바르게 포맷팅되지 않았다.
G_KEY_FILE_ERROR_NOT_FOUND	함수로 제공된 파일을 찾을 수 없다.
G_KEY_FILE_ERROR_KEY_NOT_FOUND	함수가 요청한 키를 찾을 수 없다.
G_KEY_FILE_ERROR_GROUP_NOT_FOUND	함수가 요청한 그룹을 찾을 수 없다.
G_KEY_FILE_ERROR_INVALID_VALUE	함수로 제공된 값이 성공적으로 파싱되지 않았다.
표 E-4. GKeyFileError 열거 값	

오류 값	설명
G_MARKUP_ERROR_BAD_UTF8	파싱 중인 텍스트가 유효한 UTF-8 포맷으로 명시되지 않았다. 포맷팅을 변경하고 다시 시도할 필요가 있다.
G_MARKUP_ERROR_EMPTY	문서에 내용이 없거나 공백만 포함한다.
G_MARKUP_ERROR_PARSE	파싱 중인 문서가 올바르게 포맷팅되지 않았다.
G_MARKUP_ERROR_UNKNOWN_ELEMENT	함수로 명시한 요소를 찾을 수 없다. 이 값은 GMarkupParser 함수에 의해서만 설정되어야 한다.
G_MARKUP_ERROR_UNKNOWN_ATTRIBUTE	함수로 명시한 속성을 찾을 수 없다. 이 값은 GMarkupParser 함수에 의해서만 설정되어야 한다.
G_MARKUP_ERROR_INVALID_CONTENT	문서의 내용에 문제가 발생하여 오류를 야기하였다. 이 값은 GMarkupParser 함수에 의해서만 설정되어야 한다.
표 E-5. GMarkupError 열거 값	

오류 값	설명
G_OPTION_ERROR_UNKNOWN_OPTION	파서가 함수로 제공된 옵션을 실현하지 않았다. 이 오류는 GOptionContext를 설정하여 알려지지 않은 옵션을 무시하지 않을 때에만 보고될 것이다.
G_OPTION_ERROR_BAD_VALUE	값을 올바르게 파싱할 수 없다.
G_OPTION_ERROR_FAILED	GOptionArgFunc 타입의 콜백 함수가 실패하였다.
표 E-6. GOptionError 열거 값	

오류 값	설명
G_SHELL_ERROR_BAD_QUOTING	쿼팅(Quoting)이 올바르게 매치되지 않았거나 이해할 수 없다.
G_SHELL_ERROR_EMPTY_STRING	파싱될 문자열이 완전히 비어 있다.
G_SHELL_ERROR_FAILED	다른 타입의 GShellError가 발생하였다. 상세한 정보는 error->message 를 참고한다.
표 E-7. GShellError 열거 값	

오류 값	설명
G_SPAWN_ERROR_FORK	이용 가능한 메모리가 부족하여 포크(fork)가 실패하였다.
G_SPAWN_ERROR_READ	파이프를 선택하거나 읽는 데에 실패하였다.
G_SPAWN_ERROR_CHDIR	작업 디렉터리를 성공적으로 변경하는 데 실패하였다.
G_SPAWN_ERROR_ACCES	경로 접두사에 대한 검색 권한이 거부되었거나, 새로운 파일이 일반 파일이 아니거나, 실행 권한이 거부되었거나, 파일의 실행이 비활성화된 채로 파일 시스템에 마운트되어 있었다(execv()가 EACCES를 실패하였다).
G_SPAWN_ERROR_PERM	프로세스의 권한이 올바르지 않아 연산이 허용되지 않았다(execv()가 EPERM을 실패하였다).
G_SPAWN_ERROR_2BIG	시스템에서 설정한 한계로 인해 프로세스의 인자 리스트가 너무 길다(execv()가 E2BIG을 실패하였다).
G_SPAWN_ERROR_NOEXEC	파일이 실행되는 데 필요한 권한을 갖고 있지 않다(execv()가 ENOEXEC를 실패하였다).
G_SPAWN_ERROR_NAMETOOLONG	전체 경로 또는 경로의 일부 길이가 최대 허용 길이를 초과하였다(execv()가 ENAMETOOLONG을 실패하였다).
G_SPAWN_ERROR_NOENT	프로세스 파일이 존재하지 않는다(execv()가 ENOENT를 실패하였다).
G_SPAWN_ERROR_NOMEM	프로세스에 최대 가상 메모리 할당이 존재하며, 최대량보다 많은 메모리가 요구되었다(execv()가 ENOMEM을 실패하였다).
G_SPAWN_ERROR_NOTDIR	경로 접두사가 유효하지 않은 디렉터리를 가리켰다(execv()가 ENOTDIR을 실패하였다).
G_SPAWN_ERROR_LOOP	함수가 경로에 대해 너무 많은 심볼릭 링크가 감지되었다(execv()가 ELOOP를 실패하였다).
G_SPAWN_ERROR_TXTBUSY	다른 프로세스에서 이미 열려 있어서 해당 프로세스를 열 수 없다(execv()가 ETXTBUSY를 실패하였다).
G_SPAWN_ERROR_IO	읽는 도중에 입력 또는 출력 오류가 발생하였다(execv()가 EIO를 실패하였다).
G_SPAWN_ERROR_NFILE	시스템이 지원하는 오픈 파일의 최대 개수에 도달하였다(execv()가 ENFILE를 실패하였다).
G_SPAWN_ERROR_MFILE	시스템이 하나의 프로세스에 지원하는 오픈 파일의 최대 개수에 도달하였다(execv()가 EMFILE를 실패하였다).
G_SPAWN_ERROR_INVAL	함수로 전달된 매개변수가 올바르게 포맷팅되지 않았다(execv()가 EINVAL을 실패하였다).
G_SPAWN_ERROR_ISDIR	파일이 디렉터리로 발견되었고, 디렉터리에 지원되지 않는 연산의 실행이 시도되었다(execv()가 EISDIR을 실패하였다).
G_SPAWN_ERROR_LIBBAD	접근을 시도한 공유 라이브러리가 손상되었다(execv()가 ELIBBAD를 실패하였다).

G_SPAWN_ERROR_FAILED	그 외 다른 치명적인 오류가 발생하였다. 상세한 정보는 error->message 를 참고한다.
표 E-8. GSpawnError 열거 값	

오류 값	설명
G_THREAD_ERROR_AGAI_N	쓰레드(thread)를 생성하는 데에 이용 가능한 자원이 충분하지 않았다. 이런 경우 조금 있다 다시 시도해야 한다.
표 E-9. GThreadError 열거 값	

오류 값	설명
GTK_FILE_CHOOSER_ERROR_NONEXISTENT	GtkFileChooser 로 명시한 파일이 존재하지 않는다.
GTK_FILE_CHOOSER_ERROR_BAD_FILENAME	함수로 명시한 파일명이 올바로 포맷팅되지 않았다.
GTK_FILE_CHOOSER_ERROR_ALREADY_EXISTS	함수로 명시한 파일명이 이미 존재한다.
표 E-10. GtkFileChooserError 열거 값	

오류 값	설명
GTK_ICON_THEME_NOT_FOUND	함수에 매개변수로 명시된 아이콘이 GtkIconTheme에 존재하지 않는다.
GTK_ICON_THEME_FAILED	다른 타입의 GtkIconTheme 오류가 발생하였다. 상세한 정보는 error->message를 참고한다.
표 E-11. GtkIconThemeError 열거 값	

오류 값	설명
GTK_PRINT_ERROR_GENERAL	일반적인 출력 오류가 발생하였다. 상세한 정보는 error->message를 참고한다.
GTK_PRINT_ERROR_INTERNAL_ERROR	출력 시스템에서 내부적으로 오류가 발생하였다. 사용자의 시스템에 문제가 발생했다는 의미다.
GTK_PRINT_ERROR_NOMEM	출력 연산을 계속하기에 메모리가 충분하지 않다.
표 E-12. GtkPrintError 열거 값	

오류 값	설명
GTK_RECENT_CHOOSER_ERROR_NOT_FOUND	함수로 명시한 파일이 존재하지 않는다.
GTK_RECENT_CHOOSER_ERROR_INVALID_URI	함수로 명시한 URI가 올바로 포맷팅되지 않았다.
표 E-13. GtkRecentChooserError 열거 값	

오류 값	설명
GTK_RECENT_MANAGER_ERROR_NOT_FOUND	함수로 명시한 URI가 리스트에 존재하지 않았다.
GTK_RECENT_MANAGER_ERROR_INVALID_URI	함수로 명시한 URI가 올바로 포맷팅되지 않았다.
GTK_RECENT_MANAGER_ERROR_INVALID_ENCODING	명시된 문자열이 UTF-8 인코딩으로 제공되지 않았다.
GTK_RECENT_MANAGER_ERROR_NOT_REGISTERED	함수로 명시한 항목이 어떤 애플리케이션에서도 등록되지 않았다.
GTK_RECENT_MANAGER_ERROR_READ	최근 사용한 자원 파일을 읽는 데 실패하였다.
GTK_RECENT_MANAGER_ERROR_WRITE	최근 사용한 자원 파일을 쓰는 데 실패하였다.
GTK_RECENT_MANAGER_ERROR_UNKNOWN	다른 타입의 오류가 발생하였다. 상세한 정보는 error->message를 참고한다.
표 E-14. GtkRecentManagerError 열거 값	

Notes

# FoundationsofGTKDevelopment:Appendix F

부록 F 연습문제 해답과 힌트

## 연습문제 해답과 힌트

마지막 부록에서는 이 책에 실린 연습문제의 해답을 신겠지만 전체적인 코드는 [www.gtkbook.com](http://www.gtkbook.com)에서 다운로드할 수 있다. 이 부록을 통해 연습문제에서 막히는 부분이 생기면 코드를 살펴보기 전에 문제를 해결할 수 있는 도구를 제공하고자 한다. 그리고 나서 다운로드한 해답을 참고하여 연습문제에서 소개한 애플리케이션을 필자가 어떻게 구현하는지 확인하도록 한다.

■ **Note** 연습문제가 복잡해지면서 자신이 구현한 해답과 필자의 해답이 상당히 다를 수도 있다. 애플리케이션이 성공적으로 작동한다 하더라도 다운로드한 해답과 자신의 해답을 비교해보길 바란다.

### 연습문제 2-1. 이벤트와 프로퍼티 사용하기

이 연습문제의 해답은 제 2장에 걸쳐 소개한 연습문제들과 매우 유사할 것이다. 우선 각자의 애플리케이션은 모든 GTK+ 애플리케이션에서 필요로 하는 다음 네 가지 기본 단계를 포함해야 한다.

1. `gtk_init()`를 이용해 GTK+ 초기화하기.
2. 최상위 수준의 `GtkWindow` 위젯 생성하기.
3. 사용자에게 `GtkWindow` 위젯 표시하기.
4. `gtk_main()`을 이용해 메인 루프로 이동하기.

네 가지 기본 단계에 더해 `GtkLabel` 위젯을 최상위 수준의 창으로 추가하는 일도 해야 한다. 이 라벨 위젯은 `gtk_label_set_selectable()`을 이용해 선택 가능하게(selectable) 설정할 수 있다. 다음으로, `GtkWindow` 위젯을 `key-press-event` 시그널로 연결해야 하는데, 이 시그널은 창에 포커스가 있을 때 사용자가 키를 누를 때마다 호출될 것이다.

■ **Note** `key-press-event`가 `GtkLabel` 위젯으로 연결되어 있다면 작동하지 않을 것이다! 라벨 위젯은 고유의 `GdkWindow`를 갖고 있지 않으므로 GDK 이벤트를 수신할 수 없음을 제 3장의 해답에서 학습할 것이다.

`key-press-event` 콜백 함수에서는 라벨이 현재 첫 번째 이름을 표시하는지 아니면 마지막 이름을 표시하는지 결정하기 위해 `g_ascii_strcasecmp()`를 사용할 수 있다. 이에 따라 창과 라벨 텍스트가 바뀌어야 한다. 애플리케이션이 계속해서 `key-press-event`를 처리할 것이라면 `FALSE`를 리턴해야 한다.

첫 애플리케이션 생성 단계에서 마지막으로 할 일은 최상위 수준의 창을 `destroy` 시그널로 연결하는 것이다. `destroy` 시그널의 콜백 함수에서 `gtk_main_quit()`을 호출하면 애플리케이션이 종료될 것이다. `delete-event`가 발생할 때마다 창이 소멸될 것임으로 `delete-event` 시그널을 사용할 필요가 없다.

### 연습문제 2-2. GObject 프로퍼티 시스템

이번 문제는 연습문제 2-1과 매우 비슷한데, 차이점이 있다면 프로퍼티를 변경하기 위해 `GObject` 라이브러리가 제공한 함수를 이용해야 한다는 것이다. 예를 들어, `main()` 함수에서 `GtkWindow` 위젯의 제목, 너비, 높이, 크기 조정 가능 기능은 `g_object_set()`을 이용해 설정되어야 한다.

뿐만 아니라 `key-press-event` 콜백 함수에서는 `g_object_get()`과 `g_object_set()`을 이용해 `GtkWindow`의 `title` 프로퍼티, `GtkLabel`의 `label` 프로퍼티와 상호작용해야 한다.

그 외에 창의 `title` 프로퍼티가 변경되면 알림을 제공하도록 지시하였다. 창을 `notify::title` 시그널로 연결하면 주어진 프로퍼티로 된 값을 감시하여 이를 확보할 수 있다. 이후 `g_message()`를 이용해 새로운 창 제목을 표준 출력으로 출력할 수 있다. Terminal 에뮬레이터로부터 애플리케이션을 시작할 경우 terminal 출력에서 메시지



를 확인할 수 있을 것이다.

### 연습문제 3-1. 다수의 컨테이너 사용하기

이번 연습문제는 GtkNotebook, GtkVBox, GtkHBox 를 포함해 제 3장에서 다룬 다양한 컨테이너 위젯을 사용하는 경험이 되었을 것이다. 이번에는 이 세 가지 컨테이너를 하나씩 분석해보자.

GtkNotebook 컨테이너 위젯에는 4개의 탭이 포함되어 있어야 한다. 노트북의 각 탭은 라벨 위젯과 자식 위젯에 연관된다. gtk\_notebook\_append\_page() 함수를 이용하면 새로운 페이지를 노트북으로 추가할 수 있다. 각 탭은 clicked 시그널로 연결된 GtkButton 위젯을 포함해야 한다. 버튼을 클릭하면 노트북은 다음 페이지로 넘어가고, 마지막 페이지에 도달하면 처음으로 되돌아온다(wrap around). 각 clicked 시그널을 동일한 콜백 함수로 연결하면 이를 구현할 수 있다.

다운로드한 해답에 소개된 next\_tab()이라고 불리는 콜백 함수에서는 먼저 페이지 번호를 확인할 필요가 있다. 페이지 번호가 3보다 적으면 gtk\_notebook\_next\_page()를 호출해 다음 페이지로 넘어가면 된다. 그 외의 경우 gtk\_notebook\_set\_page()를 이용해 페이지 번호를 0으로 설정할 수 있다. 노트북의 이전 페이지로 이동 시에도 동일한 방법을 이용하면 된다.

다음 컨테이너 위젯은 두 개의 버튼이 있는 GtkHBox다. 첫 번째 버튼을 누르면 GtkNotebook 컨테이너에서 이전 페이지로 이동할 것이다. 앞서 언급하였듯 다음 페이지로 이동하기 위해 사용했던 방법을 역으로(reversed) 이용하면 이전 페이지로 이동할 수 있다. 두 번째 버튼을 클릭하면 창을 닫고 애플리케이션을 종료한다. 이러한 버튼들을 gtk\_box\_pack\_end()를 이용해 패킹하면 수평 박스의 좌측면 대신 우측면에 나타난다. 애플리케이션에서 마지막 컨테이너 위젯인 GtkVBox 위젯은 GtkNotebook과 GtkHBox 위젯을 보유해야 한다. 수직 박스를 최상위 수준의 GtkWindow 위젯으로 패킹하면 애플리케이션의 사용자 인터페이스가 완성된다.

### 연습문제 3-2. 좀 더 복잡한 컨테이너

이번 연습문제 해답은 바로 앞에서 소개한 해답과 매우 비슷하다. 차이점이 있다면 gtk\_notebook\_set\_show\_tabs()를 이용해 GtkNotebook 탭을 숨겨야 한다는 점이다. 이후 GtkExpander 컨테이너를 각 GtkButton 위젯과 노트북 탭 사이에 위치시켜야 한다. 이는 각 탭에서 발견되는 버튼을 개발자가 표시하고 숨길 수 있도록 해준다. 익스팬더의 라벨을 이용해서 현재 어떤 탭이 표시되는지 확인할 수도 있다.

앞의 해답과 비교 시 또 다른 점은 노트북과 수평 박스를 패킹하는 데에 GtkVBox 위젯을 사용하지 않고 GtkVPaned 위젯을 사용해야 한다는 점이다. 이 컨테이너는 두 위젯 간 수평 구분자를 드래그함으로써 두 자식에 할당되는 공간의 재분배를 가능하게 해준다.

### 연습문제 4-1. 파일 재명명하기

이번 연습문제에서는 제 4장에서 학습한 여러 위젯을 사용해야 하는데, 스톱 버튼 GtkEntry와 GtkFileChooserButton을 예로 들 수 있겠다. 이 연습문제의 목적은 GLib에서 만든 함수를 이용해 선택된 파일을 재명명하는 기능을 사용자에게 제공하는 데에 있다.

처음으로 할 일은 자신의 사용자 인터페이스를 구성해야 하는데, 세 개의 상호작용 위젯을 포함하도록 해야 한다. 첫 번째 위젯은 파일 선택자 버튼으로, gtk\_file\_chooser\_button\_new()를 이용해 생성된다. 선택자의 액션은 GTK\_FILE\_CHOOSER\_ACTION\_OPEN으로 설정되어야 한다. 그러면 개발자는 하나의 파일만 선택할 수 있다. gtk\_file\_chooser\_set\_current\_folder() 함수를 이용하면 파일 선택자 버튼의 현재 폴더를 g\_get\_home\_dir()에서 찾을 수 있는 사용자의 홈 디렉터리로 설정한다.

GtkFileChooserButton 위젯은 selection-changed 시그널로 연결되어야 한다. 그 콜백 함수에서 개발자는 파일이 재명명 가능한 파일인지 검사할 필요가 있다. 이러한 검증은 g\_access()라고 불리는 GLib 함수를 통해 이루어진다. 애플리케이션에서 아래와 같은 호출을 이용할 수 있겠다.

```
gint mode = g_access (fn, W_OK);
```

파일로 접근할 수 없거나 현재 사용자가 파일을 변경한 경우 `GtkEntry`와 `GtkButton` 위젯이 비활성화되어야 한다. 이는 `mode`와 같은 `opposite Boolean` 값을 `gtk_widget_set_sensitive()`로 전송함으로써 이루어진다.

다음으로 사용할 위젯은 `GtkEntry`로, 사용자가 위젯에 대해 새로운 이름을 입력하도록 해준다. 파일명을 재명명하면 `GtkFileChooserButton`의 위치 뒤에 붙을 것이기 때문에 위치를 제외한 파일명이 새 이름에 해당한다. 마지막 위젯, `GtkButton`을 클릭하면 재명명 함수가 호출될 것이다.

버튼의 콜백 함수에서 개발자는 파일 선택자 버튼으로부터 현재 파일과 위치를 검색할 필요가 있다. `GtkEntry` 위젯의 내용을 비롯해 위치를 이용하면 파일에 대한 새로운 절대 경로를 만들 수 있다. 마지막으로 해야 할 일은 `g_rename()` 함수를 이용해 파일을 재명명하는 것이다. `g_rename()`이 작동하려면 `<glib/gstdio.h>`를 포함시켜야 함을 주목하라!

### 연습문제 4-2. 스피너 버튼과 스케일

이번 문제는 앞의 연습문제와는 매우 상이하며, `GtkCheckButton`, `GtkSpinButton`, `GtkHScale` 위젯을 연습하도록 해준다. 체크 버튼이 활성화되면 스피너 버튼과 수평적 스케일의 값이 동기화될 것이다. 활성화되지 않으면 이 값들은 서로 무관하게 이동할 수 있다.

이를 구현하기 위해서는 먼저 각 범위 위젯에 하나씩, 총 두 개의 동일한 조정(adjustment)을 생성해야 한다. 이 연습문제에서는 애플리케이션이 시작되면 토글 버튼이 활성화되므로 값이 즉시 동기화될 것이다.

다음으로 각 범위 위젯을 `value-changed` 시그널에 대한 동일한 콜백 함수로 연결한다. 이 함수에서는 먼저 스피너 버튼과 스케일의 현재 값을 검색한다. 토글 버튼이 활성화되면 이 값들이 비교된다. `value-changed` 시그널이 반복되어 발생하지 않도록 값이 다를 때만 액션을 취한다.

마지막으로, 콜백 함수는 어떤 타입의 위젯이 새로운 값을 보유하는지 알아내기 위해 `GTK_IS_SPIN_BUTTON()`을 이용할 수 있다. 테스트 결과를 바탕으로 다른 위젯은 새로운 값을 부여 받아야 한다.

### 연습문제 5-1. 파일 선택자 대화상자 구현하기

제 5장에 유일하게 실린 연습문제에서는 `GtkFileChooserWidget` 위젯을 `GtkDialog` 위젯에 포함시킴으로써 네 가지 타입의 파일 선택자 대화상자를 재생성하는 것이 목적이다. 각 결과는 간단히 표준 출력으로 인쇄할 수 있다.

애플리케이션 메인 창에는 `GtkFileChooser` 액션 타입마다 하나씩, 총 4개의 버튼이 포함되어 있을 것이며, `GTK_FILE_CHOOSER_ACTION_OPEN` 액션을 이용하면 다수의 파일을 선택할 수 있을 것이다. 이 버튼들은 수직 박스에 패킹한 다음 최상위 수준 창으로 패킹 가능하다.

각 콜백 함수는 동일한 패턴을 따른다. 먼저 `GtkDialog` 위젯을 생성하고, `gtk_box_pack_start()`를 이용해 대화상자의 vbox member를 패킹함으로써 대화상자의 액션 영역 위에 `GtkFileChooserWidget`을 패킹한다.

그 다음으로 `gtk_dialog_run()`을 이용해 대화상자를 실행한다. 리턴 결과가 액션의 수락과 연관된 응답일 경우 개발자는 발생하는 결과를 `g_print()`를 호출해 출력해야 한다. 가령 개발자는 파일이 저장될 것이라던지, 폴더가 생성되었다던지, 파일이 열릴 것이라던지, 폴더가 선택되었다던지 등의 정보를 사용자에게 알려야 할 것이다. `GTK_FILE_CHOOSER_ACTION_OPEN` 액션의 경우 선택된 모든 파일을 출력해야 한다.

### 연습문제 6-1. 파일로 작업하기

이번 연습문제에서는 제 6장에서 파일 조작에 관해 학습한 내용을 앞 장에 걸쳐 학습한 위젯과 통합시키는 데에 목적이 있다. 이 연습문제에서 사용되는 사용자 인터페이스는 3개의 위젯, 즉 `GtkEntry`, `GtkFileChooserButton`, `GtkButton`을 포함해야 한다.

`GtkEntry` 위젯은 사용자가 시스템 상에 파일로 저장시킬 한 행의 텍스트를 입력하도록 허용할 것이다. 파일의 위치는 `GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER`의 액션을 이용해 `GtkFileChooserButton` 위젯에서 선택된다. 마지막으로 `GtkButton` 위젯을 클릭하면 파일을 저장하기 시작할 것이다. 다운로드한 해답에서 사용자가 버튼을 클릭하면 선택된 위치에 `arbitrary_file` 이란 파일로 텍스트가 저장될 것이다.

버튼의 콜백 함수에서 개발자는 선택된 위치로부터 파일 경로를 비롯해 원하는 파일명을 만들 수 있다. 그리고 나서 `g_file_set_contents()`를 이용해 `GtkEntry` 위젯의 내용을 파일로 저장 가능하다. 파일을 쓸 때 오류가 발생하면 제 6장에서 다룬 메시지 보고 시스템을 이용해 사용자에게 보고해야 한다. 예를 들어, 사용자가 선택된 위치로 쓰기 접근성을 갖고 있지 않다면 쓰기 연산은 실패할 것이다.

### 연습문제 6-2. Timeout 함수

이 연습문제에서는 타이머를 생성하기 위해 `timeout` 함수를 이용하는데 그 과정이 매우 간단하다. 먼저 두 개의 위젯을 생성해야 하는데, 현재 계수를 출력하는 `GtkLabel`과 클릭 시 계수를 0으로 리셋하는 `GtkButton`이 해당한다. `Timeout`은 다음과 같이 `g_timeout_add_full()`을 이용해 생성되어야 한다.

```
g_timeout_add_full (G_PRIORITY_DEFAULT, 1000,
                   (GSourceFunc) timeout_function,
                   (gpointer) widget, NULL);
```

위의 `timeout` 함수는 1,000 밀리초마다 `timeout_function()`을 호출할 것이다. 함수 내에서 초는 증가하고 라벨은 업데이트된다.

두 번째 작업은 `GTimer`를 이용해 타이머 생성을 재구현하는 일이다. 앞의 해답에 두 번째 라벨을 위치시켜 계수를 비교할 수 있도록 해보라. `Timeout`은 정확한 시간을 유지하는 `GTimer`보다 느리게 계수함을 눈치챌 것이다. 이는 `timeout_function()`이 1,000 밀리초마다 호출되는 데다가 함수를 실행하는 데에 소요되는 시간까지 합쳐지기 때문이다! `Timeout`의 다음 계수 시간(counting period)은 앞의 호출이 완료될 때까지 시작하지 않으므로 함수 호출의 중복을 피할 수 있다. 연습문제에서 알 수 있듯이 시간을 추적하는 데에는 절대 `timeout` 함수를 사용해선 안 되는 이유다.

### 연습문제 7-1. 텍스트 에디터

이번 연습문제에서는 처음으로 텍스트 에디터 애플리케이션을 살펴볼 것이다. 여기서는 텍스트 에디터의 기능을 모두 구현할 것을 요구한다.

**Note** 다운로드한 해답에는 텍스트 에디터의 기본 기능만 포함되어 있다. 문제가 생기면 작업을 시작할 수 있도록 준비시키는 것이 목적이다. 하지만 제공된 해답을 능가하여 텍스트 에디터 구현을 확장할 것을 권한다.

텍스트 에디터에는 다수의 콜백 함수가 구현되고 있다. 새로운 파일을 생성하고, 기존 파일을 열고, 파일을 저장하고, 선택된 텍스트의 자름, 복사, 붙여넣기를 실행하고, 문서에서 텍스트를 검색하는 기능을 예로 들 수 있겠다.

새로운 문서를 생성하기 위해서는 애플리케이션이 `GtkMessageDialog` 위젯과 계속 작업할 것인지 사용자에게 물어보아야 한다. 사용자가 계속 사용하기로 선택하면 다운로드한 해답에서와 같이 `GtkTextBuffer` 객체를 제거하고 대화상자를 소멸한다. 그렇지 않으면 대화상자만 소멸된다.

제공된 해답에서는 문서를 열면 사용자에게 확인을 요청하지 않는다는 사실을 눈치챌 것인데, `GtkFileChooserDialog` 위젯에서 연산을 취소하는 일은 쉽기 때문이다. 파일 선택자 대화상자에는 `GTK_FILE_CHOOSER_ACTION_OPEN`이란 액션 타입이 있다. `g_file_get_contents()`를 이용하면 파일을 선택 시 그 내용이 읽히고 텍스트 버퍼로 적힌다. 연습문제 해답에서 파일을 저장하기 위해 버튼을 누를 때마다 새

로운 파일명을 묻는다. 텍스트를 선택된 파일로 저장하기 위해서는 `g_file_set_contents()`를 호출한다.

클립보드 함수는 제 7장의 클립보드 예제에서 제공된 함수와 매우 유사하다. 자르기, 복사하기, 붙여넣기 액션에 내장된 텍스트 버퍼 함수를 사용한다. 이들은 기본 클립보드, `GDK_SELECTION_CLIPBOARD`에서 실행된다.

마지막 콜백 함수는 현재 텍스트에서 대·소문자에 민감한 문자열을 검색한다. 이에 사용된 해답은 제 7장의 리스팅 7-6에 표시된 함수와 유사하므로 추가 정보는 본문의 내용을 참고한다.

### 연습문제 8-1. 파일 브라우저

이번 연습문제에서는 매우 간단한 파일 브라우저를 구현할 것이다. 이는 사용자가 시스템의 파일 구조체를 살펴보고 파일과 폴더를 구별할 수 있도록 해준다. 여기서는 `GtkTreeView` 위젯의 사용 경험을 쌓는 것이 목적이다. 이는 13장에서 좀 더 기능적인 파일 브라우저로 더 많이 확장될 것이다.

첫 번째 단계는 하나의 열만 포함하는 트리 뷰를 설정하는 것이다. 이 열은 두 개의 셀 렌더러를 포함할 것인데, 하나는 `GdkPixbuf`용이고 나머지 하나는 파일명이나 폴더명에 사용될 것이므로 제 8장에서 논한 트리 뷰 열 생성 방법을 확장시켜 사용해야 할 것이다. 첫 번째 셀 렌더러는 `GtkCellRendererPixbuf`를, 두 번째는 `GtkCellRendererText`를 이용해야 한다.

트리 모델인 `GtkListStore`는 `GDK_TYPE_PIXBUF`와 `G_TYPE_STRING` 타입으로 된 두 개의 열과 함께 생성된다. 리스트 저장소(list store)는 트리 뷰로 추가된 다음에 `g_object_unref()`를 이용해 참조해제해야만 트리 뷰 위젯과 함께 소멸될 것이다.

다운로드한 해답에서는 트리 뷰가 생성된 다음에 `populate_tree_model()` 함수가 호출되어 시작 시 파일 시스템의 루트 폴더를 표시한다. 파일 브라우저가 표시한 현재 경로는 `current_path`라는 전역적 연결 리스트에 저장된다. 리스트가 비어 있다면 루트 폴더가 표시된다. 비어 있지 않은 경우 리스트의 내용에서 경로가 만들어지고, 트리 모델에 `".."` 디렉터리 엔트리가 추가된다.

그 다음으로 `GDir`을 이용해 디렉터리의 내용을 살펴보고, 각 파일이나 폴더를 트리 모델로 추가한다. 각각이 파일인지 폴더인지 확인하기 위해 `G_FILE_TEST_IS_DIR`과 함께 `g_file_test()`를 이용하면 결과에 따라 올바른 아이콘이 표시된다.

마지막 단계는 디렉터리 이동을 처리하는 일인데, `GtkTreeView`의 `row-activated` 시그널이 사용된다. 선택내용이 `".."` 엔트리인 경우 경로의 마지막 요소가 제거되고, 트리 모델이 다시 채워진다(`repopulated`). 그 외의 경우는 현재 위치와 선택내용에서 새로운 경로가 만들어진다. 선택내용이 폴더라면 트리 모델은 새로운 디렉터리에서 다시 채워진다. 반대로 선택내용이 파일이라면 액션이 무시되고 어떤 일도 실행되지 않는다.

### 연습문제 9-1. 툴바

이번 연습문제는 연습문제 7-1을 약간 수정하여 면을 따라 위치한 버튼을 `GtkUIManager`를 이용해 생성된 `GtkToolbar`로 대체한다. 아래 UI 파일을 이용해 툴바를 이용할 수 있다.

```
<ui>
  <toolbar name="Toolbar">
    <toolitem name="FileNew" action="New"/>
    <toolitem name="FileOpen" action="Open"/>
    <toolitem name="FileSave" action="Save"/>
    <separator/>
    <toolitem name="EditCut" action="Cut"/>
    <toolitem name="EditCopy" action="Copy"/>
    <toolitem name="EditPaste" action="Paste"/>
  </toolbar>
</ui>
```

애플리케이션에서 다음으로 해야 할 일은 UI 파일 내 각 툴바 항목과 연관될 `GtkActionEntry` 객체의 배열을 생성하는 것이다. 이러한 액션들은 `GtkActionGroup` 객체에서 조직되고, 툴바는 `GtkUIManager` 객체를 이용해 생성된다. 나머지 텍스트 에디터 구현 부분은 연습문제 7-1과 동일하다.

### 연습문제 9-2. 메뉴 표시줄

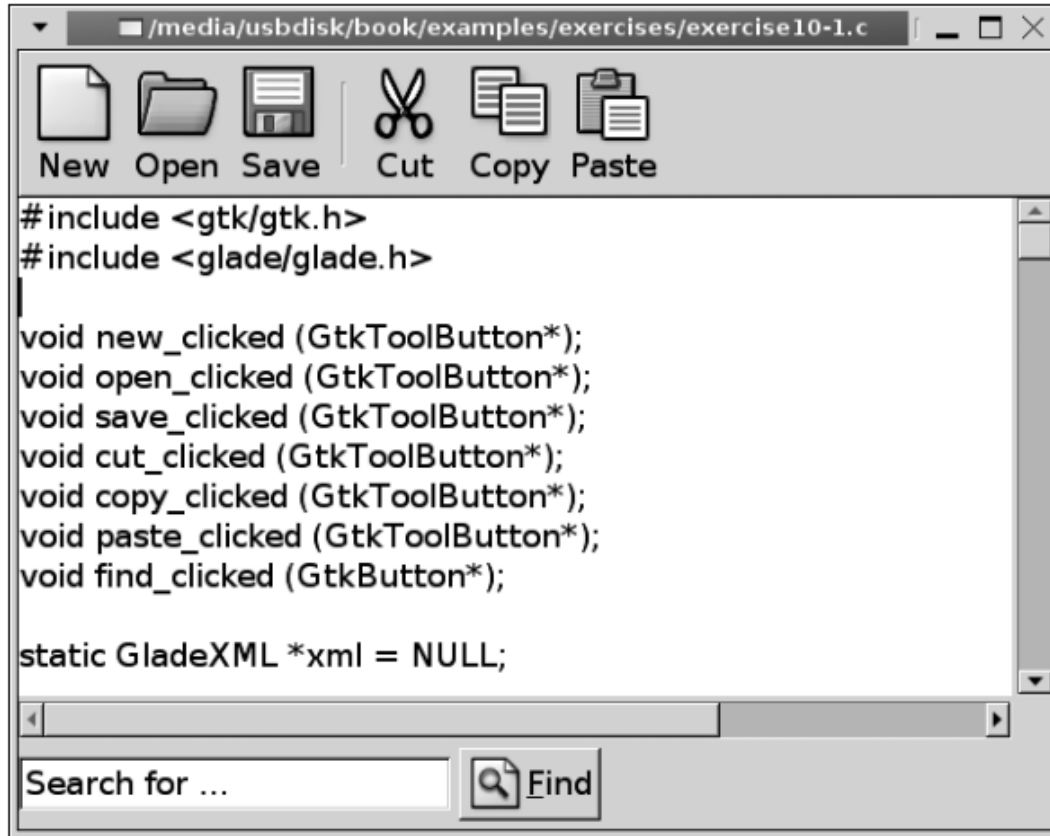
이 또한 연습문제 7-1을 수정하여 면을 따라 위치한 버튼을 `GtkUIManager`로 생성한 `GtkMenuBar` 위젯으로 대체한다. 툴바는 아래 UI 파일을 이용해 생성할 수 있다.

```
<ui>
  <menubar name="MenuBar">
    <menu name="FileMenu" action="File">
      <menuitem name="FileNew" action="New"/>
      <menuitem name="FileOpen" action="Open"/>
      <menuitem name="FileSave" action="Save"/>
    </menu>
    <menu name="EditMenu" action="Edit">
      <menuitem name="EditCut" action="Cut"/>
      <menuitem name="EditCopy" action="Copy"/>
      <menuitem name="EditPaste" action="Paste"/>
    </menu>
  </menubar>
</ui>
```

애플리케이션에서 다음으로 해야 할 일은 UI 파일 내 각 툴바 항목과 연관될 `GtkActionEntry` 객체의 배열을 생성하는 것이다. 이러한 액션들은 `GtkActionGroup` 객체에서 조직되고, 메뉴 표시줄은 `GtkUIManager` 객체를 이용해 생성된다. 나머지 텍스트 에디터 구현 부분은 연습문제 7-1과 동일하다.

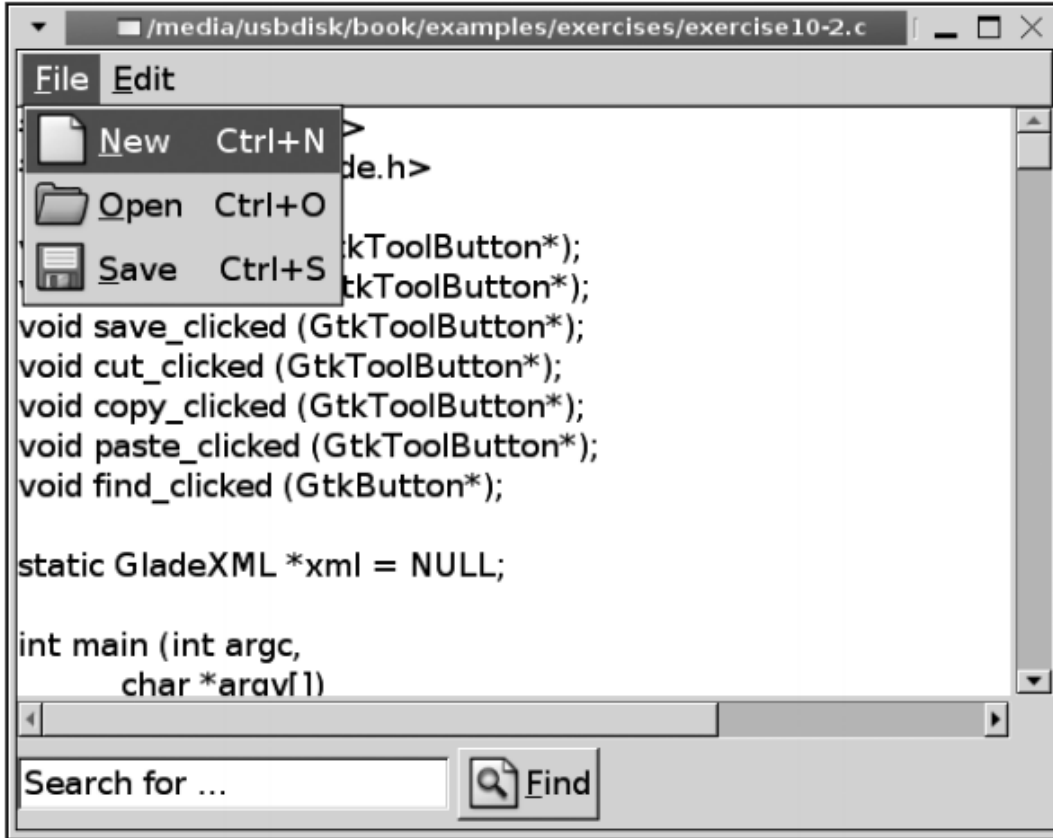
### 연습문제 10-1. Glade 텍스트 에디터

이번에도 연습문제 7-1을 확장시키되 사용자 인터페이스 전체를 Glade에서 다시 디자인하도록 요구한다. 버튼을 사용하는 대신 텍스트 편집 함수를 위한 툴바를 구현해야 한다. 다음으로 Libglade를 이용해 그래픽 사용자 인터페이스를 로딩하고 필요한 시그널을 연결할 수 있다. 그림 F-1에는 툴바를 이용해 생성한 애플리케이션의 스크린샷을 실었다.



### 연습문제 10-2. 메뉴가 있는 Glade 텍스트 에디터

이 또한 연습문제 7-1을 확장시켜 사용자 인터페이스를 모두 Glade에서 다시 디자인하도록 요한다. 하지만 이번에는 버튼을 이용하는 대신 텍스트 편집 함수를 위한 메뉴 표시줄을 구현해야 한다. 그리고 나서 Libglade를 이용해 그래픽 사용자 인터페이스를 로딩하고 필요한 시그널을 연결한다. 그림 F-2에는 메뉴 표시줄을 이용하는 애플리케이션의 스크린샷이 실려 있다.



### 연습문제 11-1. MyMarquee 확장하기

이번 연습문제에서는 제 11장에서 만든 MyMarquee 위젯을 확장할 것이다. 이번 절에는 연습문제에서 필요로 하는 추가 기능을 구현하는 것과 관련된 팁이 많이 포함되어 있다.

첫 번째 확장은 위젯 주위에 테두리를 추가하는 것이다. 이는 gdk\_draw\_rectangle() 함수를 이용하면 된다. 아래 함수는 주어진 width와 height로 된 직사각형의 테두리를 (x,y)에 상단 좌측 모서리가 위치하도록 그린다.

```

void gdk_draw_rectangle (GdkDrawable *drawable,
                        GdkGC *gc,
                        gboolean filled,
                        gint x,
                        gint y,
                        gint width,
                        gint height);
  
```

그 다음으로 여러 개의 메시지를 스크롤하는 기능을 제공하고자 한다. 이를 위해서는 우선 메시지를 private한 연결 리스트로 저장할 필요가 있다. 다음으로 메시지를 추가 및 저장하기 위한 함수들을 제공해야 한다. 메시지가 위젯의 경계를 벗어나도록 스크롤되면 리스트에서 다음 메시지의 스크롤이 시작해야 한다.

그리고 나면 메시지를 왼쪽 또는 오른쪽 방향으로 스크롤할 수 있는 기능을 제공할 필요가 있다. 이는 개발자의 슬라이드 함수에서 처리되는데, 함수가 호출될 때마다 올바른 방향으로 speed 픽셀을 이동할 것이다. 여기서 주의해야 할 점은 메시지가 스크롤 방향에서 명시한 방향으로 위젯을 벗어나서 스크롤될 수 있다는 점이

다.

마지막으로, 마우스 포인터가 위젯 위에 있으면 메시지는 스크롤을 중단해야 한다. 이는 기본 `enter-notify-event`와 `leave-notify-event` 콜백 함수를 오버라이드하여 실행된다. 메시지가 스크롤되어야 하는지 여부를 명시하기 위해서는 `boolean` 플래그를 사용해야 한다. 이를 호출하는 동안 메시지의 제거 여부를 결정하려면 `my_marquee_slide()`를 확인하면 된다.

### 연습문제 12-1. 전체 텍스트 에디터 생성하기

마지막으로 제시된 텍스트 에디터 연습문제는 10-1의 확장에 해당한다. 여기서서는 추가로 두 개의 기능을 더 해야 한다. 첫 번째는 인쇄 지원으로, 사용자가 `GtkTextBuffer` 위젯에 현재 텍스트를 인쇄하도록 허용하는 기능이다. 이 연습문제에 다운로드한 해답에 실린 인쇄 지원은 제 12장에 소개된 예제와 매우 유사하므로 해답의 도출 과정에 관한 추가 정보는 본문에 실린 설명을 확인하도록 한다.

두 번째 추가 기능은 `Open` 툴바 항목에 대한 최근 파일 선택자 메뉴에 해당한다. 이를 생성하기 위해서는 `Open` 툴바 항목을 `GtkMenuItem` 위젯으로 변환해야 한다. `gtk_recent_manager_get_default()`로 생성된 기본 최근 관리자를 이용하면 최근 파일을 제공할 수 있다. 그 다음, `gtk_recent_choose_menu_new_for_manager()`를 이용해 최근 파일 선택자 메뉴를 생성할 수 있다. 이 메뉴는 `Open` 메뉴 툴 버튼의 `GtkMenuItem`로 추가되어야 한다. 마지막으로 `selection-done` 시그널을 이용해 어떤 메뉴가 선택되었는지, 어떤 파일을 열어야 하는지 알아낼 수 있다.

## Notes



# 문서 출처 및 기여자

**FoundationsofGTKDevelopment:AbouttheAuthor** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4634> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Acknowledgments** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4621> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Introduction** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4630> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 01** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4641> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 02** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4646> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 03** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4650> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 04** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4671> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 05** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4683> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 06** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4685> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 07** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4697> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 08** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4714> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 09** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4724> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 10** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4854> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 11** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4870> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 12** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4937> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Chapter 13** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4973> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix A** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4975> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix B** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4977> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix C** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=4978> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix D** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=5118> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix E** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=5119> 기여자: Onionmixer

**FoundationsofGTKDevelopment:Appendix F** 출처: <http://trans.onionmixer.net/mediawiki/index.php?oldid=5122> 기여자: Onionmixer









**image:appendix\_D\_stock\_exit\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_exit\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_exit_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_redo\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_redo\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_redo_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_refresh\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_refresh\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_refresh_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_remove\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_remove\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_remove_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_revert\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_revert\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_revert_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_save\_as\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_save\\_as\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_save_as_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_gtk-select-all.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_gtk-select-all.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_gtk-select-all.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_colorselector\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_colorselector\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_colorselector_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_font\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_font\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_font_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_sort\_ascending\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_sort\\_ascending\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_sort_ascending_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_sort\_descending\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_sort\\_descending\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_sort_descending_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_spellcheck\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_spellcheck\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_spellcheck_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_stop\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_stop\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_stop_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_text\_strikethrough\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_text\\_strikethrough\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_text_strikethrough_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_undelete\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_undelete\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_undelete_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_text\_underline\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_text\\_underline\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_text_underline_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_undo\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_undo\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_undo_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_text\_unindent\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_text\\_unindent\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_text_unindent_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_yes\_20.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_yes\\_20.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_yes_20.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_zoom\_1\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_zoom\\_1\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_zoom_1_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_zoom\_fit\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_zoom\\_fit\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_zoom_fit_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_zoom\_in\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_zoom\\_in\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_zoom_in_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:appendix\_D\_stock\_zoom\_out\_24.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix\\_D\\_stock\\_zoom\\_out\\_24.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:Appendix_D_stock_zoom_out_24.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:FoundationsofGTKDevelopment\_Image\_Appendix\_f-1.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:FoundationsofGTKDevelopment\\_Image\\_Appendix\\_f-1.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:FoundationsofGTKDevelopment_Image_Appendix_f-1.png) 라이선스: 알 수 없음 기여자: Onionmixer

**image:FoundationsofGTKDevelopment\_Image\_Appendix\_f-2.png** 출처: [http://trans.onionmixer.net/mediawiki/index.php?title=파일:FoundationsofGTKDevelopment\\_Image\\_Appendix\\_f-2.png](http://trans.onionmixer.net/mediawiki/index.php?title=파일:FoundationsofGTKDevelopment_Image_Appendix_f-2.png) 라이선스: 알 수 없음 기여자: Onionmixer